GRLIB IP Core User's Manual

Oct 2025, Version 2025.2

Frontgrade Gaisler AB
Kungsgatan | SE-411 19 | Goteborg | Sweden
+46 31 7758650 | frontgrade.com/gaisler

Table of contents

1 Introduction 7
2 AHB2AHB - Uni-directional AHB/AHB bridge 22
3 AHB2AXIB - AHB to AXI Bridge 40
4 AHBM2AXI - AHB Master to AXI Adapter 49
5 AHBBRIDGE - Bi-directional AHB/AHB bridge 57
6 AHBCTRL - AMBA AHB controller with plug&play support 62
7 AHBJTAG - JTAG Debug Link with AHB Master Interface 71
8 AHBLITM2AHBM - AHB-LITE to AHB MASTER 77
9 AHBRAM - Single-port RAM with AHB interface 81
10 AHBDPRAM - Dual-port RAM with AHB interface 84
11 AHBROM - Single-port ROM with AHB interface 87
12 AHBSTAT - AHB Status Registers 90
13 AHBTRACE - AHB Trace buffer 95
14 AHBUART- AMBA AHB Serial Debug Interface 103
15 AMBAMON - AMBA Bus Monitor 109
16 APBCTRL - AMBA AHB/APB bridge with plug&play support 115
17 APBPS2 - PS/2 host controller with APB interface 120
18 APBUART - AMBA APB UART Serial Interface 130
19 APBUART16550 - AMBA APB UART 16550 Serial Interface 142
20 APBVGA - VGA controller with APB interface 157
21 AXI2AHB - AXI4 to AHB2 Bridge 161
22 AXI4_RESIZE - AXI4 Transaction Resize 165
23 AXIS_BUFFER - AXI4-Stream Buffer 171
24 AXIS_GEARBOX - AXI4-Stream Gearbox 174
25 BCH_DECODER - Highly configurable systematic BCH decoder 177
26 BCH_ENCODER - Highly configurable systematic BCH encoder 182
27 CAN_OC - GRLIB wrapper for OpenCores CAN Interface core 187
28 CLKGEN - Clock generation 207
29 DDRSPA - 16-, 32- and 64-bit DDR266 Controller 231
30 DDR2SPA - 16-, 32- and 64-bit Single-Port Asynchronous DDR2 Controller 245
31 DIV32 - Signed/unsigned 64/32 divider module 264
32 DSU3 - LEON3 Hardware Debug Support Unit 267
33 DSU4 - LEON4 Hardware Debug Support Unit 283
34 FTADDR - Autonomous DDR2/DDR3/DDR4 Controller with EDAC 301

35 FTAHBRAM - On-chip SRAM with EDAC and AHB interface 367
36 FTMCTRL - 8/16/32-bit Memory Controller with EDAC 375
37 FTSDCTRL - 32/64-bit PC133 SDRAM Controller with EDAC 407
38 FTSRCTRL - Fault Tolerant 32-bit PROM/SRAM/IO Controller 420
39 FTSRCTRL8 - 8-bit SRAM/16-bit IO Memory Controller with EDAC 439
40 GPTIMER - General Purpose Timer Unit 455
41 GR1553B - MIL-STD-1553B / AS15531 Interface 465
42 GRTIMER - General Purpose Timer Unit 508
43 GRACECTRL - AMBA System ACE Interface Controller 509
44 GRADCDAC - ADC / DAC Interface 514
45 GRAES - Advanced Encryption Standard 527
46 GRAES_DMA - Advanced Encryption Standard with DMA 533
47 GRCAN - CAN 2.0 Controller with DMA 542
48 GRCANFD - CAN Flexible Data-Rate Controller with CANOpen support 566
49 GRCLKGATE / GRCLKGATE2X - Clock gating unit 604
50 GRDMAC - DMA Controller with internal AHB/APB bridge 611
51 GRDMAC2 - DMA Controller 632
52 GRECC - Elliptic Curve Cryptography 664
53 GRETH - Ethernet Media Access Controller (MAC) with EDCL support 679
54 GRETH_GBIT - Gigabit Ethernet Media Access Controller (MAC) w. EDCL 699
55 GRFIFO - FIFO Interface 724
56 GRFPU - High-performance IEEE-754 Floating-point unit 748
57 GRFPC - GRFPU Control Unit 755
58 GRFPU Lite - IEEE-754 Floating-Point Unit 757
59 GRLFPC - GRFPU Lite Floating-point unit Controller 760
60 GRGPRBANK - General Purpose Register Bank 762
61 GRGPIO - General Purpose I/O Port 765
62 GRGPREG - General Purpose Register 776
63 GRHSSL - SpaceFibre and WizardLink controller with DMA engine 779
64 GRIOMMU - AHB/AHB bridge with access protection and address translation 789
65 GRIOMMU2 - AHB/AHB bridge with RISC-V IOMMU & IOPMP capabilities 834
66 GRLSEDC- Lattice FPGA soft error detection interface 880
67 GRPCI2 - 32-bit PCI(Initiator/Target) / AHB(Master/Slave) bridge 887
68 GRPPSTIMER - PPS TIMER 917
69 GRPULSE - General Purpose Input Output 935

70 GRPWM - Pulse Width Modulation Generator 942
71 GRRT - MIL-STD-1553B / AS15531 Remote Terminal Back-End 955
72 GRSCRUB - AMD/Xilinx external FPGA scrubber controller 962
73 GRSHYLOC - Wrapper for the SHYLOC compressor with DMA engine 992
74 GRSPFI - SpaceFibre codec with DMA, RMAP target and Bus Master Interface 1003
75 GRSPFI_CODEC - SpaceFibre encoder/decoder 1056
76 GRSPW - SpaceWire codec with AHB host Interface and RMAP target 1071
77 GRSPW2 - SpaceWire codec with AHB host Interface and RMAP target 1115
78 GRSPW2_GEN - GRSPW2 wrapper with Std_Logic interface 1189
79 GRSPW2_PHY - GRSPW2 Receiver Physical Interface 1196
80 GRSPW_CODEC - SpaceWire encoder-decoder 1202
81 GRSPW_CODEC_GEN - GRSPW_CODEC wrapper with Std_Logic interface 1220
82 GRSPWROUTER - SpaceWire router 1227
83 SPWTDP - SpaceWire - Time Distribution Protocol 1317
84 GRSRIO - Serial RapidIO endpoint with AHB or AXI4 bus master interface 1348
85 GRSYSMON - AMBA Wrapper for Xilinx System Monitor 1404
86 GRUSBDC - USB Device controller 1412
87 GRUSB_DCL - USB Debug Communication Link 1438
88 GRUSBHC - USB 2.0 Host Controller 1445
89 GRVERSION - Version and Revision information register 1462
90 GRWATCHDOG - Watchdog module with separate clock domain 1464
91 GRWIZL - WizardLink codec with DMA engine and Bus Master Interface 1469
92 I2C2AHB - I2C to AHB bridge 1497
93 I2CMST - I2C-master 1507
94 I2CSLV - I2C slave 1518
95 IRQMP - Multiprocessor Interrupt Controller 1526
96 IRQ(A)MP - Multiprocessor Interrupt Controller with extended ASMP support 1536
97 L2C - Level 2 Cache controller 1551
98 L2C-Lite - Level 2 Cache controller 1576
99 L3STAT - LEON3 Statistics Unit 1584
100 L4STAT - LEON4 Statistics Unit 1592
101 LEON_DSU_STAT_BASE - LEON3/4 SUBSYSTEM 1600
102 LEON3/FT - High-performance SPARC V8 32-bit Processor 1606
103 LEON4 - High-performance SPARC V8 32-bit Processor 1655
104 LEON5SYS - High-performance SPARC V8 32-bit Processor Subsystem 1699

105 LOGAN - On-chip Logic Analyzer 1770
106 MCTRL - Combined PROM/IO/SRAM/SDRAM Memory Controller 1777
107 MDIO_CTRL - MDIO Serial Bus Controller 1798
108 MEMSCRUB - AHB Memory Scrubber and Status Register 1805
109 MMA - Memory Mapped AMBA bridge 1816
110 MUL32 - Signed/unsigned 32x32 multiplier module 1821
111 MULTLIB - High-performance multipliers 1825
112 NANDFCTRL - NAND Flash Memory Controller 1827
113 NANDFCTRL2 - NAND Flash Memory Controller with Direct Memory Access 1854
114 NANDFCTRL2 PHY - NAND Flash Memory Controller PHY 1916
115 NOELVSYS - High-performance RISC-V RV64GC Processor Subsystem 1922
116 PHY - Ethernet PHY simulation model 1968
117 RGMII - Reduced Ethernet Media Access Controller 1971
118 REGFILE_3P 3-port RAM generator (2 read, 1 write) 1981
119 RSTGEN - Reset generation 1983
120 GR(2^4)(68, 60, 8, T=1) - QEC/QED error correction code encoder/decoder 1987
121 RS(24, 16, 8, E=1) - Reed-Solomon encoder/decoder 1991
122 RS(48, 32, 16, E=1+1) - Reed-Solomon encoder/decoder - interleaved 1994
123 RS(40, 32, 8, E=1) - Reed-Solomon encoder/decoder 1996
124 RS(48, 32, 16, E=2) - Reed-Solomon encoder/decoder 1999
125 SDCTRL - 32/64-bit PC133 SDRAM Controller 2003
126 SOCBRIDGE - AHB to AHB interface through byte stream 2013
127 SPI2AHB - SPI to AHB bridge 2030
128 SPICTRL - SPI Controller 2039
129 SPIMCTRL - SPI Memory Controller 2060
130 SPIMASTER - SPI Master Device 2073
131 SPISLAVE - Dual Port SPI Slave 2082
132 SRCTRL- 8/32-bit PROM/SRAM Controller 2100
133 SSRCTRL- 32-bit SSRAM/PROM Controller 2108
134 SVGACTRL - VGA Controller Core 2118
135 SYNCIOTEST - Test block for synchronous I/O interfaces 2126
136 SYNCRAM - Single-port RAM generator 2128
137 SYNCRAMBW - Single-port RAM generator with byte enables 2132
138 SYNCRAM_2P - Two-port RAM generator 2136
139 SYNCRAM_DP - Dual-port RAM generator 2141

140 SYNCRAMFT - Single-port RAM generator with EDAC 2144
141 TAP - JTAG TAP Controller 2147
142 GRTM - CCSDS/ECSS Telemetry Encoder 2151
143 GRTM_DESC - CCSDS/ECSS Telemetry Encoder - Descriptor 2178
144 GRTM_VC - CCSDS/ECSS Telemetry Encoder - Virtual Channel Generation 2181
145 GRTM_PAHB - CCSDS/ECSS Telemetry Encoder - Virtual Channel Generation Input - AMBA 2184
146 GRTM_PW - CCSDS/ECSS Telemetry Encoder - Virtual Channel Generation Input - PacketWire 2188
147 GRTM_UART - CCSDS/ECSS Telemetry Encoder - Virtual Channel Generation Input - UART 2190
148 GEFFE - CCSDS/ECSS Telemetry Encoder - Geffe Generator 2193
149 GRTMRX - CCSDS/ECSS Telemetry Receiver 2199
150 GRCE/GRCD - CCSDS/ECSS Convolutional Encoder and Quicklook Decoder 2213
151 GRTC - CCSDS/ECSS Telecommand Decoder 2218
152 TCAU - Telecommand Decoder Authentication Unit 2244
153 GRTC-COP - CCSDS/ECSS Telecommand Decoder 1
154 GRTC_HW - CCSDS/ECSS Telecommand Decoder - Hardware Commands 33
155 GRTC_UART - CCSDS/ECSS Telecommand Decoder - UART 40
156 GRTCTX - CCSDS/ECSS Telecommand Transmitter 43
157 GRCTM - CCSDS Time Manager 53
158 SPWCUC - SpaceWire - CCSDS Unsegmented Code Transfer Protocol 85
159 GRPW - PacketWire Interface 101
160 GRPWRX - PacketWire Receiver 108
161 GRPWTX - PacketWire Transmitter 117
162 PW2APB - PacketWire receiver to AMBA APB Interface 124
163 APB2PW - AMBA APB to PacketWire Transmitter Interface 130
164 AHB2PP - AMBA AHB to Packet Parallel Interface 136
165 GRRM - Reconfiguration Module 141

1 Introduction

1.1 Scope

This document describes specific IP cores provided with the GRLIB IP library. When applicable, the cores use the GRLIB plug&play configuration method as described in the 'GRLIB User's Manual'.

1.2 Other resources

There are several documents that together describe the GRLIB IP Library and Frontgrade Gaisler's IP cores:

- GRLIB IP Library User's Manual (grlib.pdf) - Main GRLIB document that describes the library infrastructure, organization, tool support and on-chip bus.
- GRLIB-FT User's Manual (grlib-ft.pdf) - Describes the FT and FT-FPGA versions of the GRLIB IP library. The document is an addendum to the GRLIB IP Library User's Manual. This document is only available in the FT and FT-FPGA distributions of GRLIB.
- GRLIB FT-FPGA Xilinx Add-on User's Manual (grlib-ft-fpga-xilinx.pdf) - Describes functionality of the Virtex5-QV and Xilinx TMRTool add-on package to the FT-FPGA version of the GRLIP IP library. The document should be read as an addendum to the 'GRLIB IP Library User's Manual' and to the GRLIB FT-FPGA User's Manual. This document is only available as part of the add-on package for FT-FPGA.
- LEON/GRLIB Configuration and Development Guide (guide.pdf) - This configuration and development guide is intended to aid designers when developing systems based on LEON/GRLIB. The guide complements the GRLIB IP Library User's Manual and the GRLIB IP Core User's Manual. While the IP Library user's manual is suited for RTL designs and the IP Core user's manual is suited for instantiation and usage of specific cores, this guide aims to help designers make decisions in the specification stage.

1.3 Reference documents

[AMBA] AMBA™ Specification, Rev 2.0, ARM IHI 0011A, 1999, Issue A, ARM Limited

[GRLIB] GRLIB IP Library User's Manual, Frontgrade Gaisler, www.gaisler.com

[AS1553] AS15531 - Digital Time Division Command/Response Multiplex Data Bus, SAE International, November 1995

[MIL1553] MIL-STD-1553B, Digital Time Division Command/Response Multiplex Data Bus, US Department of Defence, September 1978

[MIL1553N2] MIL-STD-1553B Notice 2, US Department of Defence, September 1986

[ECSS1553] Interface and Communication Protocol for MIL-STD-1553B Data Bus Onboard Spacecraft, ECSS-E-ST-50-13C. November 2008

1.4 GRLIB licensing scheme

The GRLIB IP core library can be licensed in different ways to cater for the needs of any particular project or product. Typically, a standard release contains a processor and a predefined subset of the IP core portfolio. However, it is also possible to customize a release by adding extra IP cores to the standard subset, as well as creating entirely custom releases without any processor whatsoever.

The main GRLIB licensing options are listed below:

- Type of GRLIB distribution (GPL, COM, FT-FPGA, FT).
- Processor configuration (if any).
- Optional add-on packages.

The type of GRLIB distribution and the processor configuration determine which IPs are included in the bundle. Section 1.5 lists the IPs included for each processor configuration, whereas section 1.6 covers the IPs included in each GRLIB distribution. Combining both options (plus the optional add-ons) determine the final list of IP cores to be delivered for a particular license.

The first step to set up a GRLIB license is choosing the type of distribution. The four types of GRLIB distributions are described below. For a full list of IP cores included in each distribution, please refer to 1.6.

- The **GPL** distribution is the open-source version of the IP library, and is subject to the terms and conditions of the GNU General Public License. The distribution is regularly updated and made available on Frontgrade Gaisler's website. This release includes designs for LEON3, LEON5 and NOEL-V, plus a number of peripherals and auxiliary cores. Note that fault-tolerant options are not supported in this tier.
- The **COM** distribution is the commercial version of the IP library and may target both FPGAs or ASICs. Note that fault-tolerant options are not supported in this tier.
- The **FT-FPGA** distribution is the fault-tolerant version of the IP library targeting FPGAs. It includes all features present in the COM release plus fault-tolerant functionality (like ECC protection for memories) and support for additional FPGA families.
- The **FT** distribution is the fault-tolerant version of the IP library targeting ASICs. It includes all features present in the FT-FPGA release plus additional cores and functionality for ASIC development, as well as support for ASIC technologies. An ASIC license also allows for prototyping on FPGAs without additional costs.

After selecting the type of distribution, the second step is to choose the processor and its configuration. GRLIB includes both SPARC (**LEON3**, **LEON4** and **LEON5**) and RISC-V (**NOEL-V**) processors. All commercial and fault-tolerant releases can be built around any of these processors. The processor configuration (controller, entry, general and advanced) determines some of the IPs included in the bundle, like the type of FPU and L2 cache. Further details are provided in the section 1.5. Please note that it is also possible to order a GRLIB release without any processor.

The final step is to include add-ons, i.e. additional IPs that by default are not automatically included in a particular distribution or a given processor configuration. The tables in Section 1.6 include some IPs that are never present in any distribution - instead, they shall be explicitly added to the GRLIB license as an add-on package. This is indicated by a remark in those tables.

It is also possible to create custom bundles of individual IP cores without ordering a standard GRLIB distribution. In that case, it is still important to specify whether the release shall contain fault-tolerant features or not.

A few examples are listed below to show some possible GRLIB distributions:

- Standard COM release with LEON3 in controller configuration without any add-on.
- Standard COM release with NOEL-V in general configuration with NANDFCTRL2 as an add-on.
- Standard FT-FPGA release with LEON4 in entry configuration with GRSPWROUTER and GRHSSL as add-ons.
- Standard FT release with LEON5 in advanced configuration with GRSPW2 and GRCANFD as add-ons.
- Custom delivery with only NANDFCTRL2, GRSPWROUTER, GRSPW2 and GRHSSL with fault-tolerance features.

1.5 Processor license overview

The table below provides a general overview of the different processor licenses. A license covers a specific processor from either the LEON SPARC family (LEON3, LEON4, or LEON5) or the NOEL

RISC-V family (NOEL-V) and determines which processor configurations can be instantiated in a design by the licensee.

Table 1. Processor licenses

License	Architecture	Processor	FPU Included	L2C	L2C-Lite	Striped bus
Controller	SPARC	LEON3 single issue	No	No	No	No
Entry	SPARC	LEON4 single issue	No	No	Yes	No
	RISC-V	NOEL-V single issue	NanoFPUnv			
General	SPARC	LEON5 dual issue	NanoFPU	Yes	Yes	No
	RISC-V	NOEL-V dual issue	NanoFPUnv			
Advanced	SPARC	LEON5 dual issue	GRFPU5 +NanoFPU	Yes	Yes	Yes
	RISC-V	NOEL-V dual issue	GRFPUnv +NanoFPUnv			

As illustrated in Table 1, the processor license determines which peripherals are included in terms of floating-point unit, Level-2 cache core and type of on-chip interconnect. The Advanced license includes multi-bus support for Level-2 caches and IOMMU, allowing for higher performance of the on-chip interconnect. If a peripheral is not included in a particular license, it is still available as an add-on package.

The LEON3 and LEON4 processors are only available in Controller and Entry license types, respectively.

For LEON5 and NOEL-V:

- The Advanced license also allows to instantiate the processor in configurations covered by the General and Entry licenses.
- The General license also allows to instantiate the processor in configurations covered by the Entry license.

Tables 2 and 3 provide detailed information about the specific IP cores covered by the different configurations for LEON and NOEL processors, respectively. In addition to the cores listed below, a GRLIB IP library license includes additional cores depending on the type of distribution (GPL, COM or FT), as explained in 1.4. This is further described in section 1.6.

Note: A LEON license includes either LEON4/LEON4FT or LEON5/LEON5FT, not both.

Note: FT licenses includes the fault-tolerant version of the respective processor core.

Table 2. LEON license cores

Name	Function	Controller	Entry	General	Advanced
LEON3/LEON3FT	SPARC V8 32-bit processor	Yes	No	No	No
DSU3	Multi-processor Debug support unit (LEON3)	Yes	No	No	No
L3STAT	LEON3 statistics unit	Yes	No	No	No
LEON4/LEON4FT	SPARC V8 32-bit processor	No	Yes	No	No
L4STAT	LEON4 statistics unit	No	Yes	No	No
DSU4	Multi-processor Debug support unit (LEON4)	No	Yes	No	No
LEON3 CLK2x	LEON processor double clocking (includes special LEON entity, interrupt controller and qualifier unit)	Yes	No	No	No

Table 2. LEON license cores

Name	Function	Controller	Entry	General	Advanced
LEON4 CLK2x	LEON processor double clocking (includes special LEON entity, interrupt controller and qualifier unit)	No	Yes	No	No
LEON5/LEON5FT	High-performance SPARC V8 32-bit Processor	No	Yes	Yes	Yes
DSU5	Multi-processor Debug support unit (LEON5)	No	Yes	Yes	Yes
LEON5SYS	High-performance SPARC V8 32-bit Processor Subsystem	No	Yes	Yes	Yes
GRFPU / GRFPC	High-performance IEEE-754 Floating-point unit with floating-point controller to interface LEON3/4	No	No	No	No
GRFPU-Lite / GRFPC-lite	Low-area IEEE-754 Floating-point unit with floating point controller to interface LEON3/4	No	No	No	No
GRFPU5/GRFPC5	High-performance IEEE-754 Floating-point unit with floating-point controller to interface LEON5	No	No	No	Yes
NANOFPU	Low-area combined Floating-point unit and floating point controller for LEON5	No	Yes	Yes	Yes
L2CACHE	Level-2 cache controller	No	No	Yes	Yes
L2C-LITE	Level-2 cache controller - Lite	No	Yes	Yes	Yes
GRIOMMU	I/O Memory management unit	No	Yes	Yes	Yes
GRIOMMU2	RISC-V I/O memory management unit	No	No	No	No

Table 3. NOEL license cores

Name	Function	Controller	Entry	General	Advanced
NOEL-V/NOEL-VFT	High-performance RISC-V RV64GC Processor	No	Yes	Yes	Yes
RVDM	RISC-V debug module	No	Yes	Yes	Yes
NOELVSYS	High-performance RISC-V RV64GC Processor Subsystem	No	Yes	Yes	Yes
GRFPUNV	High-performance RISC-V Floating-point unit for NOEL-V	No	No	No	Yes
NANOFPUNV	Low-area combined Floating-point unit and floating point controller for NOEL-V	No	Yes	Yes	Yes
L2CACHE	Level-2 cache controller	No	No	Yes	Yes
L2C-LITE	Level-2 cache controller - Lite	No	Yes	Yes	Yes
GRIOMMU	I/O Memory management unit	No	No	No	No
GRIOMMU2	RISC-V I/O memory management unit	No	No	Yes	Yes

1.6 IP core overview

The tables below lists all the IP cores available in GRLIB and their AMBA plug&play device identifier. The columns on the right indicate in which GRLIB distributions a core is available. Note that there are cores that are only included for certain processor configurations. These are listed in section 1.5. Spacecraft data handling IP cores are available under a separate license and can be found in section 1.7.

The distribution columns correspond to the main types of distributions for the GRLIB IP library: GPL, COM, and FT / FT-FPGA. Each commercial distribution (COM or FT) is based on a processor licensing model (see section 1.5). The lists below specify in which distribution each core is included with “Yes” or “No”. Cores marked with “Yes” are included in the corresponding distribution, whereas cores marked with “No” are excluded, regardless of the chosen processor configuration. Cores that are affected by the processor licensing model are marked with a note.

Some cores can only be licensed separately or as additions to existing releases, this is marked in the *Notes* column. Contact Frontgrade Gaisler for licensing details.

The open-source version of GRLIB includes only cores marked with “Yes” in the GPL column.

Fault tolerant features are only supported in FT or FT-FPGA distributions. This includes protection of Level-1 cache and register files for the LEON3, LEON4, LEON5 and NOEL-V processors and fault-tolerance features for other IP cores such as the PCI, Ethernet and SpaceWire controllers.

Some cores are delivered as encrypted RTL. Supported tools are listed in the GRLIB IP Library user’s manual. Contact Frontgrade Gaisler if an EDA tool is not in the list to make sure it is still supported.

Table 4 lists the processors and supporting cores that are affected by the processor license type and indicates whether they are included in the GPL release of GRLIB or not. For more details on which cores are included in each processor license, refer to section 1.5.

Table 4. Processor license functions

Name	Function	Vendor:Device	GPL	Note
LEON3	SPARC V8 32-bit processor	0x01 : 0x003	Yes	3)
LEON3FT	Fault-tolerant SPARC V8 32-bit Processor	0x01 : 0x053	No	2) 3)
DSU3	Multi-processor Debug support unit (LEON3)	0x01 : 0x004	Yes	
L3STAT	LEON3 statistics unit	0x01 : 0x098	Yes	
LEON4	SPARC V8 32-bit processor	0x01 : 0x048	No	3)
LEON4FT	Fault-tolerant SPARC V8 32-bit Processor	0x01 : 0x048	No	2) 3)
L4STAT	LEON4 statistics unit	0x01 : 0x047	No	
DSU4	Multi-processor Debug support unit (LEON4)	0x01 : 0x049	No	
LEON3/4 CLK2x	LEON processor double clocking (includes special LEON entity, interrupt controller and qualifier unit)	-	No	
LEON5	High-performance SPARC V8 32-bit Processor	0x01 : 0x0BA	Yes	3)
LEON5FT	Fault-tolerant high-performance SPARC V8 32-bit Processor	0x01 : 0x0BA	No	2) 3)
DSU5	Multi-processor Debug support unit (LEON5)	0x01 : 0x0BB	Yes	
LEON5SYS	High-performance SPARC V8 32-bit Processor Subsystem	-	Yes	
NOEL-V	High-performance RISC-V RV64GC Processor	0x01 : 0x0BD	Yes	3)
NOEL-VFT	Fault-tolerant high-performance RISC-V RV64GC Processor	0x01 : 0x0BD	No	2) 3)
RVDM	RISC-V debug module	0x01 : 0x0BE	Yes	
NOELVSYS	High-performance RISC-V RV64GC Processor Subsystem	-	Yes	
GRFPU / GRFPC	High-performance IEEE-754 Floating-point unit with floating-point controller to interface LEON3/4	-	No	1) 2)
GRFPU-Lite / GRFPC-lite	Low-area IEEE-754 Floating-point unit with floating point controller to interface LEON3/4	-	No	1) 2)
GRFPU5/GRFPC5	High-performance IEEE-754 Floating-point unit with floating-point controller to interface LEON5	-	No	1) 2)
NANOFPU	Low-area combined Floating-point unit and floating point controller for LEON5	-	Yes	1)
GRFPUNV	High-performance RISC-V Floating-point unit for NOEL-V	-	No	1) 2)

Table 4. Processor license functions

Name	Function	Vendor:Device	GPL	Note
NANOFPUNV	Low-area combined Floating-point unit and floating point controller for NOEL-V	-	Yes	1)
L2CACHE	Level-2 cache controller	0x01 : 0x04B	No	1) 2)
L2C-LITE	Level-2 cache controller - Lite	0x01 : 0x0D0	Yes	1)
GRIOMMU	I/O Memory management unit	0x01 : 0x04F	No	1)
GRIOMMU2	RISC-V I/O memory management unit	0x01 : 0x0D3	No	1)

1) Available as separate package or as addition to existing releases.
2) Delivered as encrypted RTL or in netlist format.
3) The non-FT and FT processor cores are functionally equivalent except that fault-tolerance features can be enabled for the FT versions.

Table 5. Processor support functions

Name	Function	Vendor:Device	GPL	COM	FT	Notes
CLKGEN	Clock generation	-	Yes	Yes	Yes	
DIV32	Divider module	-	Yes	Yes	Yes	
GPTIMER	General purpose timer unit	0x01 : 0x011	Yes	Yes	Yes	
GRPPSTIMER	Timer with PPS synchronization functions	0x01 : 0x0DD	No	No	No	1)
GRCLKGATE	Clock gate unit	0x01 : 0x02C	No	Yes	Yes	
GRDMAC	DMA controller with AHB/APB bridge (deprecated)	0x01 : 0x095	No	No	No	
GRDMAC2	DMA controller	0x01 : 0x0C0	Yes	Yes	Yes	2)
GRTIMER	General purpose timer unit (deprecated)	0x01 : 0x038	No	Yes	Yes	
GRWATCHDOG	Watchdog module with separate clock domain	0x01 : 0x0C9	No	Yes	Yes	
IRQMP	Multi-processor Interrupt controller	0x01 : 0x00D	Yes	Yes	Yes	
IRQ(A)MP	Multi-processor Interrupt controller	0x01 : 0x00D	Yes	Yes	Yes	
MUL32	32x32 multiplier module	-	Yes	Yes	Yes	
MULTLIB	High-performance multipliers	-	Yes	Yes	Yes	

1) Available as separate package or as addition to existing releases.
2) Hardware accelerators are not included in the GPL version of the core.

Table 6. Memory controllers and supporting cores

Name	Function	Vendor:Device	GPL	COM	FT	Note
DDRSPA	Single-port 16/32/64 bit DDR controller	0x01 : 0x025	Yes	Yes	Yes	3)
DDR2SPA	Single-port 16/32/64-bit DDR2 controller	0x01 : 0x02E	Yes	Yes	Yes	3)
MCTRL	8/16/32-bit PROM/SRAM/SDRAM controller	0x04 : 0x00F	Yes	Yes	Yes	
SDCTRL	32-bit PC133 SDRAM controller	0x01 : 0x009	Yes	Yes	Yes	
SRCTRL	8/32-bit PROM/SRAM controller	0x01 : 0x008	Yes	Yes	Yes	
SSRCTRL	32-bit Synchronous SRAM (SSRAM) controller	0x01 : 0x00A	No	Yes	Yes	
FTADDR	Autonomous DDR2/DDR3 Controller with EDAC	0x01 : 0x0AE	No	No	No	1) 2)
FTMCTRL	8//32-bit PROM/SRAM/SDRAM Controller w. RS/BCH EDAC	0x01 : 0x054	No	No	Yes	
FTSDCTRL	32/64-bit PC133 SDRAM Controller with EDAC	0x01 : 0x055	No	No	Yes	

Table 6. Memory controllers and supporting cores

Name	Function	Vendor:Device	GPL	COM	FT	Note
FTSDCTRL64	64-bit PC133 SDRAM Controller with EDAC (deprecated)	0x01 : 0x058	No	No	No	4)
FTSRCTRL	8/32-bit PROM/SRAM/IO Controller w. BCH EDAC	0x01 : 0x051	No	No	Yes	
FTSRCTRL8	8-bit SRAM / 16-bit IO Memory Controller with EDAC	0x01 : 0x056	No	No	Yes	
NANDFCTRL	NAND Flash Memory Controller	0x01 : 0x059	No	Yes	Yes	
NANDFCTRL2	NAND Flash Memory Controller with DMA	0x01 : 0x0C5	No	No	No	1) 2)
SPIMCTRL	SPI Memory controller	0x01 : 0x045	Yes	Yes	Yes	
AHBSTAT	AHB status register	0x01 : 0x052	Yes	Yes	Yes	
MEMSCRUB	Memory scrubber	0x01 : 0x057	No	No	Yes	

1) Available as separate package or as addition to existing releases.
2) Delivered as encrypted RTL or in netlist format.
3) Requires PHY for selected target technology. Please see IP core documentation for supported technologies.
4) Deprecated

Table 7. AMBA Bus control

Name	Function	Vendor:Device	GPL	COM	FT	Note
AHB2AHB	Uni-directional AHB/AHB Bridge	0x01 : 0x020	No	Yes	Yes	
AHB2AVLA	Asynchronous AHB to Avalon Bridge	0x01 : 0x096	Yes	Yes	Yes	
AHB2AXIB	AHB to AXI bridge	0x01 : 0x09F	Yes	Yes	Yes	
AHBBRIDGE	Bi-directional AHB/AHB Bridge	0x01 : 0x020	No	Yes	Yes	
AHBCTRL	AMBA AHB bus controller with plug&play	-	Yes	Yes	Yes	
APBCTRL	AMBA APB Bridge with plug&play	0x01 : 0x006	Yes	Yes	Yes	
AHBLITM2AHBM	AHB-LITE to AHB MASTER	-	Yes	Yes	Yes	
AHBM2AXI	AHB Master to AXI Adapter	-	Yes	Yes	Yes	
AHBTRACE	AMBA AHB Trace buffer	0x01 : 0x017	Yes	Yes	Yes	
AXI2AHB	AXI4 to AHB2 Bridge	0x01 : 0x0A0	Yes	Yes	Yes	
AXI4_RESIZE	AXI4 Transaction Resize	-	Yes	Yes	Yes	
AXIS_BUFFER	AXI4-Stream Buffer	-	Yes	Yes	Yes	
AXIS_GEARBOX	AXI4-Stream Gearbox	-	Yes	Yes	Yes	
MMA	Memory Mapped AMBA bridge	0x01 : 0x07F	No	Yes	Yes	

Table 8. PCI interface

Name	Function	Vendor:Device	GPL	COM	FT	Note
GRPCI2	Advanced 32-bit PCI bridge	0x01 : 0x07C	Yes	Yes	Yes	
PCITARGET	32-bit target-only PCI interface (deprecated)	0x01 : 0x012	No	No	No	
PCIMTF/GRPCI	32-bit PCI master/target interface with FIFO (deprecated)	0x01 : 0x014	No	No	No	
PCITRACE	32-bit PCI trace buffer (deprecated)	0x01 : 0x015	No	No	No	
PCIDMA	DMA controller for PCIMTF (deprecated)	0x01 : 0x016	No	No	No	
PCIARB	PCI Bus arbiter	0x04 : 0x010	Yes	Yes	Yes	

Table 9. On-chip memory functions

Name	Function	Vendor:Device	GPL	COM	FT	Note
AHBRAM	Single-port RAM with AHB interface	0x01 : 0x00E	Yes	Yes	Yes	
AHBDPRAM	Dual-port RAM with AHB and user back-end interface	0x01 : 0x00F	Yes	Yes	Yes	
AHBROM	ROM generator with AHB interface	0x01 : 0x01B	Yes	Yes	Yes	
FTAHBRAM	RAM with AHB interface and EDAC protection	0x01 : 0x050	No	No	Yes	
REGFILE_3P	Parametrizable 3-port register file	-	Yes	Yes	Yes	
SYNCRAM	Parametrizable 1-port RAM	-	Yes	Yes	Yes	
SYNCRAM_2P	Parametrizable 2-port RAM	-	Yes	Yes	Yes	
SYNCRAM_DP	Parametrizable dual-port RAM	-	Yes	Yes	Yes	

Table 10. Serial communication

Name	Function	Vendor:Device	GPL	COM	FT	Note
AHBUART	Serial/AHB debug interface	0x01 : 0x007	Yes	Yes	Yes	
AHBJTAG	JTAG/AHB debug interface	0x01 : 0x01C	Yes	Yes	Yes	
APBPS2	PS/2 host controller with APB interface	0x01 : 0x060	Yes	Yes	Yes	
APBUART	Programmable UART with APB interface	0x01 : 0x00C	Yes	Yes	Yes	
CAN_OC	Opencores CAN 2.0 MAC with AHB interface	0x01 : 0x019	Yes	Yes	Yes	
GRCAN	CAN 2.0 Controller with DMA	0x01 : 0x03D	No	Yes	Yes	
GRCANFD	CAN Flexible Data Rate Controller	0x01 : 0x0B5	No	No	No	1)
GRHSSL	SpaceFibre and WizardLink controller with DMA engine	0x01 : 0x0C8	No	No	No	1) 2) 5)
GRSPFI_CODEC	SpaceFibre Codec	N/A	No	No	No	1) 2)
GRSPFI	SpaceFibre Codec with DMA Engine and Bus Master Interface	0x01 : 0x0BC	No	No	No	1) 2) 7)
GRSPW2	SpaceWire link with RMAP and AHB interface	0x01 : 0x029	No	No	No	1) 2) 4)
GRSPW_CODEC	SpaceWire Codec	N/A	No	No	No	1) 2) 4)
GRSPW_PHY	Receiver Physical layer for GRSPW (deprecated)	N/A	No	No	No	
GRSPW2_PHY	Receiver Physical layer	N/A	No	No	No	
GRSPWROUTER	SpaceWire routing switch	0x01 : 0x03E	No	No	No	1) 2) 4)
GRSPWTDP	SpaceWire - Time Distribution Protocol	0x01 : 0x097	No	No	No	1)
GRSRIO	Serial Rapid IO (deprecated)	0x01 : 0x0A8	No	No	No	
GRWIZL	WizardLink codec with DMA engine and Bus Master Interface	0x01 : 0x0C7	No	No	No	1) 2)
I2C2AHB	I2C (slave) to AHB bridge	0x01 : 0x00B	Yes	Yes	Yes	
I2CMST	I2C Master with APB interface	0x01 : 0x028	Yes	Yes	Yes	
I2CSLV	I2C Slave with APB interface	0x01 : 0x03E	Yes	Yes	Yes	
SOCBRIDGE	AHB to AHB interface through byte stream	0x01 : 0x0C4	No	Yes	Yes	
SPI2AHB	SPI (slave) to AHB bridge	0x01 : 0x05C	Yes	Yes	Yes	

Table 10. Serial communication

Name	Function	Vendor:Device	GPL	COM	FT	Note
SPICTRL	SPI Controller with APB interface	0x01 : 0x02D	Yes	Yes	Yes	
SPIMASTER	SPI master device	0x01 : 0x0A6	No	No	No	1) 6)
SPISLAVE	Dual port SPI slave with space protocol	0x01 : 0x0A7	No	No	No	1) 6)
TAP	JTAG TAP controller	-	No	Yes	Yes	

1) Available as separate package or as addition to existing releases.
2) Delivered as encrypted RTL or in netlist format.
3) Deprecated
4) Delivered together with GRSPW2_PHY.
5) GRHSSL includes GRSPFI, GRSPFI_CODEC and GRWIZL under the same license.
6) SPIMASTER and SPISLAVE are delivered together under the same license.
7) GRSPFI includes GRSPFI_CODEC under the same license.

Table 11. Ethernet interface

Name	Function	Vendor:Device	GPL	COM	FT	Note
GRETH	Frontgrade Gaisler 10/100 Mbit Ethernet MAC with AHB I/F	0x01 : 0x01D	Yes	Yes	Yes	
GRETH_GBIT	Frontgrade Gaisler 10/100/1000 Mbit Ethernet MAC with AHB	0x01 : 0x01D	No	Yes	Yes	
MDIO_CTRL	MDIO Serial Bus Controller	0x01 : 0x0DF	No	Yes	Yes	
RGMII	Frontgrade Gaisler RGMII<-> GMII adapter	0x01 : 0x093	Yes	Yes	Yes	

Table 12. USB interface

Name	Function	Vendor:Device	GPL	COM	FT	Note
GRUSBHC	USB-2.0 Host controller (UHCI/EHCI) with AHB I/F	0x01 : 0x027	No	No	No	1)
GRUSBDC / GRUSB_DCL	USB-2.0 device controller / AHB debug communication link	0x01 : 0x022	No	No	No	1)

1) Available as separate package or as addition to existing releases.

Table 13. MIL-STD-1553 Bus interface

Name	Function	Vendor:Device	GPL	COM	FT	Note
GR1553B	Advanced MIL-ST-1553B / AS15551 Interface	0x01 : 0x04D	No	No	No	1) 2) 3)
GRRT	MIL-STD-1553B / AS15531 Remote Terminal Back-End	N/A	No	No	No	1) 2) 3)

1) Available as separate package or as addition to existing releases.
2) Delivered as encrypted RTL or in netlist format.
3) Both GR1553B and GRRT are covered by the same IP core license and are delivered in the same package.

Table 14. Encryption and compression

Name	Function	Vendor:Device	GPL	COM	FT	Note
GRAES	128-bit AES Encryption/Decryption Core	0x01 : 0x073	No	No	No	1) 2)
GRAES_DMA	Advanced Encryption Standard with DMA	0x01 : 0x07B	No	No	No	1) 2)
GRECC	Elliptic Curve Cryptography Core	0x01 : 0x074	No	No	No	1)
GRSHYLOC	Wrapper for the SHYLOC compressor with DMA engine	0x01 : 0x0B7	No	No	No	1) 3)
SHYLOC	CCSDS121/123 Lossless Data Compression core	N/A	No	No	No	1) 3)

1) Available as separate package or as addition to existing releases.
2) GRAES and GRAES_DMA are delivered together under the same license.
3) GRSHYLOC and SHYLOC are delivered together under the same license.

Table 15. FPGA control and error mitigation

Name	Function	Vendor:Device	GPL	COM	FT	Note
GRSCRUB	AMD external FPGA scrubber controller	0x01 : 0x0C1	No	No	No	1)
GRLSEDC	Lattice FPGA soft error detection interface	0x01 : 0x0DE	No	Yes	Yes	

1) Available as separate package or as addition to existing releases.

Table 16. Simulation and debugging

Name	Function	Vendor:Device	GPL	COM	FT	Note
SRAM	SRAM simulation model with srecord preload	-	Yes	Yes	Yes	
MT48LC16M16	Micron SDRAM model with srecord preload	-	Yes	Yes	Yes	
MT46V16M16	Micron DDR model	-	Yes	Yes	Yes	
CY7C1354B	Cypress ZBT SSRAM model with srecord pre-load	-	Yes	Yes	Yes	
AHBMSTEM	AHB master simulation model with scripting (deprecated)	0x01 : 0x040	Yes	Yes	Yes	
AHBSLVEM	AHB slave simulation model with scripting (deprecated)	0x01 : 0x041	Yes	Yes	Yes	
AMBAMON	AHB and APB protocol monitor	-	No	Yes	Yes	
ATF	AMBA test framework consisting of master, slave and arbiter.	0x01 : 0x068 - 0x06A	No	Yes	Yes	
LOGAN	On-chip Logic Analyzer	0x01 : 0x062	Yes	Yes	Yes	

Table 17. Graphics functions

Name	Function	Vendor:Device	GPL	COM	FT	Note
APBVGA	VGA controller with APB interface	0x01 : 0x061	Yes	Yes	Yes	
SVGACTRL	VGA controller core with DMA	0x01 : 0x063	Yes	Yes	Yes	

Table 18. Auxiliary functions

Name	Function	Vendor:Device	GPL	COM	FT	Note
GRACECTRL	AMBA SystemACE interface controller	0x01 : 0x067	Yes	Yes	Yes	
GRADCDAC	Combined ADC / DAC Interface	0x01 : 0x036	No	Yes	Yes	
GRFIFO	External FIFO Interface with DMA	0x01 : 0x035	No	Yes	Yes	
GRGPIO	General purpose I/O port	0x01 : 0x01A	Yes	Yes	Yes	
GRGPREG	General purpose register	0x01 : 0x087	Yes	Yes	Yes	
GRGPRBANK	General purpose register bank	0x01 : 0x08F	Yes	Yes	Yes	
GRPULSE	General purpose I/O with pulses	0x01 : 0x037	No	Yes	Yes	
GRPWM	PWM generator	0x01 : 0x04A	No	Yes	Yes	
GRSYSMON	AMBA Wrapper for Xilinx System Monitor	0x01 : 0x066	Yes	Yes	Yes	
GRVERSION	Version and revision register	0x01 : 0x03A	Yes	Yes	Yes	

Table 19. Error detection and correction functions

Name	Function	GPL	COM	FT	Note
BCH_DECODER	Highly configurable systematic BCH decoder	No	No	No	1)
BCH_ENCODER	Highly configurable systematic BCH encoder	No	No	No	1)
RS(24, 16, 8, E=1)	16 bit data, 8 check bits, corrects 4-bit error in 1 nibble	No	No	Yes	
RS(40, 32, 8, E=1)	32 bit data, 8 check bits, corrects 4-bit error in 1 nibble	No	No	Yes	
RS(48, 32, 16, E=1+1)	32 bit data, 16 check bits, corrects 4-bit error in 2 nibbles	No	No	Yes	
RS(48, 32, 16, E=2)	32 bit data, 16 check bits, corrects 4-bit error in 2 nibbles	No	No	Yes	
GR(2^4)(68, 60, 8, T=1)	QEC/QED error correction code encoder/decoder	No	No	Yes	

1) Available as separate package or as addition to existing releases. Included with NANDFCTRL2.

Table 20. Test functions

Name	Function	GPL	COM	FT	Note
SYNCIOTEST	Test block for synchronous I/O interfaces	Yes	Yes	Yes	

Frontgrade Gaisler AB
Kungsgatan | SE-411 19 | Goteborg | Sweden
+46 31 7758650 | frontgrade.com/gaisler

1.7 Spacecraft data handling IP cores

The Spacecraft Data Handling IP cores represent a collection of cores that have been developed specifically for the space sector.

These IP cores implement functions commonly used in spacecraft data handling and management systems. They implement international standards from organizations such as Consultative Committee for Space Data Systems (CCSDS), European Cooperation on Space Standardization (ECSS), and the former Procedures, Standards and Specifications (PSS) from the European Space Agency (ESA).

The table below lists the existing CCSDS/ECSS IP cores and AMBA plug&play device identifiers.

The Spacecraft Data Handling IP cores can not be licensed separately.

Table 21. Spacecraft data handling functions

Name	Function	Vendor : Device	GPL	COM	FT	TMTC	Note
GRTM	CCSDS Telemetry Encoder	0x01 : 0x030	No	No	No	Yes	
GRTM_DESC	CCSDS Telemetry Encoder - Descriptor	0x01 : 0x084	No	No	No	Yes	
GRTM_VC	CCSDS Telemetry Encoder - Virtual Channel Generation	0x01 : 0x085	No	No	No	Yes	
GRTM_PAHB	CCSDS Telemetry Encoder - VC Generation Input - AMBA	0x01 : 0x088	No	No	No	Yes	
GRTM_PW	CCSDS Telemetry Encoder - VC Generation Input - PacketWire	N/A	No	No	No	Yes	
GRTM_UART	CCSDS Telemetry Encoder - VC Generation Input - UART	N/A	No	No	No	Yes	
GRTM_CLCWRX	CCSDS Telemetry Encoder - CLCW Receiver	N/A	No	No	No	Yes	2)
GRTM_CLCWMUX	CCSDS Telemetry Encoder - CLCW Multiplexer	N/A	No	No	No	Yes	2)
GRGEFFE	CCSDS Telemetry Encoder - Geffe Generator	0x01 : 0x086	No	No	No	Yes	
GRCE/GRCD	CCSDS Convolutional Encoder and Quicklook Decoder	N/A	No	No	No	Yes	
GRTMRX	CCSDS Telemetry Receiver	0x01 : 0x082	No	No	No	Yes	
GRTC	CCSDS Telecommand Decoder - Coding Layer	0x01 : 0x031	No	No	No	Yes	
TCAU	ESA PSS Telecommand Decoder Authentication Unit	N/A	No	No	No	Yes	
GRTC-COP	CCSDS-ECSS Telecommand Decoder	0x01 : 0x0A5	No	No	No	Yes	
GRTC_HW	CCSDS Telecommand Decoder - Hardware Commands	N/A	No	No	No	Yes	
GRTC_UART	CCSDS Telecommand Decoder - UART	N/A	No	No	No	Yes	
GRTC_CLCWTX	CCSDS Telecommand Decoder - CLCW Transmitter	N/A	No	No	No	Yes	2)
GRTCTX	CCSDS Telecommand Transmitter	0x01 : 0x083	No	No	No	Yes	
GRCTM	CCSDS Time manager	0x01 : 0x033	No	No	No	Yes	
SPWCUC	SpaceWire - CCSDS Unsegmented Code Transfer Protocol	0x01 : 0x089	No	No	No	Yes	
GRPW	PacketWire receiver with AHB interface	0x01 : 0x032	No	No	No	Yes	
GRPWRX	PacketWire Receiver (rev 1)	0x01 : 0x08E	No	No	No	Yes	
GRPWTX	PacketWire Transmitter (rev 1)	0x01 : 0x08D	No	No	No	Yes	

Table 21. Spacecraft data handling functions

Name	Function	Vendor : Device	GPL	COM	FT	TMTC	Note
APB2PW	PacketWire Transmitter Interface (rev 0)	0x01 : 0x03B	No	No	No	Yes	
PW2APB	PacketWire Receiver Interface (rev 0)	0x01 : 0x03C	No	No	No	Yes	
AHB2PP	Packet Parallel Interface	0x01 : 0x039	No	No	No	Yes	
GRRM	Reconfiguration Module	0x01 : 0x09A	No	No	No	No	1)

1) Available as separate package or as addition to existing releases.
2) There is no user manual for these simple cores.

1.8 Supported technologies

Technology support and instructions for extending GRLIB with support for additional technologies is documented in the 'GRLIB User's Manual'. The table below shows the technology maps available from Frontgrade Gaisler for GRLIB and in which GRLIB distributions these technology maps are included.

Vendor	Technology	GPL	COM	FT	FT-FPGA	Comment
Actel / Microsemi	ProASIC3, ProASIC3e, ProASIC3l, Axcelerator, Axcelerator DSP, Fusion, IGLOO2, PolarFire	No	Yes	Yes	Yes	
Actel / Microsemi	RTG4	No	No	Yes	Yes	
Altera	Cyclone2 - 4, Stratix - StratixV	Yes	Yes	Yes	Yes	Note that several parts of the FT and FT-FPGA versions are distributed as encrypted RTL. Encrypted RTL is not provided for the Quartus II tool.
Lattice	Certus-NX, CertusPro-NX, CrossLink-NX	Yes	Yes	Yes	Yes	
Lattice	-	Yes	Yes	Yes	Yes	
Xilinx	Unisim (Virtex2 - UltraScale+)	Yes	Yes	Yes	Yes	
Xilinx	Hi-rel support (Virtex5-QV and UltraScale)	No	No	Yes	Yes	Support for built-in ECC protection in addition to Xilinx Sirf and TMR-Tool support
NanoXplore	BRAVE (NG-Medium, NG-Large, NG-Ultra)	No	No	Yes	Yes	
Other ASIC	-	No	-	-	No	Contact Frontgrade Gaisler for details. See also *GRLIB IP Library User's Manual*.

1.9 Implementation characteristics

Implementation characteristics are available in the GRLIB area spreadsheet:

http://www.gaisler.com/products/grlib/grlib_area.xls

The spreadsheet is also included in GRLIB packages together with this document.

1.10 Definitions

This section and the following subsections define the typographic and naming conventions used throughout this document.

1.10.1 Bit numbering

The following conventions are used for bit numbering:

- The most significant bit (MSb) of a data type has the leftmost position
- The least significant bit of a data type has the rightmost position
- Unless otherwise indicated, the MSb of a data type has the highest bit number and the LSb the lowest bit number

1.10.2 Radix

The following conventions is used for writing numbers:

- Binary numbers are indicated by the prefix "0b", e.g. 0b1010.
- Hexadecimal numbers are indicated by the prefix "0x", e.g. 0xF00F
- Unless a radix is explicitly declared, the number should be considered a decimal.

1.10.3 Data types

Byte (BYTE)	8 bits of data
Halfword (HWORD)	16 bits of data
Word (WORD)	32 bits of data
Double word (DWORD)	64 bits of data
Quad word (4WORD)	128-bits of data

1.11 Register descriptions

An example register, showing the register layout used throughout this document, can be seen in table 22. The values used for the reset value fields are described in table 23, and the values used for the field type fields are described in table 24. Fields that are named RESERVED, RES, or R are read-only fields. These fields can be written with zero or with the value read from the same register field.

Table 22. <Address> - <Register acronym> - <Register name>

31 24	23 16	15 8	7 0
EF3	EF2	EF1	EF0
<Reset value for EF3>	<Reset value for EF2>	<Reset value for EF1>	<Reset value for EF0>
<Field type for EF3>	<Field type for EF2>	<Field type for EF1>	<Field type for EF0>

31: 24 Example field 3 (EF3) - <Field description>
23: 16 Example field 2 (EF2) - <Field description>
15: 8 Example field 1 (EF1) - <Field description>
7: 0 Example field 0 (EF0) - <Field description>

Table 23. Reset value definitions

Value	Description
0	Reset value 0.
1	Reset value 1. Used for single-bit fields.
0xNN	Hexadecimal representation of reset value. Used for multi-bit fields.
0bNN	Binary representation of reset value. Used for multi-bit fields.
NR	Field not reset. Fields marked with NR will be reset to 0 if full reset of all registers have been enabled in the global GRLIB configuration options (see GRLIB user manual for more information).
*	Special reset condition, described in textual description of the field. Used for example when reset value is taken from a pin.
-	Don’t care / Not applicable

Table 24. Field type definitions

Value	Description
r	Read-only. Writes have no effect.
w	Write-only. Used for a writable field in a register where the field’s read-value has no meaning.
rw	Readable and writable.
rw*	Readable and writable. Special condition for write, described in textual description of field.
wc	Write-clear. Readable, and cleared when written with a 1
cas	Readable, and writable through compare-and-swap. Only applies to SpaceWire Plug-and-Play registers.

Memory mapped registers may be mapped at several locations within an address space assigned to a peripheral (register set may wrap due to decoding a subset of the address bits). Because of this, memory accesses should only be performed to memory locations where registers are documented to be present.

2 AHB2AHB - Uni-directional AHB/AHB bridge

2.1 Overview

The uni-directional AHB/AHB bridge is used to connect two AMBA AHB buses clocked by synchronous clocks with any frequency ratio. The bridge is connected through a pair consisting of an AHB slave and an AHB master interface. AHB transfer forwarding is performed in one direction, where AHB transfers to the slave interface are forwarded to the master interface. Applications of the uni-directional bridge include system partitioning, clock domain partitioning and system expansion.

Features offered by the uni-directional AHB to AHB bridge are:

- Single and burst AHB transfers
- Data buffering in internal FIFOs
- Efficient bus utilization through (optional) use of SPLIT response and data prefetching. **NOTE:** SPLIT responses require an AHB arbiter that allows assertion of HSPLIT during second cycle of SPLIT response. This is supported by GRLIB's AHBCTRL IP core.
- Posted writes
- Read and write combining, improves bus utilization and allows connecting cores with differing AMBA access size restrictions.
- Deadlock detection logic enables use of two uni-directional bridges to build a bi-directional bridge (one example is the bi-directional AHB/AHB bridge core (AHBBRIDGE))

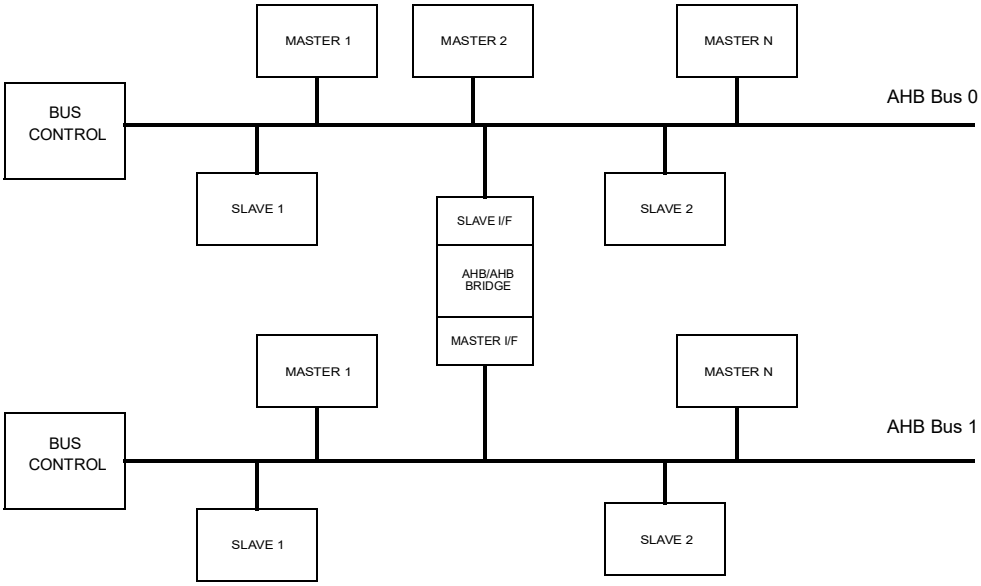


Figure 1. Two AHB buses connected with (uni-directional) AHB/AHB bridge

2.2 Operation

2.2.1 General

The address space occupied by the AHB/AHB bridge on the slave bus is configurable and determined by Bank Address Registers in the slave interface's AHB Plug&Play configuration record.

The bridge is capable of handling single and burst transfers of all burst types. Supported transfer sizes (HSIZE) are BYTE, HALF-WORD, WORD, DWORD, 4WORD and 8WORD.

For AHB write transfers write data is always buffered in an internal FIFO implementing posted writes. For AHB read transfers the bridge uses GRLIB's AMBA Plug&Play information to determine whether the read data will be prefetched and buffered in an internal FIFO. If the target address for an AHB read burst transfer is a prefetchable location the read data will be prefetched and buffered.

The bridge can be implemented to use SPLIT responses or to insert wait states when handling an access. With SPLIT responses enabled, an AHB master initiating a read transfer to the bridge is always splitted on the first transfer attempt to allow other masters to use the slave bus while the bridge performs read transfer on the master bus.The descriptions of operation in the sections below assume that the bridge has been implemented with support for AMBA SPLIT responses. The effects of disabling support for AMBA SPLIT responses are described in section 2.2.11.

If interrupt forwarding is enabled the interrupts on the slave bus interrupt lines will be forwarded to the master bus and vice versa.

2.2.2 AHB read transfers

When a read transfer is registered on the slave interface the bridge gives a SPLIT response. The master that initiated the transfer will be de-granted allowing other bus masters to use the slave bus while the bridge performs a read transfer on the master side. The master interface then requests the bus and starts the read transfer on the master side. Single transfers on the slave side are normally translated to single transfers with the same AHB address and control signals on the master side, however read combining can translate one access into several smaller accesses. Translation of burst transfers from the slave to the master side depends on the burst type, burst length, access size and the AHB/AHB bridge configuration.

If the read FIFO is enabled and the transfer is a burst transfer to a prefetchable location, the master interface will prefetch data in the internal read FIFO. If the splitted burst on the slave side was an incremental burst of unspecified length (INCR), the length of the burst is unknown. In this case the master interface performs an incremental burst up to a specified address boundary (determined by the VHDL generic *rburst).* The bridge can be configured to recognize an INCR read burst marked as instruction fetch (indicated on HPROT signal). In this case the prefetching on the master side is completed at the end of a cache line (the cache line size is configurable through the VHDL generic *iburst*). When the burst transfer is completed on the master side, the splitted master that initiated the transfer (on the slave side) is allowed in bus arbitration by asserting the appropriate HSPLIT signal to the AHB controller. The splitted master re-attempts the transfer and the bridge will return data with zero wait states.

If the read FIFO is disabled, or the burst is to non-prefetchable area, the burst transfer on the master side is performed using sequence of NONSEQ, BUSY and SEQ transfers. The first access in the burst on the master side is of NONSEQ type. Since the master interface can not decide whether the splitted burst will continue on the slave side or not, the master bus is held by performing BUSY transfers. On the slave side the splitted master that initiated the transfer is allowed in bus arbitration by asserting the HSPLIT signal to the AHB controller. The first access in the transfer is completed by returning read data. The next access in the transfer on the slave side is extended by asserting HREADY low. On the master side the next access is started by performing a SEQ transfer (and then holding the bus using BUSY transfers). This sequence is repeated until the transfer is ended on the slave side.

In case of an ERROR response on the master side the ERROR response will be given for the same access (address) on the slave side. SPLIT and RETRY responses on the master side are re-attempted until an OKAY or ERROR response is received.

2.2.3 AHB write transfers

The AHB/AHB bridge implements posted writes. During the AHB write transfer on the slave side the data is buffered in the internal write FIFO and the transfer is completed on the slave side by always giving an OKAY response. The master interface requests the bus and performs the write transfer when the master bus is granted. If the burst transfer crosses the write burst boundary (defined by VHDL

generic *wburst*), a SPLIT response is given. When the bridge has written the contents of the FIFO out on the master side, the bridge will allow the master on the slave side to perform the remaining accesses of the write burst transfer.

Writes are accepted with zero wait states if the bridge is idle and the incoming access is not locked. If the incoming access is locked, each access will have one wait state. If write combining is disabled a non-locked BUSY cycle will lead to a flush of the write FIFO. If write combining is enabled or if the incoming access is locked, the bridge will not flush the write FIFO during the BUSY cycle.

2.2.4 Deadlock conditions

When two bridges are used to form a bi-directional bridge, a deadlock situation can occur if the bridges are simultaneously accessed from both buses. The bridge that has been configured as a slave contains deadlock detection logic which will resolve a deadlock condition by giving a RETRY response, or by issuing SPLIT complete followed by a new SPLIT response. When the core resolves a deadlock while prefetching data, any data in the prefetch buffer will be dropped when the core's slave interface issues the AMBA RETRY response. When the access is retried it may lead to the same memory locations being read twice.

Deadlock detection logic for bi-directional configurations may lead to deadlocks in other parts of the system. Consider the case where a processor on bus A on one side of the bidirectional bridge needs to perform an instruction fetch over the bridge before it can release a semaphore located in memory on bus A. Another processor on bus B, on the other side of the bridge, may spin on the semaphore waiting for its release. In this scenario, the accesses from the processor on bus B could, depending on system configuration, continuously trigger a deadlock condition where the core will drop data in, or be prevented from initiating, the instruction fetch for the processor on bus A. Due to scenarios of this kind the bridge should not be used in bi-directional configurations where dependencies as the one described above exist between the buses connected by the bridge.

Other deadlock conditions exist with locked transfers, see section 2.2.5.

2.2.5 Locked transfers

The AHB/AHB bridge supports locked transfers. The master bus will be locked when the bus is granted and remain locked until the transfer completes on the slave side. Locked transfers can lead to deadlock conditions, the core's VHDL generic *lckdac* determines if and how the deadlock conditions are resolved.

With the VHDL generic *lckdac* set to 0, locked transfers may *not* be made after another read access which received SPLIT until the first read access has received split complete. This is because the bridge will return split complete for the first access first and wait for the first master to return. This will cause deadlock since the arbiter is not allowed to change master until a locked transfer has been completed. The AMBA specification requires that the locked transfer is handled before the previous transfer, which received a SPLIT response, is completed.

With *lckdac* set to 1, the core will respond with an AMBA ERROR response to locked access that is made while an ongoing read access has received a SPLIT response. With *lckdac* set to 2 the bridge will save state for the read access that received a SPLIT response, allow the locked access to complete, and then complete the first access. All non-locked accesses from other masters will receive SPLIT responses until the saved data has been read out.

If the core is used to create a bi-directional bridge there is one more deadlock condition that may arise when locked accesses are made simultaneously in both directions. If the VHDL generic *lckdac* is set to 0 the core will deadlock. If *lckdac* is set to a non-zero value the slave bridge will resolve the deadlock condition by issuing an AMBA ERROR response to the incoming locked access.

2.2.6 Read and write combining

Read and write combining allows the bridge to assemble or split AMBA accesses on the bridge's slave interface into one or several accesses on the master interface. This functionality can improve bus utilization and also allows cores that have differing AMBA access size restrictions to communicate with each other. The functionality attained by read and write combining depends on the VHDL generics *rdcomb* (defines type of read combining), *wrcomb* (defines type of write combining), *slvmstaccsz* (defines maximum AHB access size supported by the bridge's slave interface) and *mstmaccsz* (defines maximum AHB access size that can be used by bridge's master interface). These VHDL generics are described in section 2.5.4. The table below shows the effect of different settings. BYTE and HALF-WORD accesses are special cases. The table does not list illegal combinations, for instance *mstmaccsz* /= *slvmaccsz* requires that *wrcomb* /= 0 and *rdcomb* /= 0.

Table 25. Read and write combining

Access on slave interface	Access size	wrcomb	rdcomb	Resulting access(es) on master interface
BYTE or HALF-WORD single read access to any area	-	-	-	Single access of same size
BYTE or HALF-WORD read burst to prefetchable area	-	-	-	Incremental read burst of same access size as on slave interface, the length is the same as the number of 32-bit words in the read buffer, but will not cross the read burst boundary.
BYTE or HALF-WORD read burst to non-prefetchable area	-	-	-	Incremental read burst of same access size as on slave interface, the length is the same as the length of the incoming burst. The master interface will insert BUSY cycles between the sequential accesses.
BYTE or HALF-WORD single write	-	-	-	Single access of same size
BYTE or HALF-WORD write burst	-	-	-	Incremental write burst of same size and length, the maximum length is the number of 32-bit words in the write FIFO.
Single read access to any area	Access size <= mstmaccsz	-	-	Single access of same size
Single read access to any area	Access size > mstmaccsz	-	1	Sequence of single accesses of mstmaccsz. Number of accesses: (access size)/mstmaccsz
Single read access to any area	Access size > mstmaccsz	-	2	Burst of accesses of size mstmaccsz. Length of burst: (access size)/mstmaccsz
Read burst to prefetchable area	-	-	0	Burst of accesses of incoming access size up to address boundary defined by rburst.
Read burst to prefetchable area	-	-	1 or 2	Burst of accesses of size mstmaccsz up to address boundary defined by rburst.
Read burst to non-prefetchable area	Access size <= mstmaccsz	-	-	Incremental read burst of same access size as on slave interface, the length is the same as the length of the incoming burst. The master interface will insert BUSY cycles between the sequential accesses.
Read burst to non-prefetchable area	Access size > mstmaccsz	-	1 or 2	Burst of accesses of size mstmaccsz. Length of burst: (incoming burst length)*(access size)/mstmaccsz
Single write	Access size <= mstmaccsz	-	-	Single write access of same size
Single write	Access size > mstmaccsz	1	-	Sequence of single access of mstmaccsz. Number of accesses: (access size)/mstmaccsz.
Single write	Access size > mstmaccsz	2	-	Burst of accesses of mstmaccsz. Length of burst: (access size)/mstmaccsz.

Table 25. Read and write combining

Access on slave interface	Access size	wrcomb	rdcomb	Resulting access(es) on master interface
Write burst	-	0	-	Burst of same size as incoming burst, up to address boundary defined by VHDL generic wburst.
Write burst	-	1 or 2	-	Burst write of maximum possible size. The bridge will use the maximum size (up to mst-maccsz) that it can use to empty the writebuffer.

Read and write combining prevents the bridge from propagating fixed length bursts and wrapping bursts. See section 2.2.7 for a discussion on burst operation.

Read and write combining with VHDL generics *wrcomb/rdcomb* set to 1 cause the bridge to use single accesses when divding an incoming access into several smaller accesses. This means that another master on the bus may write or read parts of the memory area to be accessed by the bridge before the bridge has read or written all the data. In bi-directional configurations, an incoming access on the master bridge may cause a collision that aborts the operation on the slave bridge. This may cause the bridge to read the same memory locations twice. This is normally not a problem when accessing memory areas. The same issues apply when using an AHB arbiter that performs early burst termination. The standard GRLIB AHBCTRL core does not perform early burst termination.

To ensure that the bridge does not re-read an address, and that all data in an access from the bridge's slave interface is propagated out on the master interface without interruption the VHDL generics *rdcomb* and *wrcomb* should both be set to 0 or 2. In addition to this, the AHB arbiter may not perform early burst termination (early burst termination is not performed by the GRLIB AHBCTRL arbiter).

Read and write combining can be limited to specified address ranges. See description of the *combmask* VHDL generic for more information. Note that if the core is implemented with support for prefetch and read combining, it will not obey combmask for prefetch operations (burst read to prefetchable areas). Prefetch operations will always be performed with the maximum allowed size on the master interface.

2.2.7 Burst operation

The core can be configured to support all AMBA 2.0 burst types (single access, incrementing burst of unspecified length, fixed length incrementing bursts and wrapping bursts). Single accesses and incrementing bursts of unspecified length have previously been discussed in this document. An incoming single access will lead to one access, or multiple accesses for some cases with read/write combining, on the other side of the bridge. An incoming incrementing burst of unspecified length to a prefetchable area will lead to the prefetch buffer (if available) being filled using the same access size, or the maximum allowed access size if read/write combining is enabled, on the master interface.

If the core is used in a system where no fixed length bursts or incremental bursts will be used in accesses to the bridge, then set the *allbrst* generic to 0 and skip the remainder of this section.

The VHDL generic *allbrst* controls if the core will support fixed length and wrapping burst accesses. If *allbrst* is set to 0, the core will treat all burst accesses as incrementing of unspecified length. For fixed length and wrapping bursts this can lead to performance penalties and malfunctions. Support for fixed length and wrapping bursts is enabled by setting *allbrst* to 1 or 2. Table 26 describes how the core will handle different burst types depending on the setting of *allbrst*.

Table 26. Burst handling

Value of allbrst generic	Access type*	Undefined length incrementing burst INCR	Fixed length incrementing burst INCR{4,8,16}	Wrapping burst WRAP{4,8,16}
0	Reads to non-prefetchable area	Incrementing burst with BUSY cycles inserted. Same behaviour with read and write combining.	Fixed length burst with BUSY cycles inserted. If the burst is short then the burst may end with a BUSY cycle. If access combining is used the HBURST signal will get incorrect values.	Malfunction. Not supported
	Reads to prefetchable area	Incrementing burst of maximum allowed size, filling prefetch buffer, starting at address boundary defined by prefetch buffer.		Malfunction. Not supported
	Write burst	Incrementing burst	Incrementing burst, if write combining is enabled, and triggered, the burst will be translated to an incrementing burst of undefined length. VHDL generic *wrcomb* should not be set to 1 (but to 0 or 2) in this case	Write combining is not supported. Same access size will be used on both sides of the bridge.
1	Reads to non-prefetchable area	Incrementing burst with BUSY cycles inserted. Same behaviour with read and write combining.	Same burst type with BUSY cycles inserted. If read combining is enabled, and triggered by the incoming access size, an incremental burst of unspecified length will be used. If the burst is short then the burst may end with a BUSY cycle.	Same burst type with BUSY cycles inserted. If read combining is enabled, and triggered by the incoming access size, an incremental burst of unspecified length will be used. This will cause AMBA violations if the wrapping burst does not start from offset 0.
	Reads to prefetchable area	Incrementing burst of maximum allowed size, filling prefetch buffer.	For reads, the core will perform full (or part that fits in prefetch buffer) fixed/wrapping burst on master interface and then respond with data. No BUSY cycles are inserted. If the access made to the slave interface is larger than the maximum supported access size on the master interface then a incrementing burst of unspecified length will be used to fill the prefetch buffer. This (read combining) is not supported for wrapping bursts.	
	Write burst	Same as for allbrst = 0		
2	Reads to non-prefetchable area	Incrementing burst with BUSY cycles inserted. Same behaviour with read and write combining.	Reads are treated as a prefetchable burst. See below.	
	Reads to prefetchable area	Incrementing burst of maximum allowed size, filling prefetch buffer, starting at address boundary defined by prefetch buffer.	Core will perform full (or part that fits in prefetch buffer) fixed/wrapping burst on master interface and then respond with data. No BUSY cycles are inserted. If the access made to the slave interface is larger than the maximum supported access size on the master interface then a incrementing burst of unspecified length will be used to fill the prefetch buffer. This (read combining) is not supported for wrapping bursts.	
	Write burst	Same as for allbrst = 0		

* Access to prefetchable area where the core's prefetch buffer is ised (VHDL generic pfen /= 0).

2.2.8 Transaction ordering, starvation and AMBA arbitration schemes

The bridge is configured at implementation to use one of two available schemes to handle incoming accesses. The bridge will issue SPLIT responses when it is busy and on incoming read accesses. If the bridge has been configured to use first-come, first-served ordering it will keep track of the order of incoming accesses and serve the requests in the same order. If first-come, first-served ordering is disabled the bridge will give some advantage to the master it has a response for and then allow all masters in to arbitration simultaneously, moving the decision on which master that should be allowed to access the bridge to the bus arbitration.

When designing a system containing a bridge the expected traffic patterns should be analyzed. The designer must be aware how SPLIT responses affect arbitration and how the selected transaction ordering in the bridge will affect the system. The two different schemes are further described in sections 2.2.9 and 2.2.10.

2.2.9 First-come, first-served ordering

First-come, first served ordering is used when the VHDL generic *fcfs* is non-zero.

With first-come, first-served ordering the bridge will keep track of the order of incoming accesses. The accesses will then be served in the same order. For instance, if master 0 initiates an access to the bridge, followed by master 3 and then master 5, the bridge will propagate the access from master 0 (and respond with SPLIT on a read access) and then respond with SPLIT to the other masters. When the bridge has a response for master 0, this master will be allowed in arbitration again by the bridge asserting HSPLIT. When the bridge has finished serving master 0 it will allow the next queued master in arbitration, in this case master 3. Other incoming masters will receive SPLIT responses and will not be allowed in arbitration until all previous masters have been served.

An incoming locked access will always be given precedence over any other masters in the queue.

A burst that has initiated a pre-fetch operation will receive SPLIT and be inserted last in the master queue if the burst is longer than the maximum burst length that the bridge has been configured for.

It should be noted that first-come, first-served ordering may not work well in systems where an AHB master needs to have higher priority compared to the other masters. The bridge will not prioritize any master, except for masters performing locked accesses.

2.2.10 Bus arbiter ordering

Bus arbiter ordering is used when VHDL generic *fcfs* is set to zero.

When several masters have received SPLIT and the bridge has a response for one of these masters, the master with the queued response will be allowed in to bus arbitration by the bridge asserting the corresponding HSPLIT signal. In the following clock cycle, all other masters that have received SPLIT responses will also be allowed in bus arbitration as the bridge asserts their HSPLIT signals simultaneously. By doing this the bridge defers the decision on the master to be granted next to the AHB arbiter. The bridge does not show any preference based on the order in which it issued SPLIT responses to masters, except to the master that initially started a read or write operation. Care has been taken so that the bridge shows a consistent behavior when issuing SPLIT responses. For instance, the bridge could be simplified if it could issue a SPLIT response just to be able to change state, and not initiate a new operation, to an access coming after an access that read out prefetched data. When the bridge entered its idle state it could then allow all masters in bus arbitration and resume normal operation. That solution could lead to starvation issues such as:

T0: Master 1 and Master 2 have received SPLIT responses, the bridge is prefetching data for Master 1

T1: Master 1 is allowed in bus arbitration by setting the corresponding HSPLIT

T2: Master 1 reads out prefetch data, Master 2 HSPLIT is asserted to let Master 2 in to bus arbitration

T3: Master 2 performs an access, receives SPLIT, however the bridge does not initiate an access, it just stalls in order to enter its idle state.

T4: Master 2 is allowed in to bus arbitration, Master 1 initiates an access that leads to a prefetch and Master 1 receives a SPLIT response

T5: Master 2 performs an access, receives SPLIT since the bridge is prefetching data for master 1

T6: Go back to T0

This pattern will repeat until Master 1 backs away from the bus and Master 2 is able to make an access that starts an operation over the bridge. In most systems it is unlikely that this behavior would introduce a bus lock. However, the case above could lead to an unexpectedly long time for Master 2 to complete its access. Please note that the example above is illustrative and the problem does not exist in the core as the core does not issue SPLIT responses to (non-locked) accesses in order to just change state but a similar pattern could appear as a result of decisions taken by the AHB arbiter if Master 1 is given higher priority than Master 2.

In the case of write operations the scenario is slightly different. The bridge will accept a write immediately and will not issue a SPLIT response. While the bridge is busy performing the write on the master side it will issue SPLIT responses to all incoming accesses. When the bridge has completed the write operation on the master side it will continue to issue SPLIT responses to any incoming access until there is a cycle where the bridge does not receive an access. In this cycle the bridge will assert HSPLIT for all masters that have received a SPLIT response and return to its idle state. The first master to access the bridge in the idle state will be able to start a new operation. This can lead to the following behavior:

T0: Master 1 performs a write operation, does NOT receive a SPLIT response

T1: Master 2 accesses the bridge and receives a SPLIT response

T2: The bridge now switches state to idle since the write completed and asserts HSPLIT for Master 2.

T3: Master 1 is before Master 2 in the arbitration order and we are back at T0.

In order to avoid this last pattern the bridge would have to keep track of the order in which it has issued SPLIT responses and then assert HSPLIT in the same order. This is done with first-come, first-served ordering described in section 2.2.9.

2.2.11 AMBA SPLIT support

Support for AMBA SPLIT responses is enabled/disabled through the VHDL generic *split*. SPLIT support should be enabled in most systems. The benefits of using SPLIT responses is that the bus on the bridge's slave interface side can be free while the bridge is performing an operation on the master side. This will allow other masters to access the bus and generally improve system performance. The use of SPLIT responses also allows First-come, first-served transaction ordering.

For configurations where the bridge is the only slave interface on a bus, it can be beneficial to implement the bridge without support for AMBA SPLIT responses. Removing support for SPLIT responses reduces the area used by the bridge and may also reduce the time required to perform accesses that traverse the bridge. It should be noted that building a bi-directional bridge without support for SPLIT responses will increase the risk of access collisions.

If SPLIT support is disabled the bridge will insert wait states where it would otherwise issue a SPLIT response to a master initiating an access. This means that the arbitration ordering will be left to the bus arbiter and the bridge cannot be implemented with the First-come, first-served transaction ordering scheme. The bridge will still issue RETRY responses to resolve dead lock conditions, to split up long burst and also when the bridge is busy emptying it's write buffer on the master side.

2.2.12 Core latency

The delay incurred when performing an access over the core depends on several parameters such as core configuration, the operating frequency of the AMBA buses, AMBA bus widths and memory access patterns. Table 27 below shows core behavior in a system where both AMBA buses are running at the same frequency and the core has been configured to use AMBA SPLIT responses. Table 28 further down shows core behavior in the same system without support for SPLIT responses.

Table 27. Example of single read with FFACT = 1, and SPLIT support

Clock cycle	Core slave side activity	Core master side activity
0	Discovers access and transitions from idle state	Idle
1	Slave side waits for master side, SPLIT response is given to incoming access, any new incoming accesses also receive SPLIT responses.	Discovers slave side transition. Master interface output signals are assigned.
2		If bus access is granted, perform address phase. Otherwise wait for bus grant.
3		Register read data and transition to data ready state.
4	Discovers that read data is ready, assign read data output and assign SPLIT complete	Idle
5	SPLIT complete output is HIGH	
6	Typically a wait cycle for the SPLIT:ed master to be allowed into arbitration. Core waits for master to return. Other masters receive SPLIT responses.	
7	Master has been allowed into arbitration and performs address phase. Core keeps HREADY high	
8	Access data phase. Core has returned to idle state.	

Table 28. Example of single read with FFACT = 1, without SPLIT support

Clock cycle	Core slave side activity	Core master side activity
0	Discovers access and transitions from idle state	Idle
1	Slave side waits for master side, wait states are inserted on the AMBA bus.	Discovers slave side transition. Master interface output signals are assigned.
2		Bus access is granted, perform address phase.
3		Register read data and transition to data ready state.
4	Discovers that read data is ready, assign HREADY output register and data output register.	Idle
5	HREADY is driven on AMBA bus. Core has returned to idle state	

While the transitions shown in tables 27 and 28 are simplified they give an accurate view of the core delay. If the master interface needs to wait for a bus grant or if the read operation receives wait states, these cycles must be added to to the cycle count in the tables. The behavior of the core with a fre-

quency factor of two between the buses is shown in tables 29 and 30 (best case, delay may be larger depending on on which slave clock cycle an access is made to the core).

Table 29. Example of single read with FFACT = 2, Master freq. > Slave freq, without SPLIT support

Slave side clock cycle	Core slave side activity	Master side clock cycle	Core master side activity
0	Discovers access and transitions from idle state	0	Discovers slave side transition. Master interface output signals are assigned.
1	Slave side waits for master side, wait states are inserted on the AMBA bus.		
2		1	Bus access is granted, perform address phase.
3			
4		2	Register read data and transition to data ready state.
5			
6	Discovers that read data is ready, assign HREADY output register and data output register.	3	Idle
7	HREADY is driven on AMBA bus. Core has returned to idle state		

Table 30. Example of single read with FFACT = 2, Master freq. > Slave freq, without SPLIT support

Slave side clock cycle	Core slave side activity	Master side clock cycle	Core master side activity
0	Discovers access and transitions from idle state	0	Idle
		1	
1	Slave side waits for master side, wait states are inserted on the AMBA bus.	2	Discovers slave side transition. Master interface output signals are assigned.
		3	Bus access is granted, perform address phase.
2	Discovers that read data is ready, assign HREADY output register and data output register.	4	Register read data and transition to data ready state.
		5	Idle
3	HREADY is driven on AMBA bus. Core has returned to idle state	6	
		7	

Table 31 below lists the delays incurred for single operations that traverse the bridge while the bridge is in its idle state. The second column shows the number of cycles it takes the master side to perform the requested access, this column assumes that the master slave gets access to the bus immediately and that each access is completed with zero wait states. The table only includes the delay incurred by traversing the core. For instance, when the access initiating master reads the core's prefetch buffer, each additional read will consume one clock cycle. However, this delay would also have been present if the master accessed any other slave.

Write accesses are accepted with zero wait states if the bridge is idle, this means that performing a write to the idle core does not incur any extra latency. However, the core must complete the write operation on the master side before it can handle a new access on the slave side. If the core has not transitioned into its idle state, pending the completion of an earlier access, the delay suffered by an access be longer than what is shown in the tables in this section. Accesses may also suffer increased delays during collisions when the core has been instantiated to form a bi-directional bridge. Locked accesses that abort on-going read operations will also mean additional delays.

If the core has been implemented to use AMBA SPLIT responses there will be an additional delay where, typically, one cycle is required for the arbiter to react to the assertion of HSPLIT and one clock cycle for the repetition of the address phase.

Note that if the core has support for read and/or write combining, the number of cycles required for the master will change depending on the access size and length of the incoming burst access. For instance, in a system where the bus in the core's master side is wider than the bus on the slave side, write combining will allow the core to accept writes with zero wait states and then combine several accesses into one or several larger access. Depending on memory controller implementation this could reduce the time required to move data to external memory, and will reduce the load on the master side bus.

Table 31. Access latencies

Access	Master acc. cycles	Slave cycles	Delay incurred by performing access over core
Single read	3	1	1 * clk_{slv} + 3 * clk_{mst}
Burst read with prefetch	2 + (burst length)[x]	2	2 * clk_{slv} + (2 + burst length)* clk_{mst}
Single write[xx]	(2)	0	0
Burst write[xx]	(2 + (burst length))	0	0

[x] A prefetch operation ends at the address boundary defined by the prefetch buffer's size

[xx] The core implements posted writes, the number of cycles taken by the master side can only affect the next access.

2.2.13 Endianness

The core is designed for big-endian systems.

2.3 Registers

The core does not implement any registers.

2.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x020. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

2.5 Implementation

2.5.1 Technology mapping

The uni-directional AHB to AHB bridge has two technology mapping generics *memtech* and fcfsmtech. *memtech* selects which memory technology that will be used to implement the FIFO memories. *fcfsmtech* selects the memory technology to be used to implement the First-come, first-served buffer, if FCFS is enaled.

2.5.2 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

2.5.3 RAM usage

The uni-directional AHB to AHB bridge instantiates one or several *syncram_2p* blocks from the technology mapping library (TECHMAP). If prefetching is enabled max(*mstmaccsz*, *slvaccsz*)/32 *syncram_2p* block(s) with organization (max*(rburst,iburst*)-max(*mstmaccsz*, *slvaccsz*)/32) x 32 is used to implement read FIFO (max*(rburst,iburst*) is the size of the read FIFO in 32-bit words). max(*mstmaccsz*, *slvaccsz*)/32 *syncram_2p* block(s) with organization *(wburst* - max(*mstmaccsz*, *slvaccsz*)/32) x 32, is always used to implement the write FIFO (where *wburst* is the size of the write FIFO in 32-bit words).

If the core has support for first-come, first-served ordering then one *fcfs* x 4 *syncram_2p* block will be instantiated, using the technology specified by the VHDL generic *fcfsmtech*.

2.5.4 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

2.6 Configuration options

Table 32 shows the configuration options of the core (VHDL generics).

Table 32. Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
memtech	Memory technology		
hsindex	Slave I/F AHB index	0 to NAHBMAX-1	0
hmindex	Master I/F AHB index	0 to NAHBMAX-1	0
dir	0 - clock frequency on the master bus is lower than or equal to the frequency on the slave bus 1 - clock frequency on the master bus is higher than or equal to the frequency on the slave bus (for VHDL generic *ffact* = 1 the value of dir does not matter)	0 - 1	0
ffact	Frequency scaling factor between AHB clocks on master and slave buses.	1 - 15	2
slv	Slave bridge. Used in bi-directional bridge configuration where *slv* is set to 0 for master bridge and 1 for slave bridge. When a deadlock condition is detected slave bridge (*slv*=1) will give RETRY response to current access, effectively resolving the deadlock situation. This generic must only be set to 1 for a bridge where the frequency of the bus connecting the master interface is higher or equal to the frequency of the AHB bus connecting to the bridge's slave interface. Otherwise a race condition during access collisions may cause the bridge to deadlock.	0 - 1	0
pfen	Prefetch enable. Enables read FIFO.	0 - 1	0
irqsync	Interrupt forwarding. Forward interrupts from slave interface to master interface and vice versa. 0 - no interrupt forwarding, 1 - forward interrupts 1 - 15, 2 - forward interrupts 0 - 31. 3 - forward interrupts 0 - 63 (should only be used if GRLIB has been configured to support 64 interrupt lines). Since interrupts are forwarded in both directions, interrupt forwarding should be enabled for one bridge only in a bi-directional AHB/AHB bridge.	0 - 3	0

Table 32. Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
wburst	Length of write bursts in 32-bit words. Determines write FIFO size and write burst address boundary. If the wburst generic is set to 2 the bridge will not perform write bursts over a 2x4=8 byte boundary. This generic must be set so that the buffer can contain two of the maximum sized accesses that the bridge can handle.	2 - 32	8
iburst	Instruction fetch burst length. This value is only used if the generic *ibrsten* is set to 1. Determines the length of prefetching instruction read bursts on the master side. The maximum of (iburst,rburst) determines the size of the core's read buffer FIFO.	4 - 8	8
rburst	Incremental read burst length. Determines the maximum length of incremental read burst of unspecified length (INCR) on the master interface. The maximum of *rburst* and *iburst* determine the read burst boundary. As an example, if the maximum value of these generics is 8 the bridge will not perform read bursts over a 8x4=32 byte boundary. This generic must be set so that the buffer can contain two of the maximum sized accesses that the bridge can handle. For systems where AHB masters perform fixed length burst (INCRx , WRAPx) *rburst* should not be less than the length of the longest fixed length burst.	4 - 32	8
bar0	Address area 0 decoded by the bridge's slave interface. Appears as memory address register (BAR0) on the slave interface. The generic has the same bit layout as bank address registers with bits [19:18] suppressed (use functions ahb2ahb_membar and ahb2ahb_iobar in gaisler.misc package to generate this generic).	0 - 1073741823	0
bar1	Address area 1 (BAR1)	0 - 1073741823	0
bar2	Address area 2 (BAR2)	0 - 1073741823	0
bar3	Address area 3 (BAR2)	0 - 1073741823	0
sbus	The number of the AHB bus to which the slave interface is connected. The value appears in bits [1:0] of the user-defined register 0 in the slave interface configuration record and master configuration record.	0-3	0
mbus	The number of the AHB bus to which the master interface is connected. The value appears in bits [3:2] of the user-defined register 0 in the slave interface configuration record and master configuration record.	0-3	0
ioarea	Address of the I/O area containing the configuration area for AHB bus connected to the bridge's master interface. This address appears in the bridge's slave interface user-defined register 1. In order for a master on the slave interface's bus to access the configuration area on the bus connected to the bridge's master interface, the I/O area must be mapped on one of the bridge's BARs. If this generic is set to 0, some tools, such as Frontgrade Gaisler's GRMON debug monitor, will not perform Plug'n'Play scanning over the bridge.	0 - 16#FFF#	0
ibrsten	Instruction fetch burst enable. If set, the bridge will perform bursts of *iburst* length for opcode access (HPROT[0] = '0'), otherwise bursts of *rburst* length will be used for both data and opcode accesses.	0 - 1	0

Table 32. Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
lckdac	Locked access error detection and correction. Locked accesses may lead to deadlock if a locked access is made while an ongoing read access has received a SPLIT response. The value of *lckdac* determines how the core handles this scenario: 0: Core will deadlock 1: Core will issue an AMBA ERROR response to the locked access 2: Core will allow both accesses to complete. If the core is used to create a bidirectional bridge, a deadlock condition may arise when locked accesses are made simultaneously in both directions. With *lckdac* set to 0 the core will deadlock. With *lckdac* set to a non-zero value the slave bridge will issue an ERROR response to the incoming locked access.	0 - 2	0
slvmaccsz	The maximum size of accesses that will be made to the bridge's slave interface. This value must equal *mstmaccsz* unless *rdcomb* /= 0 and *wrcomb* /= 0.	32 - 256	32
mstmaccsz	The maximum size of accesses that will be performed by the bridge's master interface. This value must equal *mstmaccsz* unless *rdcomb* /= 0 and *wrcomb* /= 0.	32 - 256	32
rdcomb	Read combining. If this generic is set to a non-zero value the core will use the master interface's maximum AHB access size when prefetching data and allow data to be read out using any other access size supported by the slave interface. If slvmaccsz > 32 and mstmaccsz > 32 and an incoming single access, or access to a non-prefetchable area, is larger than the size supported by the master interface the bridge will perform a series of small accesses in order to fetch all the data. If this generic is set to 2 the core will use a burst of small fetches. If this generic is set to 1 the bridge will not use a burst unless the incoming access was a burst. Read combining is only supported for single accesses and incremental bursts of unspecified length.	0 - 2	0
wrcomb	Write combining. If this generic is set to a non-zero value the core may assemble several small write accesses (that are part of a burst) into one or more larger accesses or assemble one or more accesses into several smaller accesses. The settings are as follows: 0: No write combining 1: Combine if burst can be preserved 2: Combine if burst can be preserved and allow single accesses to be converted to bursts (only applicable if slvmaccsz > 32) Only supported for single accesses and incremental bursts of unspecified length	0 - 2	0

Table 32. Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
combmask	Read/write combining mask. This generic determines which ranges that the core can perform read/write combining to (only available when rdcomb respectively wrcomb are non-zero). The value given for combmask is treated as a 16-bit vector with LSB bit (right-most) indicating address 0x0 - 0x10000000. Making an access to an address in an area marked as '0' in combmask is equivalent to making an access over a bridge with rdcomb = 0 and wrcomb = 0. However, combmask is not taken into account when the core performs a prefetch operation (see pfen generic). When a prefetch operation is initiated, the core will always use the maximum supported access size (when rdcomb /= 0).	0 - 16#FFFF#	16#FFFF#
allbrst	Support all burst types 2: Support all types of burst and always prefetch for wrapping and fixed length bursts. 1: Support all types of bursts 0: Only support incremental bursts of unspecified length See section 2.2.7 for more information. When allbrst is enabled, the core's read buffer (size set via rburst/iburst generics) must have at least 16 slots.	0 - 2	0
ifctrlen	Interface control enable. When this generic is set to 1 the input signals *ifctrl.mstifen* and *ifctrl.slvifen* can be used to force the AMBA slave respectively master interface into an idle state. This functionality is intended to be used when the clock of one interface has been gated-off and any stimuli on one side of the bridge should not be propagated to the interface on the other side of the bridge. When this generic is set to 0, the ifctrl.* input signals are unused.	0 - 1	0
fcfs	First-come, first-served operation. When this generic is set to a non-zero value, the core will keep track of the order of incoming accesses and handle the requests in the same order. If this generic is set to zero the bridge will not preserve the order and leave this up to bus arbitration. If FCFS is enabled the value of this generic must be higher or equal to the number of masters that may perform accesses over the bridge.	0 - NAHBMST	0
fcfsmtech	Memory technology to use for FCFS buffer. When VHDL generic *fcfs* is set to a non-zero value, the core will instantiate a 4 bit x *fcfs* buffer to keep track of the incoming master indexes. This generic decides the memory technology to use for the buffer.	0 - NTECH	0 (inferred)
scantest	Enable scan support	0 - 1	0
split	Use AMBA SPLIT responses. When this generic is set to 1 the core will issue AMBA SPLIT responses. When this generic is set to 0 the core will insert waitstates instead and may also issue AMBA RETRY responses. If this generic is set to 0, the *fcfs* generic must also be set to 0, otherwise a simulation failure will be asserted.	0 - 1	1

Table 32. Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
pipe	This setting controls the insertion of pipeline registers between the master and slave side of the bridge. *pipe* set to 0 does not include any extra pipeline registers and the incurred delays for accesses over the bridge is as described in this documentation. *pipe* set to 1 includes extra registers on all signals between the master and slave side. *pipe* set to 2 includes pipeline registers on all signals going from the slave interface to the master interface and does NOT insert extra registers on signals going from the master interface to the slave interface. *pipe* set to 3 includes pipeline registers on all signals going from the master interface to the slave interface and does NOT insert extra registers on signals going from the slave interface to the master interface. *pipe* set to 128 includes signals on a subset of the signals to prevent direct paths from the slave clock to the master side bus and from the master clock to the slave side bus.	0, 1, 128	

2.7 Signal descriptions

Table 33 shows the interface signals of the core (VHDL ports).

Table 33. Signal descriptions (VHDL ports)

Signal name	Field	Type	Function	Active
RST		Input	Reset	Low
HCLKM		Input	AHB master bus clock	-
HCLKS		Input	AHB slave bus clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
AHBSO2	*	Input	AHB slave input vector signals (on master i/f side). Used to decode cachability and prefetch-ability Plug&Play information on bus connected to the bridge's master interface.	-
LCKI	slck blck mlck	Input	Used in systems with multiple AHB/AHB bridges (e.g. bi-directional AHB/AHB bridge) to detect deadlock conditions. Tie to "000" in systems with only uni-directional AHB/AHB bus.	High
LCKO	slck blck mlck	Output	Indicates possible deadlock condition	High
IFCTRL	mstifen	Input	Enable master interface. This input signal is unused if the VHDL generic *ifctrlen* is 0. If VHDL generic *ifctrlen* is 1 this signal must be set to '1' in order to enable the core's AMBA master interface, otherwise the master interface will always be idle and will not respond to stimuli on the core's AMBA slave interface. This signal is intended to be used to keep the core's master interface in a good state when the core's slave interface clock has been gated off. Care should be taken to ensure that the bridge is idle when the master interface is disabled.	High
	slvifen	Input	Enable slave interface. This input signal is unused if the VHDL generic *ifctrlen* is 0. If VHDL generic *ifctrlen* is 1 this signal must be set to '1' in order to enable the core's AMBA slave interface, otherwise the interface will always be ready and the bridge will not propagate stimuli on the core's AMBA slave interface to the core's AMBA master interface. This signal is intended to be used to keep the slave interface in a good state when the core's master interface clock has been gated off. Care should be taken to ensure that the bridge is idle when the slave interface is disabled.	High

* see GRLIB IP Library User's Manual

2.8 Library dependencies

Table 34 shows the libraries used when instantiating the core (VHDL libraries).

Table 34. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

2.9 Instantiation

GRLIB contains an example design with AHB2AHB and LEON processors: *designs/leon3-ahb2ahb* (only available in commercial distributions). The LEON/GRLIB Configuration and Development Guide contains more information on how to use the bridge to create multi-bus systems.

3 AHB2AXIB - AHB to AXI Bridge

3.1 Overview

The AHB2AXIB bridge allows to access an AXI3 or AXI4 slave from an AHB bus through an AHB slave interface (see Fig. 2). It can also be used to connect a standalone AHB master to an AXI slave (see Fig. 3). The bridge has an AHB slave interface on the AHB side and AXI3 or AXI4 master interface on the AXI side. The bridge has optional read prefetching and write buffering features in order to improve the latency of burst operations. The AHB2AXIB bridge is not compatible with the AHB2AHB bridge which is a part of GRLIB IP library.

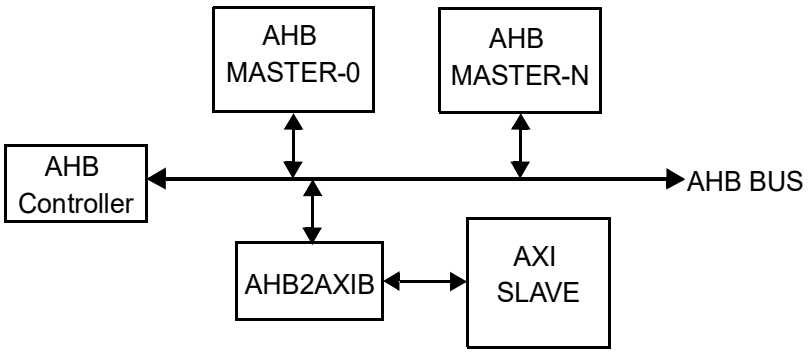


Figure 2. An AXI slave connected to the AHB bus through AHB2AXIB bridge



Figure 3. A standalone AHB master is connected to an AXI slave through AHB2AXIB bridge

3.1.1 AHB support

The AHB2AXIB bridge currently supports the following features of the AHB protocol:

Transfer Type: IDLE, NONSEQ, SEQ

Burst Operation: SINGLE, INCR, INCR4, INCR8, INCR16

Address Width: 32-bit by default. Can be increased with *extra_awidth* generic. However, Plug and Play scanning will not be fully supported if the generic is set to a non-zero value. Connecting the extended address bits is done via the dedicated input/output signals, i.e. not via the AMBA records.

Data Width: 32-bit, 64-bit, 128-bit, 256-bit

Transfer Size: All possible transfer sizes up to the selected data-width are supported.

Response: AXI read error response is translated to AHB read error.

Unsupported AHB Features:

The following features of AHB protocol are not supported by the AHB2AXIB bridge.

- BUSY transfer type: Behavior of the AHB2AXIB bridge is unpredictable when a BUSY transaction is received hence the AHB2AXIB bridge can not be used with the AHB2AHB bridge which is a part of GRLIB IP library.

- Locked transfers: Locked transfers are ignored by the AHB2AXIB bridge.

Unused AHB Features:

- RETRY and SPLIT responses: AHB2AXIB bridge does not generate these response types.
- Write error response: Due to the difference between the write error handling of AXI and AHB protocol the write errors received from the AXI side is not propagated.

3.2 Operation

3.2.1 Read Prefetching and Write Buffering and Postponed Writes

The bridge has the feature of read prefetching and write buffering for the AHB bursts in which the transfer size (HSIZE) is equal to the selected data-width. For the transfer sizes that are narrower than the data-width it can still support read prefetching and write buffering if byte invariant big endian mode is used. Otherwise each beat in the burst treated as a single transaction on the AXI side. Read prefetching and write buffering reduces the latency of undefined length burst operations since otherwise each beat in an undefined length burst has to be treated as an independent AXI transaction with a length of one.

3.2.2 Read Prefetching

Read prefetch number that is set through *rprefetch_num* generic determines the length of the AXI transaction(s) that is generated when an undefined length AHB read burst is encountered. When an undefined length AHB read burst is encountered, an AXI transaction is generated with a length of *rprefetch_num.* If the AHB read burst has less beats than *rprefetch_num* then dummy reads are generated on the AXI side to complete the AXI transaction. If the AHB read burst has more beats than *rprefetch_num* then a new AXI transaction is generated with a number of beats equal to *rprefetch_num* and this scheme continuous until the AHB burst ends. If the start address of a burst is not aligned to the prefetch boundary then the initial prefetch has less number of beats in order to align the upcoming prefetches. For example given a 32-bit (4 Byte) data-width and a *rprefetch_num* of 16 (16*4=64 Bytes) if the least significant bits of the initial burst address corresponds to byte 48, then the initial prefetch length is 4 ((64-48)/4). This way the upcoming prefetches are always aligned to 64 Byte boundary. If a new AHB burst is encountered during dummy read operations on the AXI side, the AHB burst is stalled until the current AXI transaction ends.

The maximum read prefetch number depends on the AXI protocol. For AXI3 the maximum number is 16, and for AXI4 it can be up to 256 depending on the selected data-width. Prefetch length can only be a power of two and if it is not set to be a power of two then the number is floored to the closest power of two automatically. For fixed length AHB bursts (single, INCR4, INCR8, INCR16) the length of the AXI burst is equal to the AHB burst length since in those cases the burst length is known at the beginning of the burst. The bridge will not issue a new AXI transaction while dummy cycles are inserted hence there is a trade-off for performance when selecting the read prefetch number.

3.2.3 Write Buffering

Write buffering gathers a number of consecutive beats in a AHB write burst and initiates an AXI transaction. A generic called *wbuffer_num* determines the maximum number of AHB write burst beats that will be gathered before an AXI write burst transaction is generated. If the number of beats in the AHB write burst is less than *wbuffer_num* then the AXI write transaction starts after detecting the last beat in the burst (transition from SEQ to IDLE). If the number of beats are higher than *wbuffer_num* then the first AXI transaction is generated once *wbuffer_num* number of beats are buffered. It should be noted that once an AXI write transaction is generated and AHB write burst still continues then AXI transaction and buffering of the next write batch happens in parallel to improve the latency. This scheme continuous until the AHB burst is ended. The last data beat in the burst is always

acknowledged with OKAY response immediately when it is buffered in the bridge. See section 3.2.5 for more detailed information.

Write buffer length can only be a power of two, and if it is not set to be a power of two then the number is floored to the closest power of two automatically. The maximum number has the same constraints as the read prefetch number. A synchronous memory width one read and write port is generated for write buffering. The size of the memory is determined by the write buffer length. The type of the memory can be configured with a generic also. The first AXI write transaction will not start until the buffer is filled or the AHB transaction has written the last beat in the burst. As a result there is a trade-off for performance while selecting the write buffer length which depends on the AXI slave behavior.

Write transactions which has the property of HBURST_SINGLE does not go through the buffer and has a very low latency. This is intended to improve the performance of LEON3/LEON4 since the processors are write-through and writes are done with HBURST_SINGLE property (apart from the store double in LEON3).

3.2.4 Narrow Sized Transactions

When an AHB transaction is encountered which has a narrower size (HSIZE) than the data-width of the AHB2AXIB bridge, the behavior is configurable through the generics depending on the selected endianness on the AXI side. When the endianness mode on the AXI side is set as little-endian than each beat in the narrow sized AXI transaction is treated as single transaction on the AXI side. When the endianness mode on the AXI side is set as byte invariant big-endian than the *narrow_acc_mode* generic determines the behaviour. If the *narrow_acc_mode* generic is set as zero than each beat in the narrow sized AXI transaction is treated as single transaction on the AXI side. If it is set to 1 then a corresponding narrow sized AXI burst is generated with read prefetching and write buffering. But it should be noted that the length of the narrow sized burst will be determined by *rprefetch_num* and *wbuffer_num* generics and it is same as for all access sizes. When the endianness on the AXI side is set ass little-endian then *narrow_acc_mode* generic must be set to zero. See sec. 3.2.6 for more detailed information about endianness modes.

3.2.5 Postponed Writes

Since the write response from AXI is not propagated to AHB side the last beat in the AHB write transaction is acknowledged immediately when it is buffered in the bridge. Hence the corresponding AXI write transaction will finish after the AHB write transaction is completed. The transaction order on the AHB bus side will be preserved because the bridge will block an AHB read, if there is an AXI write transaction is ongoing, until the AXI write response is received. But if a transaction order has to be preserved between the AHB side of a AHB2AXIB bridge and an independent AXI master that accesses to the same AXI slave then special considerations in software might be needed.

The bridge can have more than one outstanding writes on the AXI bus to improve the performance. The number of outstanding writes is determined by “ostand_writes” generic. As soon as outstanding write limit is not reached the bridge will continue operation after the last write in the AXI burst is acknowledged. The write acknowledgment channel is tracked in the background to confirm the overall write transaction. But if the bridge receives a READ operation when an outstanding write operation exists it will stall until all the outstanding writes are cleared. A write operation will cause a stall only if outstanding write limit is reached.

3.2.6 Endianness

The AHB side of AHB2AXIB can be either set to be little-endian or big-endian through “ahb_endianness” generic.

When AHB2AXIB is set to be little-endian then data is mapped directly to AXI since AXI is little-endian also.

When AHB2AXIB is set to be big-endian, the endianness on the AXI side is configurable through the *endianness_mode* generic.

When *endianness_mode* generic is set to zero a byte-invariant big-endian endianness mode is used on the AXI side. In order to translate big-endian AHB to byte-invariant big-endian AXI the byte order is reversed (see Fig. 4). No address translation occurs inside the adapter in this mode.

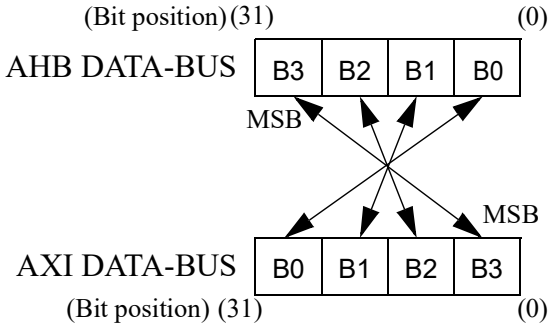


Figure 4. Big-endian AHB to byte-invariant Big-endian AXI translation (32-bit data-width)

When *endianness_mode* generic is set to one then big-endian AHB is translated to little-endian AXI. In order to achieve this the byte order is preserved but the address is translated from big-endian representation to little-endian representation when a narrow sized transaction is encountered (See Fig. 5 for an example with 32-bit data-bus width.).

The address translation formula for 32-bit, 64-bit, 128-bit and 256-bit data-bus widths are following:

32-bit data bus width:

if *HSIZE* < “010” :

axi_address(1:0) = (“100” - “1”<<“*HSIZE*” - ahb_address(1:0))(1:0)

otherwise:

axi_address(1:0) = ahb_address(1:0)

64-bit data bus width:

if *HSIZE* < “011” :

axi_address(2:0) = (“1000” - “1”<<“*HSIZE*” - ahb_address(2:0))(2:0)

otherwise:

axi_address(2:0) = ahb_address(2:0)

128-bit data bus width:

if *HSIZE* < “100” :

axi_address(3:0) = (“10000” - “1”<<“*HSIZE*” - ahb_address(3:0))(3:0)

otherwise:

axi_address(3:0) = ahb_address(3:0)

256-bit data bus width:

if *HSIZE* < “101” :

axi_address(4:0) = (“100000” - “1”<<“*HSIZE*” - ahb_address(4:0))(4:0)

otherwise:

axi_address(4:0) = ahb_address(4:0)

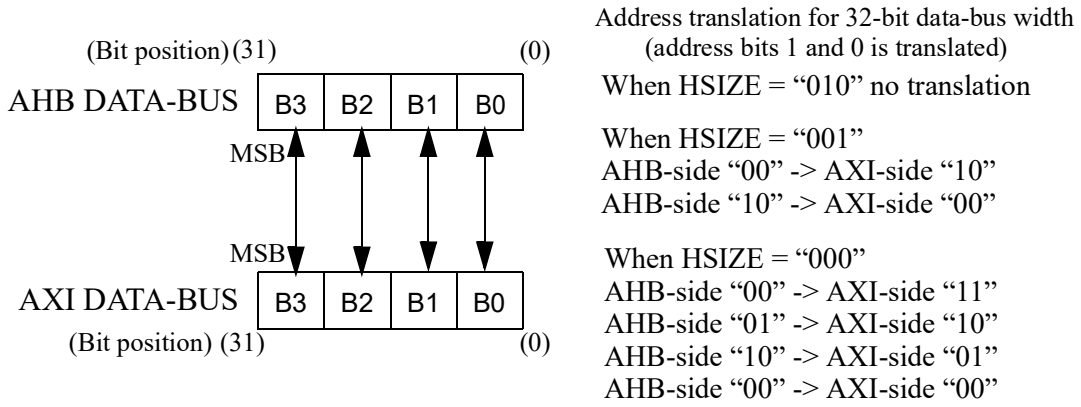


Figure 5. Big-endian AHB to little-endian AXI through address translation (32-bit data-width)

3.3 AXI AxPROT and AxCACHE Translations

The AxPROT and AxCACHE signals are translated partly according to the HPROT signal of AHB transactions. The full list of translation can be seen from Table 35.

Table 35. AxPROT and AxCACHE translations

AXI signal	Assignment
AxCACHE[3]	always logic ‘0’
AxCACHE[2]	always logic ‘0’
AxCACHE[1]	HPROT[3]
AxCACHE[0]	HPROT[2]
AxPROT[2]	not (HPROT[0])
AxPROT[1]	See configuration options (Table. 36)
AxPROT[0]	HPROT[1]

3.4 Implementation

3.4.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User’s Manual). All registers are reset.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

3.4.2 Endianness

The AHB endianness can be configured with the *ahb_endianness* VHDL generic while the AXI endianness can be configured with the *endianness_mode* VHDL generic. The *ahb_endianness* VHDL generic is by default set to the value of the GRLIB configuration package (see GRLIB User’s Manual).

3.5 Configuration Options

Table 36. Configuration options (both AHB2AXI3B and AHB2AXI4B)

Generic	Function	Allowed range	Default
aximid	AXI master ID used for Read and Write transactions	0 - 15	0
always_secure	When set to 1 the AxPROT[1] bit is tied to logic '0' (always secure access), when set to 0 the AxPROT[1] bit is tied to logic '1' (always unsecure access).	0-1	1
ahb_endianness	Determines the endianness of the AHB side of the bridge. 0 -> Big-endian 1 -> Little-endian The default value is taken from the endianness generic set in grlib configuration. "grlib_little_endian"	0-1	GRLIB
endianness_mode	Determines the endianness mode (see section 3.2.6 for more detail) 0 -> Big-endian AHB to byte-invariant big-endian AXI 1 -> Big-endian AHB to little-endian AXI Set it to 0 for little-endian "ahb_endianness" mode	0-1	0
narrow_acc_mode	Determines if bursts with narrow access size than the data-bus width should be directly translated to narrow access size AXI bursts or single AXI transactions with narrow access size. (see section 3.2.4 for more detail) 0-> Each beat in the narrow sized AHB burst is treated as single transaction on the AXI side. 1-> Narrow sized AHB bursts are translated to narrow sized AXI bursts. (supported only when *endianness_mode* generic is 0) **Note:** This generic must be set to 0 if *endianness_mode* is set to 1.	0-1	0
extra_awidth	Extra AMBA Address width. Plug and Play not fully supported if set to a non-zero value.	0 - 96	0
vendor	GRLIB plug&play vendor ID		GAISLER
device	GRLIB plug&play device ID		AHB2AXI
bar0	Address area 0 decoded by the bridge's slave interface. Appears as memory address register (BAR0) on the slave interface. The generic has the same bit layout as bank address registers with bits [19:18] suppressed (use functions ahb2ahb_membar and ahb2ahb_iobar in gaisler.misc package to generate this generic).	0 - 1073741823	0
bar1	Address area 1 (BAR1)	0 - 1073741823	0
bar2	Address area 2 (BAR2)	0 - 1073741823	0
bar3	Address area 3 (BAR2)	0 - 1073741823	0

Table 37. Configuration options specific for AXI3 (AHB2AXI3B)

Generic	Function	Allowed range	Default
wbuffer_num	Write-buffer length which determines the memory size also.	1-16	8
rprefetch_num	Read prefetch length.	1-16	8

Table 38. Configuration options specific for AXI4 (AHB2AXI4B)

Generic	Function	Allowed range	Default
wbuffer_num	Write-buffer length which determines the memory size also.	1-256 for data-width of 32-bit, 1-128 for data-width of 64-bit 1-64 for data-width of 128-bit 1-32 for data-width of 256-bit	8
rprefetch_num	Read prefetch length.	1-256 for data-width of 32-bit, 1-128 for data-width of 64-bit 1-64 for data-width of 128-bit 1-32 for data-width of 256-bit	8
ostand_writes	Number of outstanding writes allowed on the AXI bus. See section 3.2.5 for more details.	1-16	4

3.6 Signal descriptions

Table 39 shows the interface signals of the core (VHDL ports).

Table 39. Signal descriptions (VHDL ports)

Signal name	Field	Type	Function	Active
RST		Input	Reset	Low
CLK		Input	AHB & AXI bus clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
AXIMI	*	Input	AXI3/4 master input signals	-
AXIMO	*	Output	AXI3/4 master output signals	-
i_extra_addr	*	Input	Extra address for AHB	-
o_extra_r_addr	*	Output	Extra address for AXI Read Interface	-
o_extra_w_addr	*	Output	Extra address for AXI Write Interface	-

* see GRLIB IP Library User's Manual

3.7 Library dependencies

Table 40 shows the libraries used when instantiating the core (VHDL libraries).

Table 40. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA & AXI signal definitions
GAISLER	AXI	Component	Component declaration

3.8 Instantiation

The instantiation of the AHB2AXIB bridge depends on the AXI protocol type. There are two components called AHB2AXI3B which is built for AXI3 protocol and AHB2AXI4B which is built for AXI4 protocol. The difference between these two components are the AXI master output signals and the maximum values that can be set for read prefetching and write buffering.

3.8.1 AHB2AXIB bridge is used to connect an AXI slave to an AHB bus

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.axi.all;


entity ahb2axib_ex is
  port (
    rstn  : in  std_logic;
    clk   : in  std_logic;
    .
    .
    .
    aximi : in  axi_somi_type;
    aximo : out axi4_mosi_type;
);
end;


architecture rtl of ahb2axib_ex is
.
.
constant hindex_ahb2axi4b : integer := 2;

begin


.
.
ahbctrl & other components
.
.


bridge:ahb2axi4b
generic map (
 hindex => hindex_ahb2axi4b,
 aximid => 0
)
port map (
 rstn => rstn,
 clk => clk,
 ahbsi => ahbsi,
 ahbso => ahbso(hindex_ahb2axi4b),
 aximi => aximi,
 aximo => aximo);
```

3.8.2 AHB2AXIB bridge is used to connect a standalone AHB master to an AXI slave.

If AHB2AXIB bridge is intended to be used to connect a standalone AHB master to an AXI slave then the following assignments are needed for correct operations:

The *hsel* input of the AHB2AXIB must be assigned to an array of (others=>'1') so that it works regardless of the assigned *hindex* value.

The *hready* input of the AHB2AXIB must be connected to the *hready* output of the AHB2AXIB.

The *hgrant* input of the AHB master must be assigned to an array of (others=>'1') so that it works regardless of the assigned *hindex* value.

4 AHBM2AXI - AHB Master to AXI Adapter

4.1 Overview

The AHBM2AXI adapter allows a single AHB master to be used as an AXI3 or AXI4 master. The adapter has an AHB slave interface on the AHB side and AXI3 or AXI4 master interface on the AXI side (see Fig. 6). The adapter has optional read prefetching and write buffering features in order to improve the latency of burst operations. The adapter is not compatible with AHB2AHB and GRMAC components which is a part of GRLIB IP library.

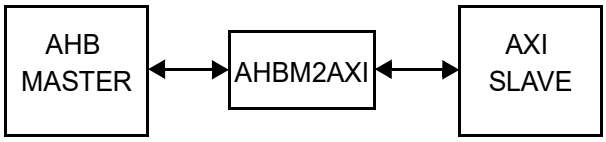


Figure 6. A standalone AHB master is connected to an AXI slave through AHBM2AXI adapter

4.1.1 AHB support

The AHBM2AXI adapter currently supports the following features of the AHB protocol:

Transfer Type: IDLE, NONSEQ, SEQ

Burst Operation: SINGLE, INCR, INCR4, INCR8, INCR16

Data-width: 32-bit, 64-bit, 128-bit, 256-bit

Transfer Size: All possible transfer sizes up to the selected data-width are supported.

Response: AXI read and write error responses are translated to AHB read and write errors.

Unsupported AHB Features:

The following features of AHB protocol are not supported by the AHBM2AXI adapter.

- BUSY transfer type : Behavior of the AHBM2AXI adapter is unpredictable when a BUSY transaction is received hence the AHBM2AXI adapter can not be used with the AHB2AHB bridge which is a part of GRLIB IP library.
- Locked transfers : Locked transfers are ignored by the AHBM2AXI adapter.

Unused AHB Features:

- RETRY and SPLIT responses : The adapter does not generate these response types.

4.2 Special Considerations

There is a combinatorial path between the incoming HTRANS signal and outgoing HREADY signal on the AHB side of the adapter, in order to allow write-buffering and write response propagation at the same time. As a result, this component is only intended to connect to a single IP core with an AHB master interface in which HTRANS output does not depend on the incoming HREADY signal combinatorially. Propagating the write response correctly is important to make sure that the intended transaction ordering has been met, meaning the AHB master that is connected to the adapter receives the acknowledgment for the last write beat in the burst when the write response has been received on the

AXI side. Being able to propagate correct write response can also simplify the software development. The AHBM2AXI adapter can not be used with the GRDMAC IP core which is a part of GRLIB IP library.

4.3 Operation

4.3.1 Read Prefetching and Write Buffering

The adapter has the feature of read prefetching and write buffering for the AHB bursts in which the transfer size (HSIZE) is equal to the selected data-width. For the transfer sizes that are narrower than the data-width each beat in the burst treated as a single transaction on the AXI side. Read prefetching and write buffering reduces the latency of undefined length burst operations since otherwise each beat in an undefined length burst has to be treated as an independent AXI transaction with a length of one.

4.3.2 Read Prefetching

Read prefetch number that is set through *rprefetch_num* generic determines the length of the AXI transaction(s) that is generated when an undefined length AHB read burst is encountered. When an undefined length AHB read burst is encountered, an AXI transaction is generated with a length of *rprefetch_num.* If the AHB read burst has less beats than *rprefetch_num* then dummy reads are generated on the AXI side to complete the AXI transaction. If the AHB read burst has more beats than *rprefetch_num* then a new AXI transaction is generated with a number of beats equal to *rprefetch_num* and this scheme continuous until the AHB burst ends. If the start address of a burst is not aligned to the prefetch boundary then the initial prefetch has less number of beats in order to align the upcoming prefetches. For example given a 32-bit (4 Byte) data-width and a *rprefetch_num* of 16 (16*4=64 Bytes) if the least significant bits of the initial burst address corresponds to byte 48, then the initial prefetch length is 4 ((64-48)/4). This way the upcoming prefetches are always aligned to 64 Byte boundary. If a new AHB burst is encountered during dummy read operations on the AXI side, the AHB burst is stalled until the current AXI transaction ends.

The maximum read prefetch number depends on the AXI protocol. For AXI3 the maximum number is 16, and for AXI4 it can be up to 256 depending on the selected data-width. Prefetch length can only be a power of two and if it is not set to be a power of two then the number is floored to the closest power of two automatically. For fixed length AHB bursts (single, INCR4, INCR8, INCR16) the length of the AXI burst is equal to the AHB burst length since in those cases the burst length is known at the beginning of the burst. The adapter will not issue a new AXI transaction while dummy cycles are inserted hence there is a trade-off for performance when selecting the read prefetch number.

4.3.3 Write Buffering

Write buffering gathers a number of consecutive beats in a AHB write burst and initiates an AXI transaction. A generic called *wbuffer_num* determines the maximum number of AHB write burst beats that will be gathered before an AXI write burst transaction is generated. If the number of beats in the AHB write burst is less than *wbuffer_num* then the AXI write transaction starts after detecting the last beat in the burst (transition from SEQ to IDLE). If the number of beats are higher than *wbuffer_num* then the first AXI transaction is generated once *wbuffer_num* number of beats are buffered. It should be noted that once an AXI write transaction is generated and AHB burst still continues then AXI transaction and buffering for the next write batch happens in parallel to minimize the latency. This scheme continuous until the AHB burst is ended. When the last data beat of the burst is reached the HREADY on the AHB side is asserted once the write response is received from the AXI side. The write buffering feature is used for the fixed size burst also in the same way as undefined length bursts.

Write buffer length can only be a power of two, and if it is not set to be a power of two then the number is floored to the closest power of two automatically. The maximum number has the same constraints as the read prefetch number. A synchronous memory width one read and write port is generated for write buffering. The size of the memory is determined by the write buffer length. The type of the memory can be configured with a generic also. The first AXI write transaction will not start until the buffer is filled or the AHB transaction has written the last beat in the burst. As a result there is a trade-off for performance while selecting the write buffer length which depends on the AXI slave behavior.

4.3.4 Endianness

The AHB side of AHB2AXIB can be either set to be little-endian or big-endian through "ahb_endianness" generic.

When AHB2AXIB is set to be little-endian then data is mapped directly to AXI since AXI is little-endian also.

When AHB2AXIB is set to be big-endian, the endianness on the AXI side is configurable through the *endianness_mode* generic.

When *endianness_mode* generic is set to zero a byte-invariant big-endian endianness mode is used on the AXI side. In order to translate big-endian AHB to byte-invariant big-endian AXI the byte order is reversed (see Fig. 7). No address translation occurs inside the adapter in this mode.

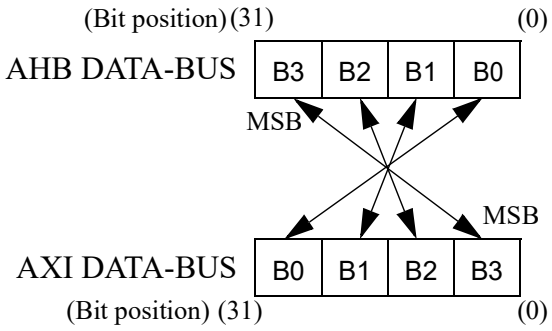


Figure 7. Big-endian AHB to byte-invariant Big-endian AXI translation (32-bit data-width)

When *endianness_mode* generic is set to one then big-endian AHB is translated to little-endian AXI. In order to achieve this the byte order is preserved but the address is translated from big-endian representation to little-endian representation when a narrow sized transaction is encountered (See Fig. 8 for an example with 32-bit data-bus width.).

The address translation formula for 32-bit, 64-bit, 128-bit and 256-bit data-bus widths are following:

32-bit data bus width:

if *HSIZE* < "010" :

axi_address(1:0) = ("100" - "1"<<"*HSIZE*" - ahb_address(1:0))(1:0)

otherwise:

axi_address(1:0) = ahb_address(1:0)

64-bit data bus width:

if *HSIZE* < "011" :

axi_address(2:0) = ("1000" - "1"<<"*HSIZE*" - ahb_address(2:0))(2:0)

otherwise:

axi_address(2:0) = ahb_address(2:0)

128-bit data bus width:

if *HSIZE* < “100” :

axi_address(3:0) = (“10000” - “1”<<“*HSIZE*” - ahb_address(3:0))(3:0)

otherwise:

axi_address(3:0) = ahb_address(3:0)

256-bit data bus width:

if *HSIZE* < “101” :

axi_address(4:0) = (“100000” - “1”<<“*HSIZE*” - ahb_address(4:0))(4:0)

otherwise:

axi_address(4:0) = ahb_address(4:0)

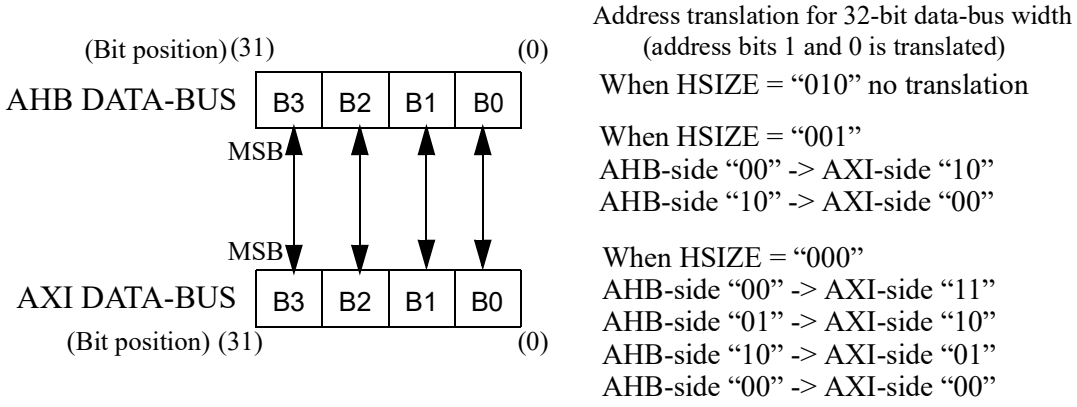


Figure 8. Big-endian AHB to little-endian AXI through address translation (32-bit data-width)

4.4 AXI AxPROT and AxCACHE Translations

The AxPROT and AxCACHE signals are translated partly according to the HPROT signal of AHB transactions. The full list of translation can be seen from Table 41.

Table 41. AxPROT and AxCACHE translations

AXI signal	Assignment
AxCACHE[3]	always logic ‘0’
AxCACHE[2]	always logic ‘0’
AxCACHE[1]	HPROT[3]
AxCACHE[0]	HPROT[2]
AxPROT[2]	not (HPROT[0])
AxPROT[1]	See configuration options (Table. 42)
AxPROT[0]	HPROT[1]

4.5 Implementation

4.5.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User’s Manual). All registers are reset.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

4.5.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User’s Manual). The core is designed for big-endian AHB systems while the AXI endianness can be configured with the *endianness_mode* generic.

4.6 Configuration Options

Table 42. Configuration options (both AHBM2AXI3 and AHBM2AXI4)

Generic	Function	Allowed range	Default
aximid	AXI master ID used for Read and Write transactions	0 - 15	0
always_secure	When set to 1 the AxPROT[1] bit is tied to logic '0' (always secure access), when set to 0 the AxPROT[1] bit is tied to logic '1' (always unsecure access).	0-1	1
ahb_endianness	Determines the endianness of the AHB side of the bridge. 0 -> Big-endian 1 -> Little-endian The default value is taken from the endianness generic set in grlib configuration. "grlib_little_endian"	0-1	GRLIB
endianness_mode	Determines the endianness mode (see section 4.3.4 for more detail) 0 -> Big-endian AHB to byte-invariant big-endian AXI 1 -> Big-endian AHB to little-endian AXI	0-1	0

Table 43. Configuration options specific for AXI3 (AHBM2AXI3)

Generic	Function	Allowed range	Default
wbuffer_num	Write-buffer length which determines the memory size also.	1-16	8
rprefetch_num	Read prefetch length.	1-16	8

Table 44. Configuration options specific for AXI4 (AHBM2AXI4)

Generic	Function	Allowed range	Default
wbuffer_num	Write-buffer length which determines the memory size also.	1-256 for data-width of 32-bit, 1-128 for data-width of 64-bit 1-64 for data-width of 128-bit 1-32 for data-width of 256-bit	8
rprefetch_num	Read prefetch length.	1-256 for data-width of 32-bit, 1-128 for data-width of 64-bit 1-64 for data-width of 128-bit 1-32 for data-width of 256-bit	8

4.7 Signal descriptions

Table 45 shows the interface signals of the core (VHDL ports).

Table 45. Signal descriptions (VHDL ports)

Signal name	Field	Type	Function	Active
RST		Input	Reset	Low
CLK		Input	AHB & AXI bus clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
AXIMI	*	Input	AXI3/4 master input signals	-
AXIMO	*	Output	AXI3/4 master output signals	-

* see GRLIB IP Library User's Manual

4.8 Library dependencies

Table 46 shows the libraries used when instantiating the core (VHDL libraries).

Table 46. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA & AXI signal definitions
GAISLER	AXI	Component	Component declaration

4.9 Instantiation

The instantiation of the AHBM2AXI adapter depends on the AXI protocol type. There are two components called AHBM2AXI3 which is built for AXI3 protocl and AHBM2AXI4 which is built for AXI4 protocol. The difference between these two components are the AXI master output signals and the maximum values that can be set for read prefetching and write buffering.

Since AHBM2AXI adapter is intended to be used for only a single core, a transaction is sampled and evaluated directly on the rising edge of the clock, the "*hsel*" and "*hready*" inputs are ignored by the AHBM2AXI adapter. The grant signal for the AHB master that is connected to the adapter should be hardwired to logic 1.

Following is an example in which a component with an ahb master interface called "*ahbm_ex*" is connected to the AHBM2AXI4 adapter which can act as an master for AXI4 protocol.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.axi.all;


entity ahbm2axi4_ex is
  port (
    rstn  : in  std_logic;
```

```
    clk   : in  std_logic;
    aximi : in  axi_somi_type;
    aximo : out axi4_mosi_type
  );
end;


architecture rtl of ahbm2axi4_ex is


signal  ahbsi : in  ahb_slave_in_type;
signal  ahbso : out ahb_slave_out_type;
signal  ahbmi : in  ahb_mst_in_type;
signal  ahbmo : out ahb_mst_out_type;

component ahbm_ex is
 port (
 signal rstn   : in std_logic;
 signal clk    : in std_logic;
 signal ahbmi  : in ahb_mst_in_type;
 signal ahbmo  : out ahb_mst_out_type);
end component;


begin

adapter:ahbm2axi4
generic map (
 memtech => 0,
 aximid => 0,
 wbuffer_num => 16,
 rprefetch_num=> 16,
 always_secure => 1
)
port map (
 rstn => rstn,
 clk => clk,
 ahbsi => ahbsi,
 ahbso => ahbso,
 aximi => aximi,
 aximo => aximo);

ahbmaster:ahbm_ex
port map (
 rstn => rstn,
 clk => clk,
 ahbmi => ahbmi,
 ahbmo => ahbmo);


ahbsi.haddr  <= ahbmo.hadddr;
ahbsi.hwrite <= ahbmo.hwrite;
ahbsi.htrans <= ahbmo.htrans;
ahbsi.hsize  <= ahbmo.hsize;
ahbsi.hburst <= ahbmo.hburst;
ahbsi.hwdata <= ahbmo.hwdata;
ahbsi.hprot  <= ahbmo.hprot;

ahbmi.hgrant <= (others=> '1');
ahbmi.hready <= ahbso.hready;
ahbmi.hresp <= ahbso.hresp;
ahbmi.hrdata <= ahbso.hrdata;
--Remaining ahb master inputs are implementation dependent
```

5 AHBBRIDGE - Bi-directional AHB/AHB bridge

5.1 Overview

A pair of uni-directional bridges (AHB2AHB) can be instantiated to form a bi-directional bridge. The bi-directional AHB/AHB bridge (AHBBRIDGE) instantiates two uni-directional bridges that are configured to suit the bus architecture shown in figure 9. The bus architecture consists of two AHB buses: a high-speed AHB bus hosting LEON3 CPU(s) and an external memory controller and a low-speed AHB bus hosting communication IP-cores.

Note: For other architectures, a more general bi-directional bridge that is more suitable can be created by instantiating two uni-directional AHB to AHB bridges (see AHB2AHB core). AHBBRIDGE is not suitable for LEON4 systems and for other systems with wide AHB buses.

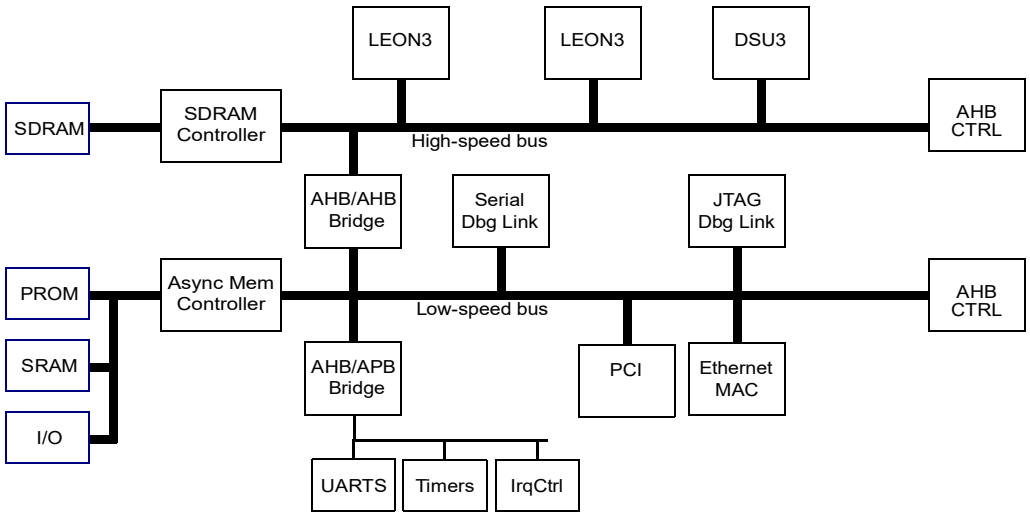


Figure 9. LEON3 system with a bi-directional AHB/AHB bridge

5.2 Operation

5.2.1 General

The AHB/AHB bridge is connected to each AHB bus through a pair consisting of an AHB master and an AHB slave interface. The address space occupied by the AHB/AHB bridge on each bus is determined by Bank Address Registers which are configured through VHDL generics. The bridge is capable of handling single and burst transfers in both directions. Internal FIFOs are used for data buffering. The bridge implements the AMBA SPLIT response to improve AHB bus utilization. For more information on AHB transfers please refer to the documentation for the uni-directional AHB/AHB bridge (AHB2AHB).

The requirements on the two bus clocks are that they are synchronous. The two uni-directional bridges forming the bi-directional AHB/AHB bridge are configured asymmetrically. Configuration of the bridge connecting high-speed bus with the low-speed bus (down bus) is optimized for the bus traffic generated by the LEON3 CPU since the CPU is the only master on the high-speed bus (except for the bridge itself). Read transfers generated by the CPU are single read transfers generated by single load instructions (LD), read bursts of length two generated by double load instructions (LDD) or incremental read bursts of maximal length equal to cache line size (4 or 8 words) generated during instruction cache line fill. The size of the read FIFO for the down bridge is therefore configurable to 4 or 8 entries which is the maximal read burst length. If a read burst is an instruction fetch (indicated on AHB HPROT signal) to a prefetchable area the bridge will prefetch data to the end of a instruction

cache line. If a read burst to a prefetchable area is a data access, two words will be prefetched (this transfer is generated by the LDD instruction). The write FIFO has two entries capable of buffering the longest write burst (generated by the STD instruction). The down bridge also performs interrupt forwarding, interrupt lines 1-15 on both buses are monitored and an interrupt on one bus is forwarded to the other one.

Since the low-speed bus does not host a LEON3 CPU, all AHB transfers forwarded by the uni-directional bridge connecting the low-speed bus and the high-speed bus (up bridge) are data transfers. Therefore the bridge does not make a distinction between instruction and data transfers. The size of the read and write FIFOs for this bridge is configurable and should be set by the user to suite burst transfers generated by the cores on the low-speed bus.

Note that the bridge has been optimized for a LEON3 system with a specific set of masters and a specific bus topology. Therefore the core may not be suitable for a design containing later versions of the LEON processor or other masters. In general it is not recommended instantiate the AHBBRIDGE core and instead instantiate two uni-directional AHB to AHB bridges (AHB2AHB cores) with configurations tailored for a specific design.

5.2.2 Deadlock conditions

A deadlock situation can occur if the bridge is simultaneously accessed from both buses. The bridge contains deadlock detection logic which will resolve a deadlock condition by giving a RETRY response on the low-speed bus.

There are several deadlock conditions that can occur with locked accesses. If the VHDL generic *lckdac* is 0, the bridge will deadlock if two simultaneous accesses from both buses are locked, or if a locked access is made while the bridge has issued a SPLIT response to a read access and the splitted access has not completed. If *lckdac* is greater than 0, the bridge will resolve the deadlock condition from two simultaneous locked accesses by giving an ERROR response on the low-speed bus. If *lckdac* is 1 and a locked access is made while the bridge has issued a SPLIT response to a read access, the bridge will respond with ERROR to the incoming locked access. If *lckdac* is 2 the bridge will allow both the locked access and the splitted read access to complete. Note that with *lckdac* set to 2 and two incoming locked accesses, the access on the low-speed bus will still receive an ERROR response.

5.2.3 Read and write combining

The bridge can be configured to support read and write combining so that prefetch operations and write bursts are always performed with the maximum access size possible on the master interface. Please see the documentation for the uni-directional AHB/AHB bridge (AHB2AHB) for a description of read and write combining and note that the same VHDL generics are used to specify both the maximum master and maximum slave access size on the bi-directional AHB/AHB bridge.

5.2.4 Endianness

The core is designed for big-endian systems

5.3 Registers

The core does not implement any registers.

5.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x020. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

5.5 Implementation

See documentation for AHB2AHB.

5.6 Configuration options

Table 47 shows the configuration options of the core (VHDL generics).

Table 47. Configuration options

Generic	Function	Allowed range	Default
memtech	Memory technology	-	0
ffact	Frequency ratio	1 -	2
hsb_hsindex	AHB slave index on the high-speed bus	0 to NAHBMAX-1	0
hsb_hmindex	AHB master index on the high-speed bus	0 to NAHBMAX-1	0
hsb_iclsize	Cache line size (in number of 32-bit words) for CPUs on the high-speed bus. Determines the number of the words that are prefetched by the bridge when CPU performs instruction bursts.	4, 8	8
hsb_bank0	Address area 0 mapped on the high-speed bus and decoded by the bridge's slave interface on the low-speed bus. Appears as memory address register (BAR0) on the bridge's low-speed bus slave interface. The generic has the same bit layout as bank address registers with bits [19:18] suppressed (use functions ahb2ahb_membar and ahb2ahb_iobar in gaisler.misc package to generate this generic).	0 - 1073741823	0
hsb_bank1	Address area 1 mapped on the high-speed bus	0 - 1073741823	0
hsb_bank2	Address area 2 mapped on the high-speed bus	0 - 1073741823	0
hsb_bank3	Address area 3 mapped on the high-speed bus	0 - 1073741823	0
hsb_ioarea	Address of high-speed bus I/O area that contains the high-speed bus configuration area. Will appear in the bridge's user-defined register 1 on the low-speed bus. Note that to allow low-speed bus masters to read the high-speed bus configuration area, the area must be mapped on one of the *hsb_bank* generics.	0 - 16#FFF#	0
lsb_hsindex	AHB slave index on the low-speed bus	0 to NAHBMAX-1	0
lsb_hmindex	AHB master index on the low-speed bus	0 to NAHBMAX-1	0
lsb_rburst	Size of the prefetch buffer for read transfers initiated on the low-speed-bus and crossing the bridge.	16, 32	16
lsb_wburst	Size of the write buffer for write transfers initiated on the low-speed bus and crossing the bridge.	16, 32	16
lsb_bank0	Address area 0 mapped on the low-speed bus and decoded by the bridge's slave interface on the high-speed bus. Appears as memory address register (BAR0) on the bridge's high-speed bus slave interface. The generic has the same bit layout as bank address registers with bits [19:18] suppressed (use functions ahb2ahb_membar and ahb2ahb_iobar in gaisler.misc package to generate this generic).	0 - 1073741823	0
lsb_bank1	Address area 1 mapped on the low-speed bus	0 - 1073741823	0
lsb_bank2	Address area 2 mapped on the low-speed bus	0 - 1073741823	0
lsb_bank3	Address area 3 mapped on the low-speed bus	0 - 1073741823	0

Table 47. Configuration options

Generic	Function	Allowed range	Default
lsb_ioarea	Address of low-speed bus I/O area that contains the low-speed bus configuration area. Will appear in the bridge's user-defined register 1 on the high-speed bus. Note that to allow high-speed bus masters to read the low-speed bus configuration area, the area must be mapped on one of the *lsb_bank* generics.	0 - 16#FFF#	0
lckdac	Locked access error detection and correction. This generic is mapped to the generic with the same name on the two AHB2AHB cores instantiated by AHBBRIDGE. Please see the documentation for the AHB2AHB core's VHDL generics for more information.	0 - 2	0
maccsz	This generic is propagated to the slvmaccsz and mstmaccsz VHDL generics on the two AHB2AHB cores instantiated by AHBBRIDGE. The generic determines the maximum AHB access size supported by the bridge. Please see the documentation for the AHB2AHB core's VHDL generics for more information.	32 - 256	32
rdcomb	Read combining, this generic is mapped to the generic with the same name on the two AHB2AHB cores instantiated by AHBBRIDGE. Please see the documentation for the AHB2AHB core's VHDL generics for more information.	0 - 2	0
wrcomb	Write combining, this generic is mapped to the generic with the same name on the two AHB2AHB cores instantiated by AHBBRIDGE. Please see the documentation for the AHB2AHB core's VHDL generics for more information.	0 - 2	0
combmask	Read/Write combining mask, this generic is mapped to the generic with the same name on the two AHB2AHB cores instantiated by AHBBRIDGE. Please see the documentation for the AHB2AHB core's VHDL generics for more information.	0 - 16#FFFF#	16#FFFF#
allbrst	Support all burst types, this generic is mapped to the generic with the same name on the two AHB2AHB cores instantiated by AHBBRIDGE. Please see the documentation for the AHB2AHB core's VHDL generics for more information.	0 - 2	0
fcfs	First-come, first-served operation, this generic is mapped to the generic with the same name on the two AHB2AHB cores instantiated by AHBBRIDGE. Please see the documentation for the AHB2AHB core's VHDL generics for more information.	0 - NAHBMST	0
scantest	Enable scan support	0 - 1	0

5.7 Signal descriptions

Table 48 shows the interface signals of the core (VHDL ports).

Table 48. Signal descriptions

Signal name	Type	Function	Active
RST	Input	Reset	Low
HSB_HCLK	Input	High-speed AHB clock	-
LSB_HCLK	Input	Low-speed AHB clock	-
HSB_AHBSI	Input	High-speed bus AHB slave input signals	-
HSB_AHBSO	Output	High-speed bus AHB slave output signals	-
HSB_AHBSOV	Input	High-speed bus AHB slave input signals	-
HSB_AHBMI	Input	High-speed bus AHB master input signals	-
HSB_AHBMO	Output	High-speed bus AHB master output signals	-
LSB_AHBSI	Input	Low-speed bus AHB slave input signals	-
LSB_AHBSO	Output	Low-speed bus AHB slave output signals	-
LSB_AHBSOV	Input	Low-speed bus AHB slave input signals	-
LSB_AHBMI	Input	Low-speed bus AHB master input signals	-
LSB_AHBMO	Output	Low-speed bus AHB master output signals	-

5.8 Library dependencies

Table 49 shows the libraries used when instantiating the core (VHDL libraries).

Table 49. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

6 AHBCTRL - AMBA AHB controller with plug&play support

6.1 Overview

The AMBA AHB controller is a combined AHB arbiter, bus multiplexer and slave decoder according to the AMBA 2.0 standard.

The controller supports up to 16 AHB masters, and 16 AHB slaves. The maximum number of masters and slaves are defined in the GRLIB.AMBA package, in the VHDL constants NAHBSLV and NAHBMST. It can also be set with the *nahbm* and *nahbs* VHDL generics.

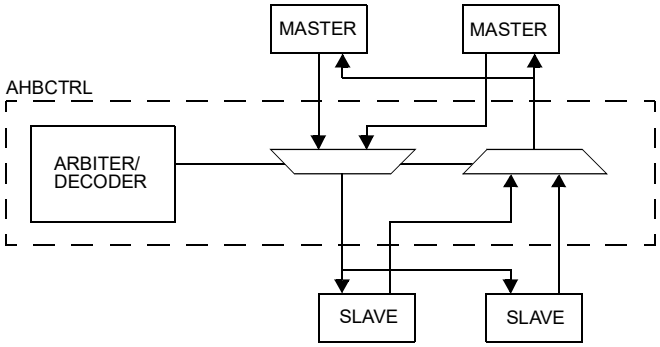


Figure 10. AHB controller block diagram

6.2 Operation

6.2.1 Arbitration

The AHB controller supports two arbitration algorithms: fixed-priority and round-robin. The selection is done by the VHDL generic *rrobin*. In fixed-priority mode (*rrobin* = 0), the bus request priority is equal to the master's bus index, with index 0 being the lowest priority. If no master requests the bus, the master with bus index 0 (set by the VHDL generic *defmast)* will be granted.

In round-robin mode, priority is rotated one step after each AHB transfer. If no master requests the bus, the last owner will be granted (bus parking). The VHDL generic *mprio* can be used to specify one or more masters that should be prioritized when the core is configured for round-robin mode.

Note that there are AHB slaves that implement split-like functionality by giving AHB retry responses until the access has finished and the original master tries again. All masters on the bus accessing such slaves must be round-robin arbitrated without prioritization to avoid deadlock situations. For GRLIB this applies to the GRPCI and GRPCI2 cores.

During incremental bursts, the AHB master should keep the bus request asserted until the last access as recommended in the AMBA 2.0 specification, or it might loose bus ownership. For fixed-length burst, the AHB master will be granted the bus during the full burst, and can release the bus request immediately after the first access has started. For this to work however, the VHDL generic *fixbrst* should be set to 1.

6.2.2 Decoding

Decoding (generation of HSEL) of AHB slaves is done using the plug&play method explained in the GRLIB User's Manual. A slave can occupy any binary aligned address space with a size of 1 - 4096 Mbyte. A specific I/O area is also decoded, where slaves can occupy 256 byte - 1 Mbyte. The default address of the I/O area is 0xFFF00000, but can be changed with the *ioaddr* and *iomask* VHDL generics. Access to unused addresses will cause an AHB error response.

The I/O area can be placed within a memory area occupied by a slave. The slave will not be selected when the I/O area is accessed.

6.2.3 Plug&play information

GRLIB devices contain a number of plug&play information words which are included in the AHB records they drive on the bus (see the GRLIB user's manual for more information). These records are combined into an array which is connected to the AHB controller unit.

The plug&play information is mapped on a read-only address area, defined by the *cfgaddr* and *cfgmask* VHDL generics, in combination with the *ioaddr* and *iomask* VHDL generics. By default, the area is mapped on address 0xFFFFF000 - 0xFFFFFFFF. The master information is placed on the first 2 kbyte of the block (0xFFFFF000 - 0xFFFFF800), while the slave information is placed on the second 2 kbyte block. Each unit occupies 32 bytes, which means that the area has place for 64 masters and 64 slaves. The address of the plug&play information for a certain unit is defined by its bus index. The address for masters is thus 0xFFFFF000 + *n**32, and 0xFFFFF800 + *n**32 for slaves.

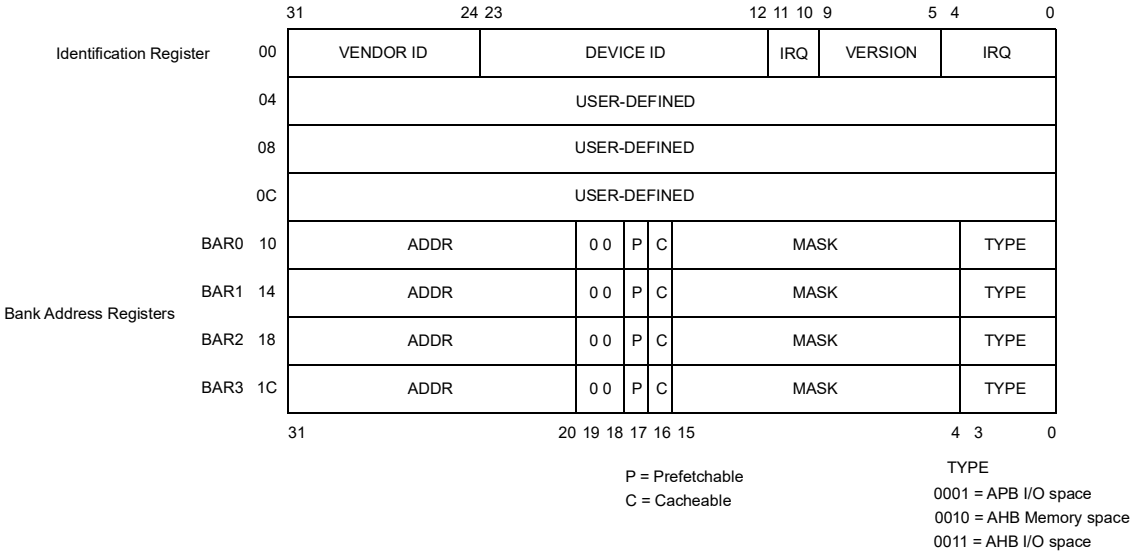


Figure 11. AHB plug&play information record

6.3 AHB split support

AHB SPLIT functionality is supported if the *split* VHDL generic is set to 1. In this case, all slaves must drive the AHB SPLIT signal.

It is important to implement the split functionality in slaves carefully since locked splits can otherwise easily lead to deadlocks. A locked access to a slave which is currently processing (it has returned a split response but not yet split complete) an access which it returned split for to another master must be handled first. This means that the slave must either be able to return an OKAY response to the locked access immediately or it has to split it but return split complete to the master performing the locked transfer before it has finished the first access which received split.

6.4 Locked accesses

The GRLIB AHB controller treats HLOCK as coupled to a specific access. If a previous access by a master received a SPLIT/RETRY response then the arbiter will disregard the current value of HLOCK. This is done as opposed to always treating HLOCK as being valid for the next access which can result in a previously non-locked access being treated as locked when it is retried. Consider the following sequence:

T0: MSTx write 0
T1: MSTx write 1, HLOCK asserted as next access performed by master will be locked
T2: MSTx locked read

If (the non-locked) write 0 access at T0 receives a RETRY or SPLIT response (given at time T1), then the next access to be performed may be a retry of write 0. In this case the arbiter will disregard the HLOCK setting and the retried access will not have HMASTLOCK set.

6.5 AHB bus monitor

An AHB bus monitor is integrated into the core. It is enabled with the *enbusmon* generic. It has the same functionality as the AHB and arbiter parts in the AMBA monitor core (AMBAMON). For more information on which rules are checked se the AMBAMON documentation.

6.6 Registers

The core does not implement any registers.

6.7 Implementation

6.7.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers if the GRLIB config package setting *grlib_async_reset_enable* is set.

6.7.2 Endianness

The core supports both big-endian and little-endian systems. The endianness of the AHB bus controlled by the entity is selected through the generic *ahbendian*. The *ahbendian* VHDL generic is by default set to the value of the constant in *grlib_little_endian* the GRLIB configuration package (see GRLIB User's Manual).

6.8 Configuration options

Table 50 shows the configuration options of the core (VHDL generics).

Table 50. Configuration options

Generic	Function	Allowed range	Default
ioaddr	The MSB address of the I/O area. Sets the 12 most significant bits in the 32-bit AHB address (i.e. 31 downto 20)	0 - 16#FFF#	16#FFF#
iomask	The I/O area address mask. Sets the size of the I/O area and the start address together with ioaddr.	0 - 16#FFF#	16#FFF#
cfgaddr	The MSB address of the configuration area. Sets 12 bits in the 32-bit AHB address (i.e. 19 downto 8).	0 - 16#FFF#	16#FF0#
cfgmask	The address mask of the configuration area. Sets the size of the configuration area and the start address together with cfgaddr. If set to 0, the configuration will be disabled.	0 - 16#FFF#	16#FF0#
rrobin	Selects between round-robin (1) or fixed-priority (0) bus arbitration algorithm.	0 - 1	0

Table 50. Configuration options

Generic	Function	Allowed range	Default
split	Enable support for AHB SPLIT response	0 - 1	0
defmast	Default AHB master	0 - NAHBMST-1	0
ioen	AHB I/O area enable. Set to 0 to disable the I/O area	0 - 1	1
disirq	Set to 1 to disable interrupt routing	0 - 1	0
nahbm	Number of AHB masters	1 - NAHBMST	NAHBMST
nahbs	Number of AHB slaves	1 - NAHBSLV	NAHBSLV
timeout	Perform bus timeout checks (NOT IMPLEMENTED).	0 - 1	0
fixbrst	Enable support for fixed-length bursts	0 - 1	0
debug	Print configuration (0=none, 1=short, 2=all cores)	0 - 2	2
fpnpen	Enables full decoding of the PnP configuration records. When disabled the user-defined registers in the PnP configuration records are not mapped in the configuration area.	0 - 1	0
icheck	Check bus index	0 - 1	1
devid	Assign unique device identifier readable from plug and play area.	N/A	0
enbusmon	Enable AHB bus monitor	0 - 1	0
assertwarn	Enable assertions for AMBA recommendations. Violations are asserted with severity warning.	0 - 1	0
asserterr	Enable assertions for AMBA requirements. Violations are asserted with severity error.	0 - 1	0
hmstdisable	Disable AHB master rule check. To disable a master rule check a value is assigned so that the binary representation contains a one at the position corresponding to the rule number, e.g 0x80 disables rule 7.	N/A	0
hslvdisable	Disable AHB slave tests. Values are assigned as for hmstdisable.	N/A	0
arbdisable	Disable Arbiter tests. Values are assigned as for hmstdisable.	N/A	0
mprio	Master(s) with highest priority. This value is converted to a vector where each position corresponds to a master. To prioritize masters x and y set this generic to $2^x + 2^y$.	N/A	0
mcheck	Check if there are any intersections between core memory areas. If two areas intersect an assert with level failure will be triggered (in simulation). mcheck = 1 does not report intersects between AHB IO areas and AHB memory areas (as IO areas are allowed to override memory areas). mcheck = 2 triggers on all overlaps. See also documentation of VHDL generic shadow below.	0 - 2	1
ccheck	Perform sanity checks on PnP configuration records (in simulation).	0 - 1	1

Table 50. Configuration options

Generic	Function	Allowed range	Default
acdm	AMBA compliant data multiplexing (for HSIZE > word). If this generic is set to 1, and the AMBA bus data width in the system exceeds 32-bits, the core will ensure AMBA compliant data multiplexing for access sizes (HSIZE) over 32-bits. GRLIB cores have an optimization where they drive the same data on all lanes. Read data is always taken from the lowest lanes. If an AMBA compliant core from another vendor is introduced in the design, that core may not always place valid data on the low part of the bus. By setting this generic to 1, the AHBCTRL core will replicate the data, allowing the non-GRLIB cores to be instantiated without modification. Refer to [GRLIB] for more information.	0 - 1	0
index	AHB bus index optionally for trace print-out. See the *ahbtrace* generic.		0
ahbtrace	Enables AHB trace print-out to simulator console when non-zero. Some customization of prints is possible depending on the value: *ahbtrace* = 1 + 2***printindex* + 4***printfullword* *printindex*: If set to 1 then each trace line is prefixed with the bus number taken from the *index* generic. *printfullword*: If set to 1 then all bits of HWDATA/HRDATA are included in the trace. Otherwise only the subword selected by HSIZE and HADDR is printed. This option is can be useful to debug endianness and data muxing on wide buses (see the *acdm* generic).	0 - 7	0
hwdebug	Enable hardware debug registers. If this generic is set to 1 the configuration area will include to diagnostic registers at offsets 0xFF4 and 0xFF8. Offset 0xFF4 will show a 32-bit register where bit *n* shows the current status of AHB master *n*'s HBUSREQ signal. Offset 0xFF8 will show a 32-bit register where bit *n* shows the current SPLIT status of AHB master *n*. The bit will be set when AHB master *n* receives a SPLIT reply and will be re-set to '0' when HSPLIT for AHB master *n* has been asserted. This functionality is not intended to be used in production systems but can provide valuable information while debugging systems with cores that have problems with AMBA SPLIT replies.	0 - 1	0
fourgslave	Allow and optimize for case with one single slave that has one 4 GiB bar	0 - 1	0

Table 50. Configuration options

Generic	Function	Allowed range	Default
shadow	Allow memory areas to shadow other memory areas. If this generic is set to 0 and two slaves map the same memory area then HSEL/HMBSEL signals will be asserted for both memory bars / slaves. This may lead to system malfunctions and causes a simulation failure if the mcheck VHDL generic is set to a non-zero value. If the shadow generic is set to 1 then memory area intersects are allowed and only the lowest HSEL and HMBSEL (HSEL has priority) will be asserted - only the slave or bar with the lowest index will be selected instead of both slaves / bars. The mcheck simulation failure will instead be asserted as a note about intersecting memory areas. Also note that intersections of cacheable and noncacheable areas will be treated as cacheable by GRLB cores that decode the plug&play information. If a non-cacheable area is placed in a cacheable area then it is recommended to use fixed cacheability.	0 - 1	0
unmapslv	If this generic is non-zero then accesses to unmapped address space (address space not occupied by any slave) will be redirected to the slave and bar selected via: 256+bar*32+slv.		0
ahbendian	Selects the endianness of the AHB bus controlled by the entity. 0: Big-endian 1: Little-endian The default value is taken from the endianness generic set in GRLIB configuration package "*grlib_little_endian*" (see GRLIB User's Manual).	0-1	GRLIB
dbgtag	If this generic is a non-empty string, it will be prefixed to each AHB trace line printed to the simulator console (see *ahbtrace* generic).	N/A	Empty string

6.9 Signal descriptions

Table 51 shows the interface signals of the core (VHDL ports).

Table 51. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AHB reset	Low
CLK	N/A	Input	AHB clock	-
MSTI	*	Output	AMBA AHB master interface record array	-
MSTO	*	Input	AMBA AHB master interface record array	-
SLVI	*	Output	AMBA AHB slave interface record array	-
SLVO	*	Input	AMBA AHB slave interface record array	-

* see GRLIB IP Library User's Manual

6.10 Library dependencies

Table 52 shows libraries used when instantiating the core (VHDL libraries).

Table 52. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions

6.11 Component declaration

```
library grlib;
use grlib.amba.all;

component ahbctrl
  generic (
    defmast : integer := 0; -- default master
    split   : integer := 0; -- split support
    rrobin  : integer := 0; -- round-robin arbitration
    timeout : integer range 0 to 255 := 0; -- HREADY timeout
    ioaddr  : ahb_addr_type := 16#fff#; -- I/O area MSB address
    iomask  : ahb_addr_type := 16#fff#; -- I/O area address mask
    cfgaddr : ahb_addr_type := 16#ff0#; -- config area MSB address
    cfgmask : ahb_addr_type := 16#ff0#; -- config area address maskk
    nahbm   : integer range 1 to NAHBMST := NAHBMST; -- number of masters
    nahbs   : integer range 1 to NAHBSLV := NAHBSLV; -- number of slaves
    ioen    : integer range 0 to 15 := 1; -- enable I/O area
    disirq  : integer range 0 to 1 := 0; -- disable interrupt routing
    fixbrst : integer range 0 to 1 := 0; -- support fix-length bursts
    debug   : integer range 0 to 2 := 2; -- print configuration to consolee
    fpnpen  : integer range 0 to 1 := 0; -- full PnP configuration decoding
    icheck  : integer range 0 to 1 := 1;
    devid   : integer := 0; -- unique device ID
    enbusmon    : integer range 0 to 1 := 0; -- enable bus monitor
    assertwarn  : integer range 0 to 1 := 0; -- enable assertions for warnings
    asserterr   : integer range 0 to 1 := 0; -- enable assertions for errors
    hmstdisable : integer := 0; -- disable master checks
    hslvdisable : integer := 0; -- disable slave checks
    arbdisable  : integer := 0; -- disable arbiter checks
    mprio       : integer := 0; -- master with highest priority
    enbusmon    : integer range 0 to 1 := 0; -- enable bus monitor
    assertwarn  : integer range 0 to 1 := 0; -- enable assertions for warnings
    asserterr   : integer range 0 to 1 := 0; -- enable assertions for errors
    hmstdisable : integer := 0; -- disable master checks
    hslvdisable : integer := 0; -- disable slave checks
    arbdisable  : integer := 0; -- disable arbiter checks
    mprio       : integer := 0; -- master with highest priority
    mcheck      : integer range 0 to 2 := 1;  -- check memory map for intersects
    ccheck      : integer range 0 to 1 := 1;  -- perform sanity checks on pnp config
    acdm        : integer := 0; -- AMBA compliant data muxing (for hsize > word)
    index       : integer := 0; -- index for trace print-out
    ahbtrace    : integer := 0; -- AHB trace enable
    hwdebug     : integer := 0;
    fourgslv    : integer := 0;
    shadow      : integer range 0 to 1 := 0; -- Allow overlapping memory areas
    unmapslv    : integer := 0;
    ahbendian   : integer := GRLIB_ENDIAN
-- pragma translate_off
    ;
    dbgtag      : string := "" -- Prefix for AHB trace printout
-- pragma translate_on
  );
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    msti    : out ahb_mst_in_type;
    msto    : in  ahb_mst_out_vector;
    slvi    : out ahb_slv_in_type;
```

```
    slvo    : in  ahb_slv_out_vector;
    testen  : in  std_ulogic := '0';
    testrst : in  std_ulogic := '1';
    scanen  : in  std_ulogic := '0';
    testoen : in  std_ulogic := '1'
  );
  end component;
```

6.12 Instantiation

This example shows the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;

.
.

  -- AMBA signals
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

begin

-- ARBITER

ahb0 : ahbctrl -- AHB arbiter/multiplexer
  generic map (defmast => CFG_DEFMST, split => CFG_SPLIT,
rrobin => CFG_RROBIN, ioaddr => CFG_AHBIO, nahbm => 8, nahbs => 8)
  port map (rstn, clkm, ahbmi, ahbmo, ahbsi, ahbso);


-- AHB slave

sr0 : srctrl generic map (hindex => 3)
port map (rstn, clkm, ahbsi, ahbso(3), memi, memo, sdo3);

-- AHB master

e1 : eth_oc
      generic map (mstndx => 2, slvndx => 5, ioaddr => CFG_ETHIO, irq => 12, memtech =>
memtech)
      port map (rstn, clkm, ahbsi, ahbso(5), ahbmi  => ahbmi,
ahbmo => ahbmo(2), ethi1, etho1);
...
end;
```

6.13 Debug print-out

If the debug generic is set to 2, the plug&play information of all attached AHB units are printed to the console during the start of simulation. Reporting starts by scanning the master interface array from 0 to NAHBMST - 1 (defined in the grlib.amba package). It checks each entry in the array for a valid vendor-id (all nonzero ids are considered valid) and if one is found, it also retrieves the device-id. The descriptions for these ids are obtained from the GRLIB.DEVICES package, and are then printed on standard out together with the master number. If the index check is enabled (done with a VHDL generic), the report module also checks if the hindex number returned in the record matches the array number of the record currently checked (the array index). If they do not match, the simulation is aborted and an error message is printed.

This procedure is repeated for slave interfaces found in the slave interface array. It is scanned from 0 to NAHBSLV - 1 and the same information is printed and the same checks are done as for the master interfaces. In addition, the address range and memory type is checked and printed. The address information includes type, address, mask, cacheable and pre-fetchable fields. From this information, the report module calculates the start address of the device and the size of the range. The information finally printed is type, start address, size, cacheability and pre-fetchability. The address ranges currently defined are AHB memory, AHB I/O and APB I/O. APB I/O ranges are ignored by this module.

```
# vsim -c -quiet leon3mp
VSIM 1> run
# LEON3 MP Demonstration design
# GRLIB Version 1.0.7
# Target technology: inferred,  memory library: inferred
# ahbctrl: AHB arbiter/multiplexer rev 1
# ahbctrl: Common I/O area disabled
# ahbctrl: Configuration area at 0xfffff000, 4 kbyte
# ahbctrl: mst0: Frontgrade Gaisler        Leon3 SPARC V8 Processor
# ahbctrl: mst1: Frontgrade Gaisler        AHB Debug UART
# ahbctrl: slv0: European Space Agency   Leon2 Memory Controller
# ahbctrl:       memory at 0x00000000, size 512 Mbyte, cacheable, prefetch
# ahbctrl:       memory at 0x20000000, size 512 Mbyte
# ahbctrl:       memory at 0x40000000, size 1024 Mbyte, cacheable, prefetch
# ahbctrl: slv1: Frontgrade Gaisler        AHB/APB Bridge
# ahbctrl:       memory at 0x80000000, size 1 Mbyte
# apbctrl: APB Bridge at 0x80000000 rev 1
# apbctrl: slv0: European Space Agency   Leon2 Memory Controller
# apbctrl:       I/O ports at 0x80000000, size 256 byte
# apbctrl: slv1: Frontgrade Gaisler        Generic UART
# apbctrl:       I/O ports at 0x80000100, size 256 byte
# apbctrl: slv2: Frontgrade Gaisler        Multi-processor Interrupt Ctrl.
# apbctrl:       I/O ports at 0x80000200, size 256 byte
# apbctrl: slv3: Frontgrade Gaisler        Modular Timer Unit
# apbctrl:       I/O ports at 0x80000300, size 256 byte
# apbctrl: slv7: Frontgrade Gaisler        AHB Debug UART
# apbctrl:       I/O ports at 0x80000700, size 256 byte
# apbctrl: slv11: Frontgrade Gaisler        General Purpose I/O port
# apbctrl:       I/O ports at 0x80000b00, size 256 byte
# grgpio11: 8-bit GPIO Unit rev 0
# gptimer3: GR Timer Unit rev 0, 8-bit scaler, 2 32-bit timers, irq 8
# irqmp: Multi-processor Interrupt Controller rev 3, #cpu 1
# apbuart1: Generic UART rev 1, fifo 4, irq 2
# ahbuart7: AHB Debug UART rev 0
# leon3_0: LEON3 SPARC V8 processor rev 0
# leon3_0: icache 1*8 kbyte, dcache 1*8 kbyte
VSIM 2>
```

7 AHBJTAG - JTAG Debug Link with AHB Master Interface

7.1 Overview

The JTAG debug interface provides access to on-chip AMBA AHB bus through JTAG. The JTAG debug interface implements a simple protocol which translates JTAG instructions to AHB transfers. Through this link, a read or write transfer can be generated to any address on the AHB bus.

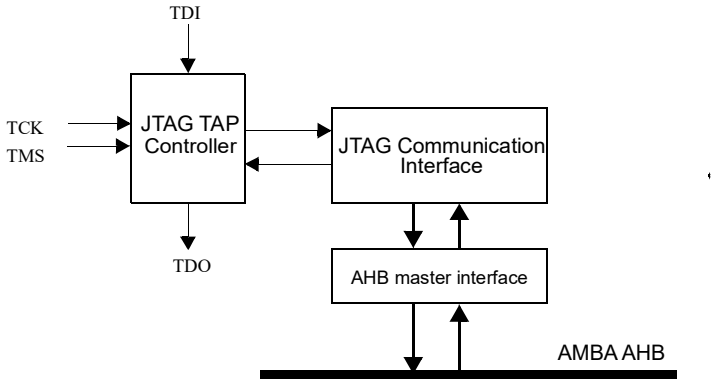


Figure 12. JTAG Debug link block diagram.

7.2 Operation

7.2.1 Transmission protocol

The JTAG Debug link decodes two JTAG instructions and implements two JTAG data registers: a 35-bit Command/Address register (ADATA) and a 33-bit Data register (DDATA), see tables 53 and 54 respectively.

An AHB read access is performed by shifting a command with the read/write bit set to ‘0’, and the desired AHB access size and AHB address fields into the ADATA register. When the AHB read access completes, the AHB response data is latched and can be retrieved by selecting the DDATA register and shifting out its contents. An AHB write access is performed by first shifting a command (with the read/write bit set to ‘1’) into the ADATA register and later shifting the desired write data into the DDATA register.

For both reads and writes, accesses are nominally initiated when the TAP enters the Update-DR state. However, a few extra TCK cycles may be needed before this information reaches the AMBA clock domain.

The protocol also provides a means to perform *sequential transfers* where the ADATA register only needs to be accessed to set the address of the first access in a sequence. A sequential read access is made by shifting in a ‘1’ into the SEQ bit of the DDATA register when shifting out the result of the previous read access. This causes the ADATA address field to be incremented by 4 and an AHB read access to the new address to be initiated. Similarly, if a ‘1’ is shifted into the SEQ when shifting in data for an AHB write access into the DDATA register, then the address field in ADATA is incremented by 4 for the subsequent access. Sequential transfers should not cross a 1 kB boundary and are always word based.

Table 53. JTAG debug link Command/Address register (ADATA)

34	33 32	31 … 0
W	SIZE	AHB ADDRESS

34	Write (W) - ‘0’ - read transfer, ‘1’ - write transfer
33: 32	AHB transfer size - “00” - byte, “01” - half-word, “10” - word, “11”- reserved
31: 30	AHB address

Table 54. JTAG debug link Data register (DDATA)

32	31 … 0
SEQ	AHB DATA

32	Sequential transfer (SEQ) - If '1' is shifted in this bit position when read data is shifted out or write data shifted in, the subsequent transfer will be to next word address. When read out from the device, this bit is '1' if the AHB access has completed and '0' otherwise.
31: 30	AHB Data - AHB write/read data. For byte and half-word transfers data is aligned according to big-endian order where data with address offset 0 data is placed in MSB bits.

In previous versions the debug host could not determine if an AHB accesses had finished when the read data was shifted out of the JTAG debug link data register. But as of version 1 a debug host can look at bit 32 of the received data to determine if the access was successful. If bit 32 is '1' the access completed and the data is valid. If bit 32 is '0', the AHB access was not finished when the host started to read data. In this case the host can repeat the read of the data register until bit 32 is set to '1', signaling that the data is valid and that the AMBA AHB access has completed. However, it is not possible to distinguish whether the the AHB access completed with an AHB ERROR or OKAY response.

It should be noted that while bit 32 returns '0', new data will not be latched into the data register. The debug host should therefore inspect bit 32 when shifting in data for a sequential AHB access to see if the previous command has completed. If bit 32 is '0', the read data is not valid and the command just shifted in has been dropped by the core.

Inspection of bit 32 should not be done for JTAG Debug links with version number 0.

7.2.2 Endianness

The core is designed for big-endian systems.

7.3 Registers

The core does not implement any registers mapped in the AMBA AHB or APB address space.

7.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x01C. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

7.5 Implementation

7.5.1 Clocking

Except for the TAP state machine and instruction register, the JTAG debug link operates in the AMBA clock domain. To detect when to shift the address/data register, the JTAG clock and TDI are resynchronized to the AMBA domain. The JTAG clock must be less than 1/3 of the AHB clock frequency for the debug link commands to work when nsync=2, and less than 1/2 of the AHB frequency when nsync=1.

7.5.2 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). Registers in the JTAG clock domain have asynchronous reset connected to the JTAG TRST. Registers in the system clock domain have synchronous reset.

The core will add reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support the GRLIB config package setting *grlib_async_reset_enable*.

7.5.3 Endianness

The core is designed for big-endian systems and does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). Nevertheless, the core is compatible with a 32-bit AHB of any endianness because it provides direct access to the 32-bit HRDATA/HWDATA value regardless of the access size. Therefore the core is endianness-agnostic when it comes to subword accesses.

On AHB buses wider than 32 bits, the core replicates its 32-bit data across the width of the bus. This allows it to be compatible to some little-endian systems as long as all slaves also replicate data during narrow read accesses (i.e. accesses narrower then the bus width).

7.5.4 Technology mapping

When the generic technology is selected (*tech* VHDL generic set to 0) a technology agnostic customizable TAP is instantiated. It uses a 6-bit instruction register, and implements the BYPASS and IDCODE instructions in addition to the instructions for selecting the ADATA and DDATA registers.

When an FPGA technology with a supported built-in TAP is selected, the core instead instantiates a technology specific macro to access signals from the native TAP. In this case all properties of the TAP are device dependent. This includes instruction register length, IDCODE and in some cases the instruction codes that select the ADATA and DDATA registers. The GRLIB techmap layer currently includes native TAPs for Xilinx, Actel/Microsemi and Altera devices. For a complete list of supported technologies, see the *has_tap* constant in the *gencomp* package in the *techmap* library. Consult device specific documentation for details.

7.6 Configuration options

Table 55 shows the configuration options of the core (VHDL generics).

Table 55. Configuration options

Generic	Function	Allowed range	Default
tech	Target technology	0 - NTECH	0
hindex	AHB master index	0 - NAHBMST-1	0
nsync	Number of synchronization registers between clock regions	1 - 2	1
*idcode	JTAG instruction code to select IDCODE register.	0 - 255	9
*manf	Manufacturer id. Appears as bits 11-1 in TAP controller's IDCODE register. Default is Frontgrade Gaisler manufacturer ID.	0 - 2047	804
*part	Part number. Sets bits 27-12 in IDCODE register.	0 - 65535	0
*ver	Version number. Sets bits 31-28 in IDCODE register .	0 - 15	0
**ainst	JTAG instruction code to select JTAG debug link command/address (ADATA) register. Should be set to 16 for Actel/Microsemi technologies. The default value (2) can be used for all other technologies.	0 - 255	2
**dinst	JTAG instruction code to select JTAG debug link data (DDATA) register. Should be set to 17 for Actel/Microsemi TAPs. The default value (3) can be used for all other technologies.	0 - 255	3
scantest	Enable scan test support	0 - 1	0
oepol	Output enable polarity for TDOEN	0 - 1	1
*tcknen	Support externally inverted TCK.	0 - 1	0
taptecharg	Passed to TAP techmap. Used in BSCANE2_TAP to select between standard user scan chain (0: USER1/USER2, 1: USER3/USER4).		0

*) Only used with generic tech (*tech* VHDL generic set to 0).

**) Tha *ainst* and *dinst* generics are ignored in Xilinx TAPs (*tech* VHDL generic set to a Xilinx technology).

7.7 Signal descriptions

Table 56 shows the interface signals of the core (VHDL ports).

Table 56. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	System clock (AHB clock domain)	-
TCK	N/A	Input	JTAG clock*	-
TMS	N/A	Input	JTAG TMS signal*	High
TDI	N/A	Input	JTAG TDI signal*	High
TDO	N/A	Output	JTAG TDO signal*	High
AHBI	***	Input	AHB Master interface input	-
AHBO	***	Output	AHB Master interface output	-
TAPO_TCK	N/A	Output	TAP Controller User interface TCK signal**	High
TAPO_TDI	N/A	Output	TAP Controller User interface TDI signal**	High
TAPO_INST[7:0]	N/A	Output	TAP Controller User interface INSTsignal**	High
TAPO_RST	N/A	Output	TAP Controller User interface RST signal**	High
TAPO_CAPT	N/A	Output	TAP Controller User interface CAPT signal**	High
TAPO_SHFT	N/A	Output	TAP Controller User interface SHFT signal**	High
TAPO_UPD	N/A	Output	TAP Controller User interface UPD signal**	High
TAPI_TDO	N/A	Input	TAP Controller User interface TDO signal**	High
TRST	N/A	Input	JTAG TRST signal	Low
TDOEN	N/A	Output	Output-enable for TDO	See oepol
TCKN	N/A	Input	Inverted JTAG clock* (if tcknen is set)	-
TAPO_TCKN	N/A	Output	TAP Controller User interface TCKN signal**	High
TAPO_NINST	N/A	Output	TAP Controller User interface NINSTsignal**	High
TAPO_IUPD	N/A	Output	TAP Controller User interface IUPD signal**	High

*) If the target technology is Xilinx or Altera the cores JTAG signals TCK, TCKN, TMS, TDI and TDO are not used. Instead the dedicated FPGA JTAG pins are used. These pins are implicitly made visible to the core through TAP controller instantiation.

**) User interface signals from the JTAG TAP controller. These signals are used to interface additional user defined JTAG data registers such as boundary-scan register. For more information on the JTAG TAP controller user interface see JTAG TAP Controller IP-core documentation. If not used tie TAPI_TDO to ground and leave TAPO_* outputs unconnected.

***) see GRLIB IP Library User's Manual

7.8 Signal definitions and reset values

The signals and their reset values are described in table 57.

Table 57. Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
dsutck	Input	JTAG clock	-	-
dsutms	Input	JTAG TMS	High	-
dsutdi	Input	JTAG TDI	High	-
dsutdo	Output	JTAG TDO	High	undefined

7.9 Timing

The timing waveforms and timing parameters are shown in figure 13 and are defined in table 58.

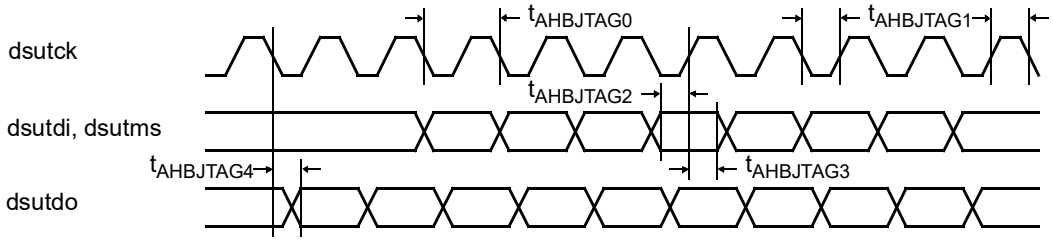


Figure 13. Timing waveforms

Table 58. Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{AHBJTAG0}$	clock period	-	TBD	-	ns
$t_{AHBJTAG1}$	clock low/high period	-	TBD	-	ns
$t_{AHBJTAG2}$	data input to clock setup	rising *dsutck* edge	TBD	-	ns
$t_{AHBJTAG3}$	data input from clock hold	rising *dsutck* edge	TBD	-	ns
$t_{AHBJTAG4}$	clock to data output delay	falling *dsutck* edge	-	TBD	ns

7.10 Library dependencies

Table 59 shows libraries used when instantiating the core (VHDL libraries).

Table 59. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	JTAG	Signals, component	Signals and component declaration

7.11 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.jtag.all;

entity ahbjtag_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- JTAG signals
    tck  : in std_ulogic;
    tms  : in std_ulogic;
    tdi  : in std_ulogic;
    tdo  : out std_ulogic
);
end;

architecture rtl of ahbjtag_ex is
```

```
  -- AMBA signals
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
signal gnd : std_ulogic;

constant clkperiod : integer := 100;

begin

  gnd <= '0';

  -- AMBA Components are instantiated here
  ...

  -- AHBJTAG debug link
  ahbjtag0 : ahbjtag generic map(tech => 0, hindex => 1)
 port map(rstn, clkm, tck, tckn, tms, tdi, tdo, ahbmi, ahbmo(1),
          open, open, open, open, open, open, open, gnd);


  jtagproc : process
  begin
    jtagcom(tdo, tck, tms, tdi, 100, 20, 16#40000000#, true);
    wait;
  end process;

end;
```

7.12 Simulation

The *jtagtst* package in the *gaisler* library contains VHDL procedures *jread* and *jwrite* that can be used to perform read and write accesses over the JTAG debug link in simulation. The *jread* and *jwrite* procedures send, and return the results of, JTAG commands to the AHBJTAG using JTAG signals TCK, TMS, TDI and TDO. The *jread* and *jwrite* procedures have the following declarations:

```
procedure jwrite(addr, hsize, data    : in std_logic_vector;
                 signal tck, tms, tdi : out std_ulogic;
                 signal tdo           : in std_ulogic;
                 cp                   : in integer; -- 0.25 * TCK clock period
                 ainst                : in integer := 2;
                 dinst                : in integer := 3;
                 isize                : in integer := 6);

procedure jread(addr, hsize          : in  std_logic_vector;
                data                 : out std_logic_vector;
                signal tck, tms, tdi : out std_ulogic;
                signal tdo           : in std_ulogic;
                cp                   : in integer; -- 0.25 * TCK clock period
                reread               : in boolean := false;
                assertions           : in boolean := false;
                ainst                : in integer := 2;
                dinst                : in integer := 3;
                isize                : in integer := 6);
```

This JTAG test works if the generic JTAG tap controller is used (selected by setting the generic *tech* to 0) and generally does not work with built-in TAP macros common in FPGA since these macros often don't have visible JTAG pins. Though in some cases there are simulation macros available that expose JTAG pins for the built-in TAP (one example being JTAG_SIME2 providing access to BSCANE2 macros in some device families from Xilinx). In other cases the built-in TAP macro simulation models only implement a bypass register (this has been the case for the UJTAG used in some device families from Microsemi).

8 AHBLITM2AHBM - AHB-LITE to AHB MASTER

8.1 Overview

The AHBLITM2AHBM converts an AHB-LITE interface to an AHB master interface which in turns allows an AHB-LITE master to be connected to an AHB bus. This component is mainly intended to be used with the AHBCTRL of GRLIB IP library. Fig. 14 illustrates the usage of AHBLITM2AHBM.

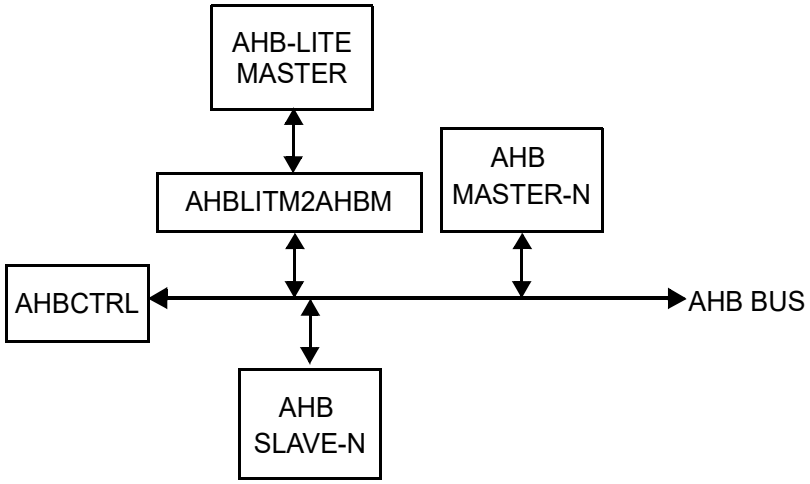


Figure 14. An AHB-LITE master is connected to an AHB BUS through AHBLITM2AHBM

8.1.1 AHB support

The AHBLITE2AHBM component currently supports the following features of the AHB protocol:

Transfer Type: IDLE, NONSEQ, SEQ

Burst Operation: SINGLE, INCR, INCR4, INCR8, INCR16

Data-width: 32-bit, 64-bit, 128-bit, 256-bit

Transfer Size: All possible transfer sizes up to the selected data-width are supported.

Response: OK, ERROR, RETRY, SPLIT

Unsupported AHB Features:

The following features of AHB protocol are not supported by the AHBLIT2AHBM component.

- BUSY transfer type.
- WRAP burst operations.

8.2 Operation

8.2.1 Considerations for INCR burst type.

The AHBCTRL in GRLIB IP library changes the arbitration during an INCR burst type (undefined length burst) when busreq signal is deasserted. When an AHB-LITE master connected to the AHBLITM2AHBM starts an INCR burst the busreq signal will be asserted and it will be deasserted

when IDLE sequence is encountered. As a result if AHB-LITE master issues back-to-back INCR burst transfers without any IDLE cycles in between the busreq will be keep asserted until IDLE sequence is encountered. In order to prevent AHBLITM2AHBM to hold the bus for a long time an IDLE sequence can be inserted between back-to-back INCR burst types.

8.2.2 ERROR, RETRY and SPLIT responses

AHBLIT2AHBM will handle the RETRY and SPLIT responses from AHB bus transparently to the AHB-LITE interface. Error responses from the AHB bus will be propagated directly to the AHB-LITE interface.

8.2.3 Endianness

The AHBLIT2AHBM component does not support any endianness conversion between the two buses and requires the endianness to be the same on both sides.

8.3 Input Interface

The AHBLIT2AHBM interface uses AHB-LITE input signals in a combinatorial way in order to drive the AHB bus signals. This is needed to prevent complex buffering schemes and have zero latency. As a result it is recommended to use registered outputs on the AHB-LITE master that is connected to the AHBLIT2AHBM component.

8.4 Implementation

8.4.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and reset all registers.

8.4.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core supports both big-endian and little-endian systems. The core assumes the same endianness on both sides, i.e. no endianness conversion is supported.

8.5 Configuration Options

Table 60. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0 - (NAHBMST-1)	0
venid	GRLIB plug&play vendor ID		GAISLER
devid	GRLIB plug&play device ID		AHBLITM2AHBM
version	GRLIB plug&play version		0

8.6 Signal descriptions

Table 61 shows the interface signals of the core (VHDL ports).

Table 61. Signal descriptions (VHDL ports)

Signal name	Field	Type	Function	Active
RSTN		Input	Reset	Low
CLK		Input	AHB & AHB-LITE clock	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
AHBMO_HTRANS		Input	AHB-LITE HTRANS input	-
AHBMO_HADDR		Input	AHB-LITE HADDR input	-
AHBMO_HWRITE		Input	AHB-LITE HWRITE input	-
AHBMO_HSIZE		Input	AHB-LITE HSIZE input	-
AHBMO_HWDATA		Input	AHB-LITE HWDATA input	-
AHBMO_HBURST		Input	AHB-LITE HBURST input	-
AHBMO_HPROT		Input	AHB-LITE HPROT input	-
AHBMI_HREADY		Output	AHB-LITE HREADY input	-
AHBMI_HRESP		Output	AHB-LITE HRESP output	-
AHBMi_HRDATA		Output	AHB-LITE HRDATA output	-

* see GRLIB IP Library User's Manual

8.7 Library dependencies

Table 62 shows the libraries used when instantiating the core (VHDL libraries).

Table 62. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions, Component declaration

8.8 Instantiation

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
```

```
entity ahblitm2ahbm_ex is
  port (
    rstn  : in  std_logic;
    clk   : in  std_logic;
);
end;


architecture rtl of ahblitm2ahbm_ex is

begin


ahblitm: example_ahblitm
port map (
 rstn => rstn,
 clk => clk,
 ahbmo_htrans => ahbmo_htrans,
 ahbmo_haddr => ahbmo_haddr,
 ahbmo_hwrite => ahbmo_hwrite,
 ahbmo_hsize => ahbmo_hsize,
 ahbmo_hwdata => ahbmo_hwdata,
 ahbmo_hburst => ahbmo_hburst,
 ahbmo_hprot => ahbmo_hprot,
 ahbmi_hready => ahbmi_hready,
 ahbmi_hresp => ahbmi_hresp,
 ahbmi_hrdata => ahbmi_hrdata);



u0:ahblitm2ahbm
generic map (
 hindex => 0)
port map (
 rstn => rstn,
 clk => clk,
 ahbmi => ahbmi, --connected to the AHBCTRL
 ahbmo => ahbmo(0), --connected to the AHBCTRL
 ahbmo_htrans => ahbmo_htrans,
 ahbmo_haddr => ahbmo_haddr,
 ahbmo_hwrite => ahbmo_hwrite,
 ahbmo_hsize => ahbmo_hsize,
 ahbmo_hwdata => ahbmo_hwdata,
 ahbmo_hburst => ahbmo_hburst,
 ahbmo_hprot => ahbmo_hprot,
 ahbmi_hready => ahbmi_hready,
 ahbmi_hresp => ahbmi_hresp,
 ahbmi_hrdata => ahbmi_hrdata);
```

9 AHBRAM - Single-port RAM with AHB interface

9.1 Overview

AHBRAM implements on-chip RAM with an AHB slave interface. Memory size is configurable in binary steps through a VHDL generic. Minimum size is 1KiB and maximum size is dependent on target technology and physical resources. Read accesses have zero or one waitstate (configured at implementation time), write access have one waitstate. The RAM supports byte- and half-word accesses, as well as all types of AHB burst accesses.

Internally, the AHBRAM instantiates a SYNCRAM block with byte writes. Depending on the target technology map, this will translate into memory with byte enables or to multiple 8-bit wide SYNCRAM blocks.

The size of the RAM implemented within AHBRAM can be read via the core's AMBA plug&play version field. The version field will display log2(number of bytes), for a 1 KiB SYNCRAM the version field will have the value 10, where 2^{10} = 1024 bytes = 1 KiB.

9.1.1 Endianness

The core is designed for big-endian systems.

9.2 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x00E. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

9.3 Implementation

9.3.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

9.3.2 Endianness

The core support both big-endian and little-endian systems. The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The endianness can be configured through the *endianness* generic.

9.4 Configuration options

Table 63 shows the configuration options of the core (VHDL generics).

Table 63. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave bus index	0 - NAHBSLV-1	0
haddr	The MSB address of the AHB area. Sets the 12 most significant bits in the 32-bit AHB address.	0 - 16#FFF#	16#FFF#
hmask	The AHB area address mask. Sets the size of the AHB area and the start address together with *haddr*.	0 - 16#FFF#	16#FF0#
tech	Technology to implement on-chip RAM	0 - NTECH	0
kbytes	RAM size in KiB. The size of the RAM implemented will be the minumum size that will hold the size specified by *kbytes*. A value of 1 here will instantiate a 1 KiB SYNCRAM, a value of 3 will instantiate a 4 KiB SYNCRAM. The actual RAM usage on the target technology then depends on the available RAM resources and the technology map.	target-dependent	1
pipe	Add registers on data outputs. If set to 0 the AMBA data outputs will be connected directly to the core's internal RAM. If set to 1 the core will include registers on the data outputs. Settings this generic to 1 makes read accesses have one waitstate, otherwise the core will respond to read accesses with zero waitstates.	0 - 1	0
maccsz	Maximum access size supported. This generic restricts the maximum AMBA access size supported by the core and selects the width of the SYNCRAMBW RAM used internally. The default value is assigned from AHBDW, which sets the maximum bus width for the GRLIB design.	32, 64, 128, 256	AHBDW
scantest	Enable scan test support (passed on to syncram)	0 - 1	0
endianness	Select endianness. Big endian if set to 0 and little endian if set to 1.	0 - 1	0

9.5 Signal descriptions

Table 64 shows the interface signals of the core (VHDL ports).

Table 64. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AMB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-

* see GRLIB IP Library User's Manual

9.6 Library dependencies

Table 65 shows libraries used when instantiating the core (VHDL libraries).

Table 65. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions
GAISLER	MISC	Component	Component declaration

9.7 Component declaration

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

component ahbram
generic ( hindex : integer := 0; haddr : integer := 0; hmask : integer := 16#fff#;
tech : integer := 0; kbytes : integer := 1);
port (
rst : in std_ulogic;
clk : in std_ulogic;
ahbsi : in ahb_slv_in_type;
ahbso : out ahb_slv_out_type
);
end component;
```

9.8 Instantiation

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

.
.

ahbram0 : ahbram generic map (hindex => 7, haddr => CFG_AHBRADDR,
tech => CFG_MEMTECH, kbytes => 8)
    port map ( rstn, clkm, ahbsi, ahbso(7));
```

10 AHBDPRAM - Dual-port RAM with AHB interface

10.1 Overview

AHBDPRAM implements a 32-bit wide on-chip RAM with one AHB slave interface port and one back-end port for a user application. The AHBDPRAM is therefore useful as a buffer memory between the AHB bus and a custom IP core with a RAM interface

The memory size is configurable in binary steps through the *abits* VHDL generic. The minimum size is 1kB while maximum size is dependent on target technology and physical resources. Read accesses are zero-waitstate, write access have one waitstate. The RAM optionally supports byte- and half-word accesses, as well as all types of AHB burst accesses. Internally, the AHBRAM instantiates one 32-bit or four 8-bit wide SYNCRAM_DP blocks. The target technology must have support for dual-port RAM cells.

The back-end port consists of separate clock, address, datain, dataout, enable and write signals. All these signals are sampled on the rising edge of the back-end clock (CLKDP), implementing a synchronous RAM interface. Read-write collisions between the AHB port and the back-end port are not handled and must be prevented by the user. If byte write is enabled, the WRITE(0:3) signal controls the writing of each byte lane in big-endian fashion. WRITE(0) controls the writing of DATAIN(31:24) and so on. If byte write is disabled, WRITE(0) controls writing to the complete 32-bit word.

10.1.1 Endianness

The core is designed for big-endian systems.

10.2 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x00F. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

10.3 Implementation

10.3.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset.

10.3.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

10.4 Configuration options

Table 66 shows the configuration options of the core (VHDL generics).

Table 66. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave bus index	0 - NAHBSLV-1	0
haddr	The MSB address of the AHB area. Sets the 12 most significant bits in the 32-bit AHB address.	0 - 16#FFF#	16#FFF#
hmask	The AHB area address mask. Sets the size of the AHB area and the start address together with *haddr*.	0 - 16#FFF#	16#FF0#
tech	Technology to implement on-chip RAM	0 - NTECH	2
abits	Address bits. The RAM size in Kbytes is equal to 2**(abits +2)	8 - 19	8
bytewrite	If set to 1, enabled support for byte and half-word writes	0 - 1	0
cacheable	Set to configure the memory area as prefetchable and cacheable on the AHB side of the memory.	0 - 1	1

10.5 Signal descriptions

Table 67 shows the interface signals of the core (VHDL ports).

Table 67. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AHB Reset	Low
CLK	N/A	Input	AHB Clock	-
AHBSI	*	Input	AMB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
CLKDP		Input	Clock for back-end port	-
ADDRESS(abits-1:0)		Input	Address for back-end port	-
DATAIN(31 : 0)		Input	Write data for back-end port	-
DATAOUT(31 : 0)		Output	Read data from back-end port	-
ENABLE		Input	Chip select for back-end port	High
WRITE(0 : 3)		Input	Write-enable byte select for back-end port	High

* see GRLIB IP Library User's Manual

10.6 Library dependencies

Table 68 shows libraries used when instantiating the core (VHDL libraries).

Table 68. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions
GAISLER	MISC	Component	Component declaration

10.7 Component declaration

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;
```

```
component ahbdpram
  generic (
    hindex  : integer := 0;
    haddr   : integer := 0;
    hmask   : integer := 16#fff#;
    tech    : integer := 2;
    abits   : integer range 8 to 19 := 8;
    bytewrite : integer range 0 to 1 := 0
  );
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;
    ahbso   : out ahb_slv_out_type;
    clkdp   : in std_ulogic;
    address : in std_logic_vector((abits -1) downto 0);
    datain  : in std_logic_vector(31 downto 0);
    dataout : out std_logic_vector(31 downto 0);
    enable  : in std_ulogic;-- active high chip select
    write   : in std_logic_vector(0 to 3)-- active high byte write enable
  );
  end component;
```

11 AHBROM - Single-port ROM with AHB interface

11.1 Overview

The AHBROM core implements a 32/64/128-bit wide on-chip ROM with an AHB slave interface. Read accesses take zero waitstates, or one waitstate if the pipeline option is enabled. The ROM supports byte- and half-word accesses, as well as all types of AHB burst accesses.

11.2 PROM generation

The AHBPROM is automatically generated by the make utility in GRLIB. The input format is a sparc-elf binary file or a riscv-elf binary file, produced respectively by the BCC cross-compiler (sparc-elf-gcc) or by a RISCV compiler (riscv-elf-gcc). To create a PROM, first compile a suitable binary and then run the make utility:

```
bash$ sparc-elf-gcc prom.S -o prom.exe
bash$ make ahbrom.vhd

Creating ahbrom.vhd : file size 272 bytes, address bits 9
```

The default binary file for creating a PROM is prom.exe. To use a different file, run make with the FILE parameter set to the input file:

```
bash$ make ahbrom.vhd FILE=myfile.exe
```

The created PROM is realized in synthesizable VHDL code, using a CASE statement. For FPGA targets, most synthesis tools will map the CASE statement on a block RAM/ROM if available. For ASIC implementations, the ROM will be synthesized as gates. It is then recommended to use the *pipe* option to improve the timing.

The default is to build a 32-bit wide ahbrom, to instead build 64-bit or 128-bit wide ahbrom versions, use the flow described above but with the “make ahbrom64.vhd” and “make ahbrom128.vhd” make targets.

11.2.1 Endianness

The core supports both big-endian and little-endian systems.

11.3 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x01B. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

11.4 Implementation

11.4.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User’s Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support the GRLIB config package setting *grlib_async_reset_enable*.

11.4.2 Endianness

The endianness of the core is selected through the AHB sideband signal *ahbsi.endian*. The AHB bus controller (i.e. AHBCTRL) drives this signal, according to its generic *ahbendian*.

11.5 Configuration options

Table 69 shows the configuration options of the core (VHDL generics).

Table 69. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave bus index	0 - NAHBSLV-1	0
haddr	The MSB address of the AHB area. Sets the 12 most significant bits in the 32-bit AHB address.	0 - 16#FFF#	16#FFF#
hmask	The AHB area address mask. Sets the size of the AHB area and the start address together with *haddr*.	0 - 16#FFF#	16#FF0#
tech	Not used		
pipe	Add a pipeline stage on read data	0	0
kbytes	Not used		
Only on ahbrom64 and ahbrom128:			
wideonly	Removes muxing logic needed to properly support 32-bit masters on wide bus	0 - 1	0

11.6 Signal descriptions

Table 70 shows the interface signals of the core (VHDL ports).

Table 70. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AMB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-

* see GRLIB IP Library User's Manual

11.7 Library dependencies

Table 71 shows libraries used when instantiating the core (VHDL libraries).

Table 71. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions

11.8 Component declaration

```
component ahbrom
generic ( hindex : integer := 0; haddr : integer := 0; hmask : integer := 16#fff#;
pipe : integer := 0; tech : integer := 0);
port (
rst : in std_ulogic;
clk : in std_ulogic;
ahbsi : in ahb_slv_in_type;
```

```
ahbso : out ahb_slv_out_type
);
end component;
```

11.9 Instantiation

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
.
.

brom : entity work.ahbrom
      generic map (hindex => 8, haddr => CFG_AHBRODDR, pipe => CFG_AHBROPIP)
      port map ( rstn, clkm, ahbsi, ahbso(8));
```

12 AHBSTAT - AHB Status Registers

12.1 Overview

The status registers store information about AMBA AHB accesses triggering an error response. There is a status register and a failing address register capturing the control and address signal values of a failing AMBA bus transaction, or the occurrence of a correctable error being signaled from a another peripheral in the system.

The status register and the failing address register are accessed from the AMBA APB bus.

12.2 Operation

12.2.1 Errors

The registers monitor AMBA AHB bus transactions and store the current HADDR, HWRITE, HMASTER and HSIZE internally. The monitoring are always active after startup and reset until an error response (HRESP = “01”) is detected. When the error is detected, the status and address register fields CE, HWRITE, HMASTER, HSIZE and HADDR are frozen and the New Error (NE) bit is set to one. At the same time an interrupt is generated, as described hereunder.

Note that many of the fault tolerant units containing EDAC signal an un-correctable error as an AMBA error response, so that it can be detected by the processor as described above.

12.2.2 Correctable errors

Not only error responses on the AHB bus can be detected. Many of the fault tolerant units containing EDAC have a correctable error signal which is asserted each time a correctable error is detected. When such an error is detected, the effect will be the same as for an AHB error response. The only difference is that the Correctable Error (CE) bit in the status register is set to one when a correctable error is detected.

When the CE bit is set the interrupt routine can acquire the address containing the correctable error from the failing address register and correct it. When it is finished it resets the NE bit and the monitoring becomes active again. Interrupt handling is described in detail hereunder.

The correctable error signals from the fault tolerant units should be connected to the *stati.cerror* input signal vector of the AHB status register core, which is or-ed internally and if the resulting signal is asserted, it will have the same effect as an AHB error response.

12.2.3 Interrupts

The interrupt is generated on the line selected by the *pirq* VHDL generic.

The interrupt is connected to the interrupt controller to inform the processor of the error condition. The normal procedure is that an interrupt routine handles the error with the aid of the information in the status registers. When it is finished it resets the NE bit (and ME bit, if implemented) and the monitoring becomes active again. Interrupts are generated for both AMBA error responses and correctable errors as described above.

12.2.4 Filtering and multiple error detection

The status register can optionally be implemented with two sets of status and failing address register. In this case the core also supports filtering on errors and has a status bit that gets set in case additional errors are detected when the New Error (NE) bit is set. The core will only react to the first error in a burst operation. After the first error has been detected, monitoring of the burst is suspended.

An error event will only be recorded by the first status register that should react based on filter settings. If register set 1 has reacted then register 2 will not be set for the same error event. In the case

where an error occurs and all register sets that could have recorded an error, based on filter settings, already have the NE bit set then the ME bit will be set for all the matching register sets.

The extra register set, filtering, and multiple error detection is available in revision 1 of the status register. The functionality is enabled through the *ver* VHDL generic. The value of this generic also affects the core version in the GRLIB plug&play information.

12.3 Registers

The core is programmed through registers mapped into APB address space.

Table 72. AHB Status registers

APB address offset	Registers
0x00	AHB Status register
0x04	AHB Failing address register
0x08	AHB Status register 2 (optional)
0x0C	AHB Failing Address register 2 (optional)

12.3.1 AHB Status register

Table 73. 0x00, 0x08- AHBS - AHB Status register

31 14	13	12	11	10	9	8	7	6 3	2 0
RESERVED	ME	FW	CF	AF	CE	NE	HWRITE	HMASTER	HSIZE
0	0	0	0	0	0	0	NR	NR	NR
r	rw*	w*	rw*	rw*	rw	rw	r	r	r

31: 14	RESERVED
13	Multiple Error detection (ME) - This field is set to 1 when the New Error bit is set and one more error is detected. Filtering is considered when setting the ME bit. This field is only available in version 1 of the core (version is selected at implementation).
12	Filter Write (FW) - This bit needs to be set to '1' during a write operation for CF and AF fields to be updated in the same write operation. Always reads as zero. This field is only available in version 1 of the core (version is selected at implementation).
11	Correctable Error Filter (CF) - If this bit is set to 1 then this status register will ignore correctable errors. This field will only be written if the FW bit is set. This field is only available in version 1 of the core (version is selected at implementation).
10	AMBA ERROR Filter (AF) - If this bit is set to 1 then this status register will ignore AMBA ERROR. This field will only be written if the FW bit is set. This field is only available in version 1 of the core (version is selected at implementation).
9	Correctable Error (CE) - Set if the detected error was caused by a correctable error and zero otherwise.
8	New Error (NE) - Deasserted at start-up and after reset. Asserted when an error is detected. Reset by writing a zero to it.
7	The HWRITE signal of the AHB transaction that caused the error.
6: 3	The HMASTER signal of the AHB transaction that caused the error.
2: 0	The HSIZE signal of the AHB transaction that caused the error

12.3.2 AHB Failing address register

Table 74. 0x04, 0x0C - AHBFAR - AHB Failing address register

31 0
AHB FAILING ADDRESS
NR
t

31: 0	The HADDR of the AHB transaction that caused the error.

12.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x052. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

12.5 Implementation

12.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). When reset is asserted the new error and correctable error registers are reset to zero.

12.6 Configuration options

Table 75 shows the configuration options of the core (VHDL generics).

Table 75. Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAHBSLV-1	0
paddr	APB address	0 - 16#FFF#	0
pmask	APB address mask	0 - 16#FFF#	16#FFF#
pirq	Interrupt line driven by the core	0 - 16#FFF#	0
nftslv	Number of FT slaves connected to the cerror vector	1 - NAHBSLV-1	3
ver	Selects version of the core. Setting this value to 1 implements the two sets of registers, multiple error detection, and filter functionality.	0 - 1	0

12.7 Signal descriptions

Table 76 shows the interface signals of the core (VHDL ports).

Table 76. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AHB slave input signals	-
AHBSI	*	Input	AHB slave output signals	-
STATI	CERROR	Input	Correctable Error Signals	High
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-

* see GRLIB IP Library User's Manual

12.8 Library dependencies

Table 77 shows libraries used when instantiating the core (VHDL libraries).

Table 77. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MISC	Component	Component declaration

12.9 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the status register. There are three Fault Tolerant units with EDAC connected to the status register *cerror* vector. The connection of the different memory controllers to external memory is not shown.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
```

```
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;
  --other signals
  ....
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo, sdo2: sdctrl_out_type;

  signal sdi : sdctrl_in_type;

-- correctable error vector
  signal stati : ahbstat_in_type;
  signal aramo : ahbram_out_type;

begin

  -- AMBA Components are defined here ...

-- AHB Status Register
  astat0 : ahbstat generic map(pindex => 13, paddr => 13, pirq => 11,
    nftslv => 3)
    port map(rstn, clkm, ahbmi, ahbsi, stati, apbi, apbo(13));
    stati.cerror(3 to NAHBSLV-1) <= (others => '0');

--FT AHB RAM
  a0 : ftahbram generic map(hindex => 1, haddr => 1, tech => inferred,
    kbytes => 64, pindex => 4, paddr => 4, edacen => 1, autoscrub => 0,
    errcnt => 1, cntbits => 4)
    port map(rst, clk, ahbsi, ahbso, apbi, apbo(4), aramo);
    stati.cerror(0) <= aramo.ce;
-- SDRAM controller
  sdc : ftsdctrl generic map (hindex => 3, haddr => 16#600#, hmask => 16#F00#,
    ioaddr => 1, fast => 0, pwron => 1, invclk => 0, edacen => 1, errcnt => 1,
    cntbits => 4)
    port map (rstn, clk, ahbsi, ahbso(3), sdi, sdo);
    stati.cerror(1) <= sdo.ce;

-- Memory controller
  mctrl0 : ftsrctrl generic map (rmw => 1, pindex => 10, paddr => 10,
    edacen => 1, errcnt => 1, cntbits => 4)
    port map (rstn, clk, ahbsi, ahbso(0), apbi, apbo(10), memi, memo, sdo2);
    stati.cerror(2) <= memo.ce;
end;
```

13 AHBTRACE - AHB Trace buffer

13.1 Overview

The trace buffer consists of a circular buffer that stores AMBA AHB data transfers. The address, data and various control signals of the AHB bus are stored and can be read out for later analysis.

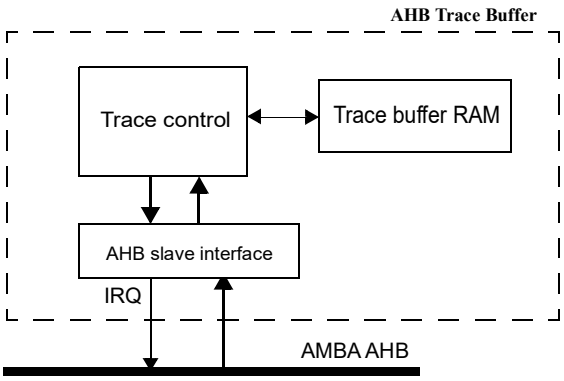


Figure 15. Block diagram

When the trace buffer is configured in 32-bit bus mode, it is 128 bits wide. The information stored is indicated in the table below:

Table 78. AHB Trace buffer data allocation

Bits	Name	Definition
127:96	Time tag	The value of the time tag counter
95	AHB breakpoint hit	Set to '1' if a DSU AHB breakpoint hit occurred.
94:80	-	Not used
79	Hwrite	AHB HWRITE
78:77	Htrans	AHB HTRANS
76:74	Hsize	AHB HSIZE
73:71	Hburst	AHB HBURST
70:67	Hmaster	AHB HMASTER
66	Hmastlock	AHB HMASTLOCK
65:64	Hresp	AHB HRESP
63:32	Load/Store data	AHB HRDATA[31:0] or HWDATA[31:0]
31:0	Load/Store address	AHB HADDR

In addition to the AHB signals, a 32-bit counter is also stored in the trace as time tag.

When the trace buffer is configured in 64-bit or 128-bit bus mode, its contents are extended according to the table below.

Bits	Name	Definition
223:160	128-bit extended load/store data	AHB HRDATA[127:64] or HWDATA[127:64]
159:128	64-bit extended load/store data	AHB HRDATA[63:32] or HWDATA[63:32]

13.2 Operation

13.2.1 Overview

The trace buffer is enabled by setting the enable bit (EN) in the trace control register. Each AMBA AHB transfer is then stored in the buffer in a circular manner. The address to which the next transfer is written is held in the trace buffer index register, and is automatically incremented after each transfer. Tracing is stopped when the EN bit is reset, or when a AHB breakpoint is hit. An interrupt is generated when a breakpoint is hit.

Note: the LEON3 and LEON4 Debug Support Units (DSU3/DSU4) also includes an AHB trace buffer. The standalone trace buffer is intended to be used in system without a processor or when the DSU3 is not present.

The size of the trace buffer is configured by means of the *kbytes* VHDL generic, defining the size of the complete buffer in kbytes.

The number of lines in the trace buffer is *kbytes* * 1024 / 16 bytes.

The total size of the trace buffer depends on the *bwidth* generic. When the ahb trace buffer is in 32-bit bus mode, the size of the buffer is simply *kbytes* kbytes.

When the ahb trace buffer is configured in 64-bit or 128-bit bus mode, the *kbytes* generic will not reflect the exact amount of memory used in the core. You will have to multiply each line, calculated as above, for 20 bytes or 28 bytes, for 64-bit bus mode or 128-bit bus mode respectively. Therefore the total size for the buffer when in 64-bit mode is *kbytes* * 1.25 kbytes, and for the buffer in 128-bit bus mode it is *kbytes* * 1.75 kbytes.

13.2.2 AHB statistics

The core can be implemented to generate statistics from the traced AHB bus. When statistics collection is enabled the core will assert outputs that are suitable to connect to a LEON statistics unit (L3STAT and L4STAT). The statistical outputs can be filtered by the AHB trace buffer filters, this is controlled by the Performance counter Filter bit (PF) in the AHB trace buffer control register. The core can collect data for the events listed in table 79 below.

Table 79. AHB events

Event	Description	Note
idle	HTRANS=IDLE	Active when HTRANS IDLE is driven on the AHB slave inputs and slave has asserted HREADY.
busy	HTRANS=BUSY	Active when HTRANS BUSY is driven on the AHB slave inputs and slave has asserted HREADY.
nseq	HTRANS=NONSEQ	Active when HTRANS NONSEQ is driven on the AHB slave inputs and slave has asserted HREADY.
seq	HTRANS=SEQ	Active when HTRANS SEQUENTIAL is driven on the AHB slave inputs and slave has asserted HREADY.
read	Read access	Active when HTRANS is SEQUENTIAL or NON-SEQUENTIAL, slave has asserted HREADY and the HWRITE input is low.
write	Write access	Active when HTRANS is SEQUENTIAL or NON-SEQUENTIAL, slave has asserted HREADY and the HWRITE input is high.
hsize[5:0]	Transfer size	Active when HTRANS is SEQUENTIAL or NON-SEQUENTIAL, slave has asserted HREADY and HSIZE is BYTE (hsize[0]), HWORD (HSIZE[1]), WORD (hsize[2]), DWORD (hsize[3]), 4WORD hsize[4], or 8WORD (hsize[5]).
ws	Wait state	Active when HREADY input to AHB slaves is low and AMBA response is OKAY.
retry	RETRY response	Active when master receives RETRY response

Table 79. AHB events

Event	Description	Note
split	SPLIT response	Active when master receives SPLIT response
spdel	SPLIT delay	Active during the time a master waits to be granted access to the bus after reception of a SPLIT response. The core will only keep track of one master at a time. This means that when a SPLIT response is detected, the core will save the master index. This event will then be active until the same master is re-allowed into bus arbitration and is granted access to the bus. This also means that the delay measured will include the time for re-arbitration, delays from other ongoing transfers and delays resulting from other masters being granted access to the bus before the SPLIT:ed master is granted again after receiving SPLIT complete. If another master receives a SPLIT response while this event is active, the SPLIT delay for the second master will not be measured.
locked	Locked access	Active while the HMASTLOCK signal is asserted on the AHB slave inputs. (Currently not used by L3STATand L4STAT)

13.3 Registers

13.3.1 Register address map

The trace buffer occupies 128 KiB of address space in the AHB I/O area. The address mapping in parentheses is only available when the core is in 64-bit or 128-bit bus mode. Only 32-bit single-accesses to the area are supported. The following register addresses are decoded:

Table 80. Trace buffer address space

Address	Register
0x000000	Trace buffer control register
0x000004	Trace buffer index register
0x000008	Time tag counter
0x00000C	Trace buffer master/slave filter register
0x000010	AHB break address 1
0x000014	AHB mask 1
0x000018	AHB break address 2
0x00001C	AHB mask 2
0x010000 - 0x020000	Trace buffer
...0	Trace bits 127 - 96
...4	Trace bits 95 - 64
...8	Trace bits 63 - 32
...C	Trace bits 31 - 0
(...10)	Trace bits 159 - 128, when in 64- or 128-bit bus mode
(...14)	Trace bits 223 - 192, when in 128-bit bus mode
(...18)	Trace bits 191 - 160, when in 128-bit bus mode
(...1C)	Zero

13.3.2 Trace buffer control register

The trace buffer is controlled by the trace buffer control register:

Table 81. 0x000000 - CTRL - Trace buffer control register

31 16	15	14 12	11 10	9	8	7 6	5	4	3	2	1	0
DCNT	BA	BSEL	RES	EF	PF	BW	RF	AF	FR	FW	DM	EN
0	*	0	0	(nr)	(nr)	*	(nr)	(nr)	(nr)	(nr)	(nr)	*
rw	r	rw	r	rw	rw	r	rw	rw	rw	rw	r	rw

31: 16 Trace buffer delay counter (DCNT) - Note that the number of bits actually implemented depends on the size of the trace buffer.

15 Bus select Available (BA) - If this field is set to '1', the core has several buses connected. The bus to trace is selected via the BSEL field. If this field is '0', the core is only capable of tracing one AHB bus.

14: 12 Bus select (BSEL) - If the BA field is '1' this field selects the bus to trace. If the BA field is '0', this field is not writable.

11: 10 RESERVED

9 Error filter (EF) - When set to 1, only error responses will be included in the trace buffer.

8 Performance counter Filter (PF) - If this bit is set to '1', the cores performance counter (statistical) outputs will be filtered using the same filter settings as used for the trace buffer. If a filter inhibits a write to the trace buffer, setting this bit to '1' will cause the same filter setting to inhibit the pulse on the statistical output.

7: 6 Bus width (BW) - This value corresponds to log2(Supported bus width / 32)

5 Retry filter (RF) - If this bit is set to '1', AHB retry responses will not be included in the trace buffer. This bit can only be set of the core has been implemented with support for filtering

4 Address Filter (AF) - If this bit is set to '1', only the address range defined by AHB trace buffer breakpoint 2's address and mask will be included in the trace buffer. This bit can only be set of the core has been implemented with support for filtering

3 Filter Reads (FR) - If this bit is set to '1', read accesses will not be included in the trace buffer. This bit can only be set of the core has been implemented with support for filtering.

2 Filter Writes (FW) - If this bit is set to '1', write accesses will not be included in the trace buffer. This bit can only be set of the core has been implemented with support for filtering.

1 Delay counter mode (DM) - Indicates that the trace buffer is in delay counter mode.

0 Trace enable (EN) - Enables the trace buffer

13.3.3 Trace buffer index register

The trace buffer index register indicates the address of the next 128-bit line to be written.

Table 82. 0x000004 - INDEX - Trace buffer index register

31 4	3 0
INDEX	0x0
NR	0
rw	r

31: 4 Trace buffer index counter (INDEX). Note that the number of bits actually implemented depends on the size of the trace buffer

3: 0 Read as 0x0

13.3.4 Trace buffer time tag register

The time tag register contains a 32-bit counter that increments each clock when the trace buffer is enabled. The value of the counter is stored in the trace to provide a time tag.

Table 83. 0x000008 - TIMETAG - Trace buffer time tag counter

31	0
TIME TAG VALUE	
0	
r	

13.3.5 Trace buffer master/slave filter register

The master/slave filter register allows filtering out specified master and slaves from the trace. This register can only be assigned if the trace buffer has been implemented with support for filtering.

Table 84. Trace buffer master/slave filter register

31	16	15	0
SMASK[15:0]		MMASK[15:0]	
0		0	
rw		rw	

31: 16 Slave Mask (SMASK) - If SMASK[n] is set to '1', the trace buffer will not save accesses performed to slave n.

15: 0 Master Mask (MMASK) - If MMASK[n] is set to '1', the trace buffer will not save accesses performed by master n.

13.3.6 Trace buffer breakpoint registers

The DSU contains two breakpoint registers for matching AHB addresses. A breakpoint hit is used to freeze the trace buffer by clearing the enable bit. Freezing can be delayed by programming the DCNT field in the trace buffer control register to a non-zero value. In this case, the DCNT value will be decremented for each additional trace until it reaches zero and after two additional entries, the trace buffer is frozen. A mask register is associated with each breakpoint, allowing breaking on a block of addresses. Only address bits with the corresponding mask bit set to '1' are compared during breakpoint detection. To break on AHB load or store accesses, the LD and/or ST bits should be set.

Table 85. Trace buffer AHB breakpoint address register

31	2	1	0
BADDR[31:2]		0b00	
NR		0	
rw		r	

31: 2 Breakpoint address (BADDR) - Bits 31:2 of breakpoint address

1: 0 Reserved, read as 0

Table 86. Trace buffer AHB breakpoint mask register

31	2	1	0
BMASK[31:2]		LD	ST
NR		0	0
rw		rw	rw

31: 2 Breakpoint mask (BMASK) - Bits 31:2 of breakpoint mask

1 Load (LD) - Break on data load address

0 Store (ST) - Break on data store address

13.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x017. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

13.5 Implementation

13.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

13.5.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core supports both big-endian and little-endian systems.

13.6 Configuration options

Table 87 shows the configuration options of the core (VHDL generics).

Table 87. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave bus index	0 - NAHBSLV-1	0
ioaddr	The MSB address of the I/O area. Sets the 12 most significant bits in the 20-bit I/O address.	0 - 16#FFF#	16#000#
iomask	The I/O area address mask. Sets the size of the I/O area and the start address together with ioaddr.	0 - 16#FFF#	16#E00#
irq	Interrupt number	0 - NAHBIRQ-1	0
tech	Technology to implement on-chip RAM	0 - NTECH	0
kbytes	Trace buffer size in kbytes	1 - 64	1
bwidth	Traced AHB bus width	32, 64, 128	64
ahbfilt	If this generic is set to 1 the core will be implemented with support for AHB trace buffer filters. If *ahbpf* is larger than 1 then the core's statistical outputs will be enabled.	0 - 2	0
ntrace	Number of buses to trace. This generic is only available if the entity ahbtrace_mmb is instantiated.	1 - 8	1
scantest	Support scan test and memory BIST	0 - 1	0
exttimer	If set to 1 then the time tag value will be taken from the core's timer signal input. Otherwise the core will use an internal timer.	0 - 1	0

13.7 Signal descriptions

Table 88 shows the interface signals of the core (VHDL ports).

Table 88. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AHB master input signals	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
TIMER_EN	*	Input	External signal to enable tracing at run-time (logically AND-ed with trace enable configuration register)	High
TIMER[30:0]	N/A	Input	External timestamp (only used when VHDL generic exttimer is nonzero). Suitable for connection to dbgo.timer signal from debug support unit (DSU IP Core)	-
ASTAT	*	Output	AHB statistics outputs. Intended to be connected to L3STAT and L4STAT core.	-
RESEN	*	Input	Reset value for trace enable	

* see GRLIB IP Library User's Manual

13.8 Library dependencies

Table 89 shows libraries used when instantiating the core (VHDL libraries).

Table 89. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions
GAISLER	MISC	Component	Component declaration

13.9 Component declaration

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

component ahbtrace is
  generic (
    hindex  : integer := 0;
    ioaddr  : integer := 16#000#;
    iomask  : integer := 16#E00#;
    tech    : integer := 0;
    irq     : integer := 0;
    kbytes  : integer := 1;
    exttimer : integer range 0 to 1 := 0);
  port (
    rst    : in  std_ulogic;
    clk    : in  std_ulogic;
    ahbmi  : in  ahb_mst_in_type;
    ahbsi  : in  ahb_slv_in_type;
    ahbso  : out ahb_slv_out_type;
    timer  : in  std_logic_vector(30 downto 0) := (others => '0'));
end component;

-- Tracebuffer that can trace separate bus:
```

```
component ahbtrace_mb is
  generic (
    hindex  : integer := 0;
    ioaddr    : integer := 16#000#;
    iomask    : integer := 16#E00#;
    tech    : integer := DEFMEMTECH;
    irq     : integer := 0;
    kbytes  : integer := 1;
    exttimer : integer range 0 to 1 := 0);
  port (
    rst    : in  std_ulogic; clk    : in  std_ulogic;
    ahbsi  : in  ahb_slv_in_type;       -- Register interface
    ahbso  : out ahb_slv_out_type;
    tahbmi : in  ahb_mst_in_type; tahbsi : in  ahb_slv_in_type -- Trace
    timer  : in  std_logic_vector(30 downto 0) := (others => '0'));
  end component;

-- Tracebuffer that can trace several separate buses:
component ahbtrace_mmb is
  generic (
    hindex  : integer := 0;
    ioaddr    : integer := 16#000#;
    iomask    : integer := 16#E00#;
    tech    : integer := DEFMEMTECH;
    irq     : integer := 0;
    kbytes  : integer := 1;
    ntrace  : integer range 1 to 8 := 1;
    exttimer : integer range 0 to 1 := 0);
  port (
    rst     : in  std_ulogic; clk    : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;       -- Register interface
    ahbso   : out ahb_slv_out_type;
    tahbmiv : in  ahb_mst_in_vector_type(0 to ntrace-1);
    tahbsiv : in  ahb_slv_in_vector_type(0 to ntrace-1) -- Trace
    timer   : in  std_logic_vector(30 downto 0) := (others => '0'));
  end component;
```

14 AHBUART- AMBA AHB Serial Debug Interface

14.1 Overview

The interface consists of a UART connected to the AMBA AHB bus as a master. A simple communication protocol is supported to transmit access parameters and data. Through the communication link, a read or write transfer can be generated to any address on the AMBA AHB bus.

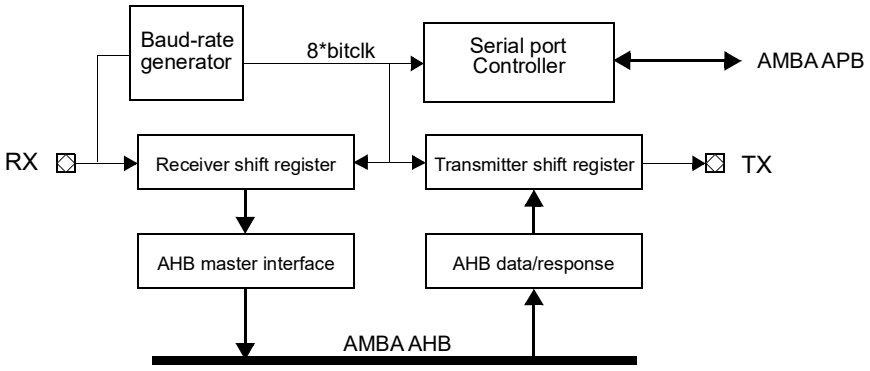


Figure 16. **Block diagram**

14.2 Operation

14.2.1 Transmission protocol

The interface supports a simple protocol where commands consist of a control byte, followed by a 32-bit address, followed by optional write data. Write access does not return any response, while a read access only returns the read data. Data is sent on 8-bit basis as shown below.

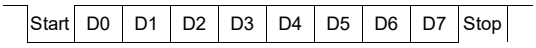


Figure 17. **Data frame**

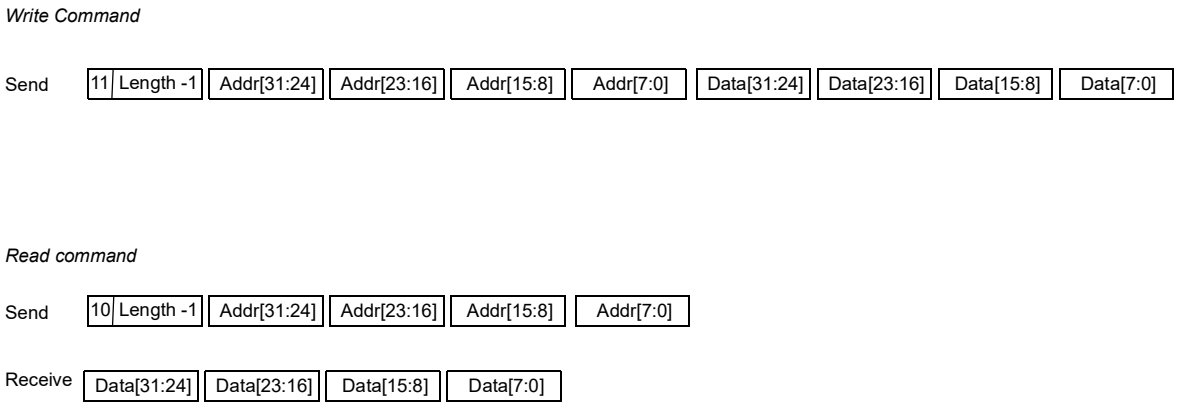


Figure 18. **Commands**

Block transfers can be performed be setting the length field to n-1, where n denotes the number of transferred words. For write accesses, the control byte and address is sent once, followed by the number of data words to be written. The address is automatically incremented after each data word. For

read accesses, the control byte and address is sent once and the corresponding number of data words is returned.

14.2.2 Baud rate generation

The UART contains a 18-bit down-counting scaler to generate the desired baud-rate. The scaler is clocked by the system clock and generates a UART tick each time it underflows. The scaler is reloaded with the value of the UART scaler reload register after each underflow. The resulting UART tick frequency should be 8 times the desired baud-rate.

If not programmed by software, the baud rate will be automatically discovered. This is done by searching for the shortest period between two falling edges of the received data (corresponding to two bit periods). When three identical two-bit periods has been found, the corresponding scaler reload value is latched into the reload register, and the BL and RXEN bits are set in the UART control register. If the BL bit is reset by software, the baud rate discovery process is restarted. The baud-rate discovery is also restarted when a 'break' or framing error is detected by the receiver, allowing to change to baudrate from the external transmitter. For proper baudrate detection, the value 0x55 should be transmitted to the receiver after reset or after sending break.

The best scaler value for manually programming the baudrate can be calculated as follows:

```
scaler = (((system_clk*10)/(baudrate*8))-5)/10
```

14.3 Registers

The core is programmed through registers mapped into APB address space.

Table 90. AHB UART registers

APB address offset	Register
0x4	AHB UART status register
0x8	AHB UART control register
0xC	AHB UART scaler register

14.3.1 AHB UART control register

Table 91. 0x08 - CTRL - AHB UART control register

31	2	1	2
RESERVED		BL	EN
0		0	0
r		rw	rw

0: Receiver enable (EN) - if set, enables both the transmitter and receiver. Reset value: ‘0’.

1: Baud rate locked (BL) - is automatically set when the baud rate is locked. Reset value: ‘0’.

14.3.2 AHB UART status register

Table 92. 0x04 - STAT - AHB UART status register

31	10	9	8	7	6	5	4	3	2	1	0
RESERVED		TCNT		RX	FE	R	OV	BR	TH	TS	DR
		00		0	0	0	0	0	1	1	0
		r		r	rw	r	rw	rw	r	r	r

0: Data ready (DR) - indicates that new data has been received by the AMBA AHB master interface. Read only. Reset value: ‘0’.

1: Transmitter shift register empty (TS) - indicates that the transmitter shift register is empty. Read only. Reset value: ‘1’

2: Transmitter hold register empty (TH) - indicates that the transmitter hold register is empty. Read only. Reset value: ‘1

3: Break (BR) - indicates that a BREAKE has been received. Reset value: ‘0’

4: Overflow (OV) - indicates that one or more character have been lost due to receiver overflow. Reset value: ‘0’

6: Frame error (FE) - indicates that a framing error was detected. Reset value: ‘0’

7: Input state (RX) - Filtered input state. Reset value: ‘0’

9: 8 Counter State (TCNT) - Internal Counter state. Reset value: “00”

14.3.3 AHB UART scaler register

Table 93. 0x0C - SCALER - AHB UART scaler register

31	18	17	0
RESERVED		SCALER RELOAD VALUE	
0		0x3FFFB	
r		rw	

17: 0 Baudrate scaler reload value = (((system_clk*10)/(baudrate*8))-5)/10. Reset value: “3FFFF“.

14.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x007. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

14.5 Implementation

14.5.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User’s Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

14.5.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual).

14.6 Configuration options

Table 94 shows the configuration options of the core (VHDL generics).

Table 94. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#

14.7 Signal descriptions

Table 95 shows the interface signals of the core (VHDL ports)..

Table 95. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
UARTI	RXD	Input	UART receiver data	High
	CTSN	Input	UART clear-to-send	High
	EXTCLK	Input	Use as alternative UART clock	-
UARTO	RTSN	Output	UART request-to-send	High
	TXD	Output	UART transmit data	High
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-

* see GRLIB IP Library User's Manual

14.8 Signal definitions and reset values

The signals and their reset values are described in table 96.

Table 96. Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
dsutx	Output	UART transmit data line	-	Logical 1
dsurx	Input	UART receive data line	-	-

14.9 Timing

The timing waveforms and timing parameters are shown in figure 19 and are defined in table 97.

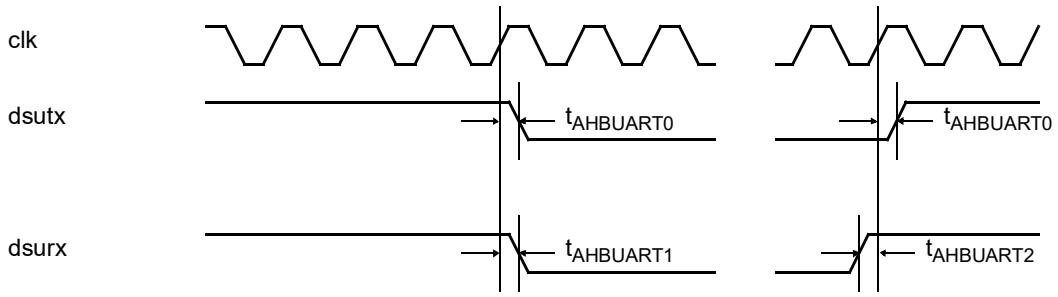


Figure 19. **Timing waveforms**

Table 97. Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{AHBUART0}$	clock to output delay	rising clk edge	TBD	TBD	ns
$t_{AHBUART1}$	input to clock hold	rising clk edge	-	-	ns
$t_{AHBUART2}$	input to clock setup	rising clk edge	-	-	ns

Note: The dsurx input is re-synchronized internally. The signal does not have to meet any setup or hold requirements.

14.10 Library dependencies

Table 98 shows libraries used when instantiating the core (VHDL libraries).

Table 98. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	UART	Signals, component	Signals and component declaration

14.11 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.uart.all;

entity ahbuart_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- UART signals
    ahbrxd   : in  std_ulogic;
    ahbtxd   : out std_ulogic
    );
end;

architecture rtl of ahbuart_ex is
```

```
  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- UART signals
  signal ahbuarti : uart_in_type;
  signal ahbuarto : uart_out_type;

begin

  -- AMBA Components are instantiated here
  ...


  -- AHB UART
  ahbuart0 : ahbuart
  generic map (hindex => 5, pindex => 7, paddr => 7)
  port map (rstn, clk, ahbuarti, ahbuarto, apbi, apbo(7), ahbmi, ahbmo(5));

  -- AHB UART input data
  ahbuarti.rxd <= ahbrxd;

  -- connect AHB UART output to entity output signal
  ahbtxd <= ahbuarto.txd;

end;
```

15 AMBAMON - AMBA Bus Monitor

15.1 Overview

The AMBA bus monitor checks the AHB and APB buses for violations against a set of rules. When an error is detected a signal is asserted and error message is (optionally) printed.

15.2 Rules

This section lists all rules checked by the AMBA monitor. The rules are divided into four different tables depending on which type of device they apply to.

Some requirements of the AMBA specification are not adopted by the GRLIB implementation (on a system level). These requirements are listed in the table below.

Table 99. Requirements not checked in GRLIB

Rule Number	Description	References
1	A slave which issues RETRY must only be accessed by one master at a time.	AMBA Spec. Rev 2.0 3-38.

Table 100. AHB master rules.

Rule Number	Description	References
1	Busy can only occur in the middle of bursts. That is only after a NON-SEQ, SEQ or BUSY.	AMBA Spec. Rev 2.0 3-9. http://www.arm.com/support/faqip/492.html
2	Busy can only occur in the middle of bursts. It can be the last access of a burst but only for INCR bursts.	AMBA Spec. Rev 2.0 3-9. http://www.arm.com/support/faqip/492.html
3	The address and control signals must reflect the next transfer in the burst during busy cycles.	AMBA Spec. Rev 2.0 3-9.
4	The first transfer of a single access or a burst must be NONSEQ (this is ensured together with rule 1).	AMBA Spec. Rev 2.0 3-9.
5	HSIZE must never be larger than the bus width.	AMBA Spec. Rev 2.0 3-43.
6	HADDR must be aligned to the transfer size.	AMBA Spec. Rev 2.0 3-12, 3-25. http://www.arm.com/support/faqip/582.html
7	Address and controls signals can only change when hready is low if the previous HTRANS value was IDLE, BUSY or if an ERROR, SPLIT or RETRY response is given.	http://www.arm.com/support/faqip/487.html http://www.arm.com/support/faqip/579.html
8	Address and control signals cannot change between consecutive BUSY cycles.	AMBA Spec. Rev 2.0 3-9.
9	Address must be related to the previous access according to HBURST and HSIZE and control signals must be identical for SEQUENTIAL accesses.	AMBA Spec. Rev 2.0 3-9.
10	Master must cancel the following transfer when receiving an RETRY response.	AMBA Spec. Rev 2.0 3-22.
11	Master must cancel the following transfer when receiving an SPLIT response.	AMBA Spec. Rev 2.0 3-22.

Table 100. AHB master rules.

Rule Number	Description	References
12	Master must reattempt the transfer which received a RETRY response.	AMBA Spec. Rev 2.0 3-21. http://www.arm.com/support/faqip/603.html.
13	Master must reattempt the transfer which received a SPLIT response.	AMBA Spec. Rev 2.0 3-21. http://www.arm.com/support/faqip/603.html.
14	Master can optionally cancel the following transfer when receiving an ERROR response. Only a warning is given if assertions are enabled if it does not cancel the following transfer.	AMBA Spec. Rev 2.0 3-23.
15	Master must hold HWDATA stable for the whole data phase when wait states are inserted. Only the appropriate byte lanes need to be driven for subword transfers.	AMBA Spec. Rev 2.0 3-7. AMBA Spec. Rev 2.0 3-25.
16	Bursts must not cross a 1 kB address boundary.	AMBA Spec. Rev 2.0 3-11.
17	HMASTLOCK indicates that the current transfer is part of a locked sequence. It must have the same timing as address/control.	AMBA Spec. Rev 2.0 3-28.
18	HLOCK must be asserted at least one clock cycle before the address phase to which it refers.	AMBA Spec. Rev 2.0 3-28.
19	HLOCK must be asserted for the duration of a burst and can only be deasserted so that HMASTLOCK is deasserted after the final address phase.	http://www.arm.com/support/faqip/597.html
20	HLOCK must be deasserted in the last address phase of a burst.	http://www.arm.com/support/faqip/588.html
21	HTRANS must be driven to IDLE during reset.	http://www.arm.com/support/faqip/495.html
22	HTRANS can only change from IDLE to NONSEQ or stay IDLE when HREADY is deasserted.	http://www.arm.com/support/faqip/579.html

Table 101. AHB slave rules.

Rule Number	Description	References
1	AHB slave must respond with a zero wait state OKAY response to BUSY cycles in the same way as for IDLE.	AMBA Spec. Rev 2.0 3-9.
2	AHB slave must respond with a zero wait state OKAY response to IDLE.	AMBA Spec. Rev 2.0 3-9.
3	HRESP should be set to ERROR, SPLIT or RETRY only one cycle before HREADY is driven high.	AMBA Spec. Rev 2.0 3-22.
4	Two-cycle ERROR response must be given.	AMBA Spec. Rev 2.0 3-22.
5	Two-cycle SPLIT response must be given.	AMBA Spec. Rev 2.0 3-22.
6	Two-cycle RETRY response must be given.	AMBA Spec. Rev 2.0 3-22.
7	SPLIT complete signalled to master which did not have pending access.	AMBA Spec. Rev 2.0 3-36.
8	Split complete must not be signalled during same cycle as SPLIT.	http://www.arm.com/support/faqip/616.html
9	It is recommended that slaves drive HREADY high and HRESP to OKAY when not selected. A warning will be given if this is not followed.	http://www.arm.com/support/faqip/476.html

Table 101. AHB slave rules.

Rule Number	Description	References
10	It is recommended that slaves do not insert more than 16 wait states. If this is violated a warning will be given if assertions are enabled.	AMBA Spec. Rev 2.0 3-20.
11	Slaves should not assert the HSPLIT (Split complete) signal for more than one cycle for each SPLIT response. If a slave asserts HSPLIT for more than one cycle it will not cause the system to malfunction. It can however be a indication that a core does not perform as expected. Therefore assertion of HSPLIT during more than one cycle for a SPLIT response is reported as a warning.	No reference

Table 102. APB slave rules.

Rule Number	Description	References
1	The bus must move to the SETUP state or remain in the IDLE state when in the IDLE state.	AMBA Spec. Rev 2.0 5-4.
2	The bus must move from SETUP to ENABLE in one cycle.	AMBA Spec. Rev 2.0 5-4.
3	The bus must move from ENABLE to SETUP or IDLE in one cycle.	AMBA Spec. Rev 2.0 5-5.
4	The bus must never be in another state than IDLE, SETUP, ENABLE.	AMBA Spec. Rev 2.0 5-4.
5	PADDR must be stable during transition from SETUP to ENABLE.	AMBA Spec. Rev 2.0 5-5.
6	PWRITE must be stable during transition from SETUP to ENABLE.	AMBA Spec. Rev 2.0 5-5.
7	PWDATA must be stable during transition from SETUP to ENABLE.	AMBA Spec. Rev 2.0 5-5.
8	Only one PSEL must be enabled at a time.	AMBA Spec. Rev 2.0 5-4.
9	PSEL must be stable during transition from SETUP to ENABLE.	AMBA Spec. Rev 2.0 5-5.

Table 103. Arbiter rules

Rule Number	Description	References
1	HreadyIn to slaves and master must be driven by the currently selected device.	http://www.arm.com/support/faqip/482.html
2	A master which received a SPLIT response must not be granted the bus until the slave has set the corresponding HSPLIT line.	AMBA Spec. Rev 2.0 3-35.
3	The dummy master must be selected when a SPLIT response is received for a locked transfer.	http://www.arm.com/support/faqip/14307.html

15.3 Configuration options

Table 104 shows the configuration options of the core (VHDL generics).

***Table 104.*Configuration options**

Generic	Function	Allowed range	Default
asserterr	Enable assertions for AMBA requirements. Violations are asserted with severity error.	0 - 1	1
assertwarn	Enable assertions for AMBA recommendations. Violations are asserted with severity warning.	0 - 1	1
hmstdisable	Disable AHB master rule check. To disable a master rule check a value is assigned so that the binary representation contains a one at the position corresponding to the rule number, e.g 0x80 disables rule 7.	-	0
hslvdisable	Disable AHB slave tests. Values are assigned as for hmstdisable.	-	0
pslvdisable	Disable APB slave tests. Values are assigned as for hmstdisable.	-	0
arbdisable	Disable Arbiter tests. Values are assigned as for hmstdisable.	-	0
nahbm	Number of AHB masters in the system.	0 - NAHBMST	NAHBMST
nahbs	Number of AHB slaves in the system.	0 - NAHBSLV	NAHBSLV
napb	Number of APB slaves in the system.	0 - NAPBSLV	NAPBSLV
ebterm	Relax rule checks to allow use in systems with early burst termination. This generic should be set to 0 for systems that use GRLIB's AHBCTRL core.	0 - 1	0

15.4 Signal descriptions

Table 105 shows the interface signals of the core (VHDL ports).

*Table 105.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AHB reset	Low
CLK	N/A	Input	AHB clock	-
AHBMI	*	Input	AHB master interface input record	-
AHBMO	*	Input	AHB master interface output record array	-
AHBSI	*	Input	AHB slave interface input record	-
AHBSO	*	Input	AHB slave interface output record array	-
APBI	*	Input	APB slave interface input record	
APBO	*	Input	APB slave interface output record array	
ERR	N/A	Output	Error signal (error detected)	High

* see GRLIB IP Library User's Manual

15.5 Library dependencies

Table 106 shows libraries used when instantiating the core (VHDL libraries).

*Table 106.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions
GAISLER	SIM	Component	Component declaration

15.6 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.sim.all;

entity ambamon_ex is
  port (
    clk : in std_ulogic;
    rst : in std_ulogic
end;

architecture rtl of ambamon_ex is
-- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);


  -- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);


begin
  -- AMBA Components are instantiated here
   ...
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.sim.all;

entity ambamon_ex is
  port (
    clk : in  std_ulogic;
    rst : in  std_ulogic;
    err : out std_ulogic
end;

architecture rtl of ambamon_ex is
  -- AHB signals
  signal ahbmi  : ahb_mst_in_type;
  signal ahbmo  : ahb_mst_out_vector := (others => apb_none);

  -- AHB signals
  signal ahbsi  : ahb_slv_in_type;
  signal ahbso  : ahb_slv_out_vector := (others => apb_none);

  -- APB signals
```

```
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);


begin

  mon0 : ambamon
  generic map(
    assert_err => 1,
    assert_war => 0,
    nahbm      => 2,
    nahbs      => 2,
    napb       => 1
  )
  port map(
    rst        => rst,
    clk        => clk,
    ahbmi      => ahbmi,
    ahbmo      => ahbmo,
    ahbsi      => ahbsi,
    ahbso      => ahbso,
    apbi       => apbi,
    apbo       => apbo,
    err        => err);

end;
```

16 APBCTRL - AMBA AHB/APB bridge with plug&play support

16.1 Overview

The AMBA AHB/APB bridge is a APB bus master according the AMBA 2.0 standard.

The controller supports up to 16 slaves. The actual maximum number of slaves is defined in the GRLIB.AMBA package, in the VHDL constant NAPBSLV. The number of slaves can also be set using the *nslaves* VHDL generic.

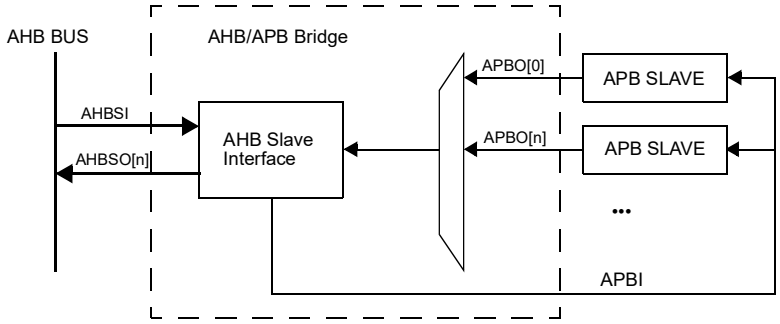


Figure 20. AHB/APB bridge block diagram

16.2 Operation

16.2.1 Decoding

Decoding (generation of PSEL) of APB slaves is done using the plug&play method explained in the GRLIB IP Library User's Manual. A slave can occupy any binary aligned address space with a size of 256 bytes - 1 Mbyte. Writes to unassigned areas will be ignored, while reads from unassigned areas will return an arbitrary value. AHB error response will never be generated.

16.2.2 Plug&play information

GRLIB APB slaves contain two plug&play information words which are included in the APB records they drive on the bus (see the GRLIB IP Library User's Manual for more information). These records are combined into an array which is connected to the APB bridge.

The plug&play information is mapped on a read-only address area at the top 4 kbytes of the bridge address space. Each plug&play block occupies 8 bytes. The address of the plug&play information for a certain unit is defined by its bus index. If the bridge is mapped on AHB address 0x80000000, the address for the plug&play records is thus 0x800FF000 + n*8.

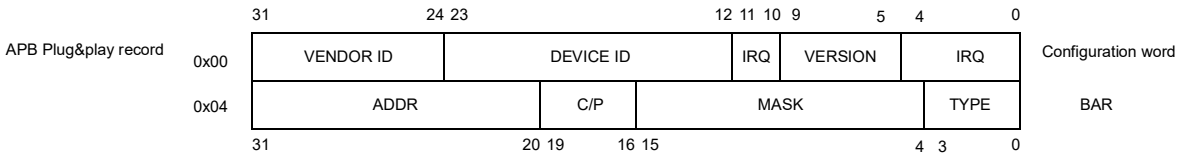


Figure 21. APB plug&play information

16.3 APB bus monitor

An APB bus monitor is integrated into the core. It is enabled with the enbusmon generic. It has the same functionality as the APB parts in the AMBA monitor core (AMBAMON). For more information on which rules are checked se the AMBAMON documentation.

16.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x006. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

16.5 Implementation

16.5.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers if the GRLIB config package setting *grlib_async_reset_enable* is set.

16.5.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual).

16.6 Configuration options

Table 107 shows the configuration options of the core (VHDL generics).

*Table 107.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	0 - NAHBSLV-1	0
haddr	The MSB address of the AHB area. Sets the 12 most significant bits in the 32-bit AHB address.	0 - 16#FFF#	0
hmask	The AHB area address mask. Sets the size of the AHB area and the start address together with haddr.	0 - 16#FFF#	16#FFF#
nslaves	The maximum number of slaves	1 - NAPBSLV	NAPBSLV
debug	Print debug information during simulation	0 - 2	2
icheck	Enable bus index checking (PINDEX)	0 - 1	1
enbusmon	Enable APB bus monitor	0 - 1	0
asserterr	Enable assertions for AMBA requirements. Violations are asserted with severity error.	0 - 1	0
assertwarn	Enable assertions for AMBA recommendations. Violations are asserted with severity warning.	0 - 1	0
pslvdisable	Disable APB slave rule check. To disable a slave rule check a value is assigned so that the binary representation contains a one at the position corresponding to the rule number, e.g 0x80 disables rule 7.	N/A	0
mcheck	Check if there are any intersections between APB slave memory areas. If two areas intersect an assert with level failure will be triggered (in simulation).	0 - 1	1
ccheck	Perform sanity checks on PnP configuration records (in simulation).	0 - 1	1

16.7 Signal descriptions

Table 108 shows the interface signals of the core (VHDL ports).

*Table 108.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AHB reset	Low
CLK	N/A	Input	AHB clock	-
AHBI	*	Input	AHB slave input	-
AHBO	*	Output	AHB slave output	-
APBI	*	Output	APB slave inputs	-
APBO	*	Input	APB slave outputs	-

* see GRLIB IP Library User's Manual

16.8 Library dependencies

Table 109 shows libraries used when instantiating the core (VHDL libraries).

*Table 109.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions

16.9 Component declaration

```
library grlib;
use grlib.amba.all;

component apbctrl
  generic (
    hindex  : integer := 0;
    haddr   : integer := 0;
    hmask   : integer := 16#fff#;
    nslaves : integer range 1 to NAPBSLV := NAPBSLV;
    debug   : integer range 0 to 2 := 2;   -- print config to console
    icheck  : integer range 0 to 1 := 1
  );
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    ahbi    : in  ahb_slv_in_type;
    ahbo    : out ahb_slv_out_type;
    apbi    : out apb_slv_in_type;
    apbo    : in  apb_slv_out_vector
  );
  end component;
```

16.10 Instantiation

This example shows how an APB bridge can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use work.debug.all;

.
.

  -- AMBA signals

  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);

signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector := (others => apb_none);


begin


-- APB bridge

apb0 : apbctrl-- AHB/APB bridge
  generic map (hindex => 1, haddr => CFG_APBADDR)
  port map (rstn, clk, ahbsi, ahbso(1), apbi, apbo );


-- APB slaves

uart1 : apbuart
    generic map (pindex => 1, paddr => 1,  pirq => 2)
    port map (rstn, clk, apbi, apbo(1), u1i, u1o);

irqctrl0 : irqmp
    generic map (pindex => 2, paddr => 2)
    port map (rstn, clk, apbi, apbo(2), irqo, irqi);
```

```
...
end;
```

16.11 Debug print-out

The APB bridge can print-out the plug-play information from the attached during simulation. This is enabled by setting the debug VHDL generic to 2. Reporting starts by scanning the array from 0 to NAPBSLV - 1 (defined in the grlib.amba package). It checks each entry in the array for a valid vendor-id (all nonzero ids are considered valid) and if one is found, it also retrieves the device-id. The description for these ids are obtained from the GRLIB.DEVICES package, and is printed on standard out together with the slave number. If the index check is enabled (done with a VHDL generic), the report module also checks if the pindex number returned in the record matches the array number of the record currently checked (the array index). If they do not match, the simulation is aborted and an error message is printed.

The address range and memory type is also checked and printed. The address information includes type, address and mask. The address ranges currently defined are AHB memory, AHB I/O and APB I/O. All APB devices are in the APB I/O range so the type does not have to be checked. From this information, the report module calculates the start address of the device and the size of the range. The information finally printed is start address and size.

17 APBPS2 - PS/2 host controller with APB interface

17.1 Introduction

The PS/2 interface is a bidirectional synchronous serial bus primarily used for keyboard and mouse communications. The APBPS2 core implements the PS2 protocol with a APB back-end. Figure 22 shows a model of APBPS2 and the electrical interface.

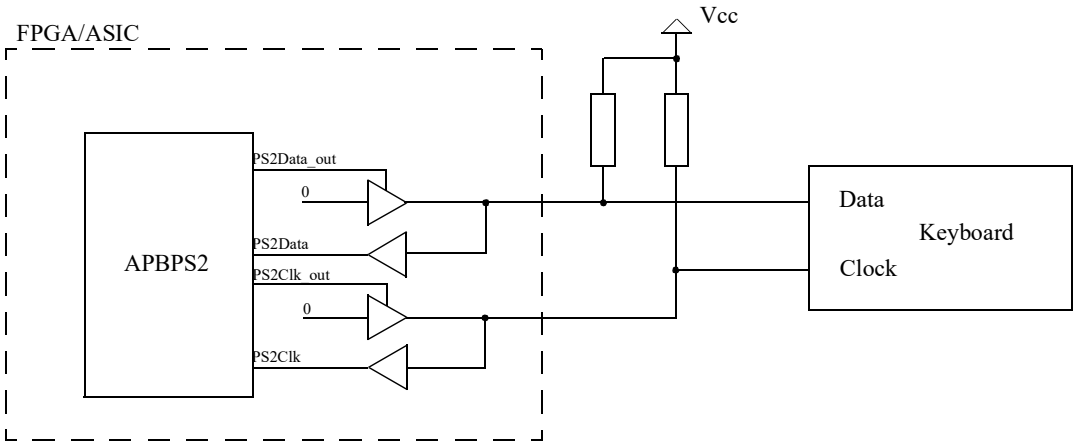


Figure 22. APBPS2 electrical interface

PS/2 data is sent in 11 bits frames. The first bit is a start bit followed by eight data bits, one odd parity bit and finally one stop bit. Figure 23 shows a typical PS/2 data frame.

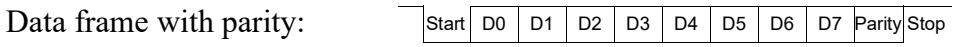


Figure 23. PS/2 data frame

17.2 Receiver operation

The receiver of APBPS2 receives the data from the keyboard or mouse, and converts it to 8-bit data frames to be read out via the APB bus. It is enabled through the receiver enable (RE) bit in the PS/2 control register. If a parity error or framing error occurs, the data frame will be discarded. Correctly received data will be transferred to a 16 byte FIFO. The data ready (DR) bit in the PS/2 status register will be set, and retained as long as the FIFO contains at least one data frame. When the FIFO is full, the receiver buffer full (RF) bit in the status register is set. The keyboard will be inhibited and buffer data until the FIFO gets read again. Interrupt is sent when a correct stop bit is received then it's up to the software to handle any resend operations if the parity bit is wrong. Figure 24 shows a flow chart for the operations of the receiver state machine.

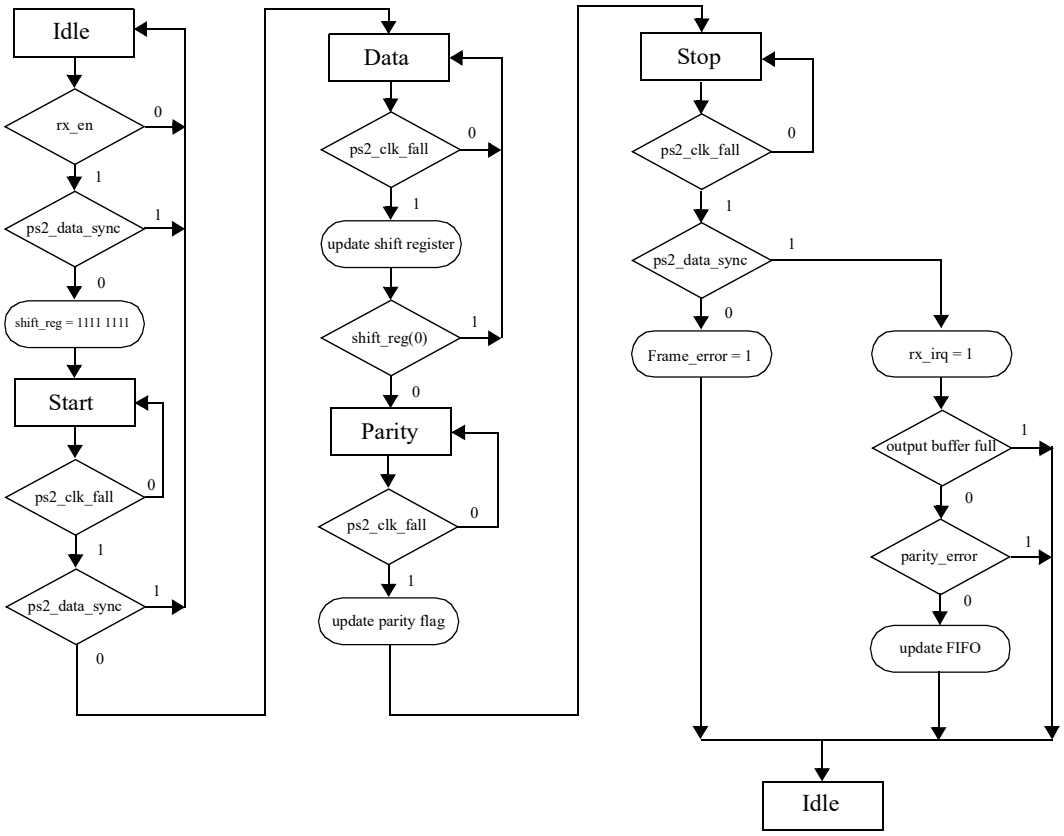


Figure 24. Flow chart for the receiver state machine

17.3 Transmitter operations

The transmitter part of APBPS2 is enabled for through the transmitter enable (TE) bit in the PS/2 control register. The PS/2 interface has a 16 byte transmission FIFO that stores commands sent by the CPU. Commands are used to set the LEDs on the keyboard, and the typematic rate and delay. Typematic rate is the repeat rate of a key that is held down, while the delay controls for how long a key has to be held down before it begins automatically repeating. Typematic repeat rates, delays and possible other commands are listed in table 121.

If the TE bit is set and the transmission FIFO is not empty a transmission of the command will start. The host will pull the clock line low for at least 100 us and then transmit a start bit, the eight bit command, an odd parity bit, a stop bit and wait for an acknowledgement bit by the device. When this happens an interrupt is generated. Figure 25 shows the flow chart for the transmission state machine.

17.4 Clock generation

A PS/2 interface should generate a clock of 10.0 - 16.7 kHz. To transmit data, a PS/2 host must inhibit communication by pulling the clock low for at least 100 microseconds. To do this, APBPS2 divides the APB clock with either a fixed or programmable division factor. The divider consist of a 17-bit down-counter and can divide the APB clock with a factor of 1 - 131071. The division rate, and the reset value of the timer reload register, is set to the *fKHz* generic divided by 10 in order to generate the 100 microsecond clock low time. If the VHDL generic *fixed* is 0, the division rate can be programmed through the timer reload register and should be programmed with the system frequency in kHz divided by ten. The reset value of the reload register is always set to the *fKHz* value divided by ten. However, the register will not be readable via the APB interface unless the *fixed* VHDL generic has been set to 0.

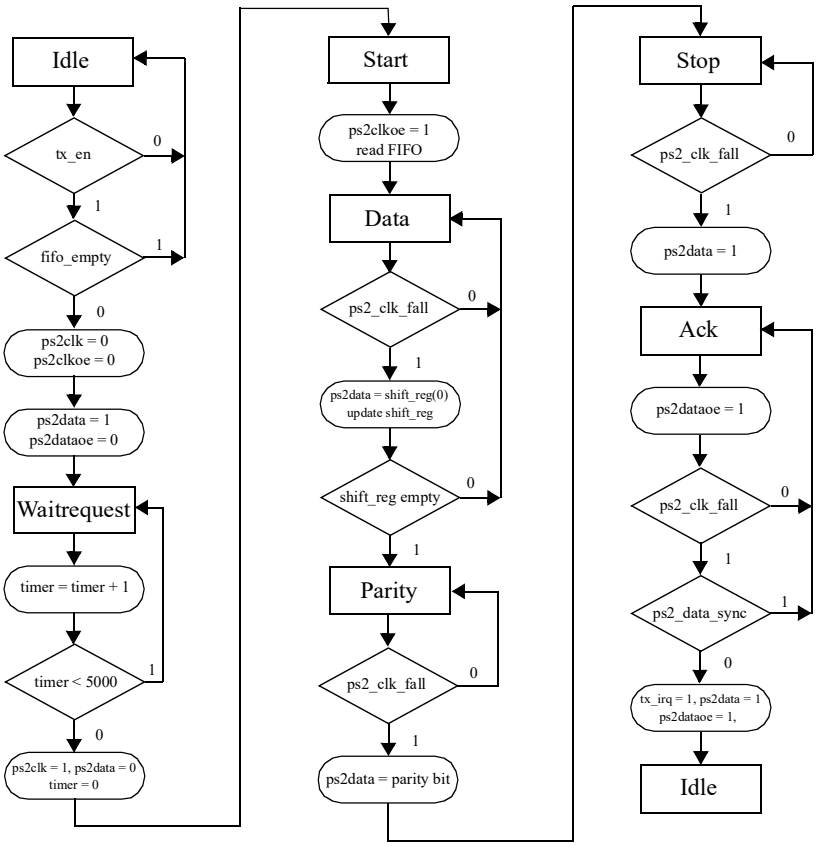


Figure 25. Flow chart for the transmitter state machine

17.5 Registers

The core is controlled through registers mapped into APB address space.

*Table 110.*APB PS/2 registers

APB address offset	Register
0x00	PS/2 Data register
0x04	PS/2 Status register
0x08	PS/2 Control register
0x0C	PS/2 Timer reload register

17.5.1 PS/2 Data Register

*Table 111.*0x00 - DATA - PS/2 data register

31 … 8	7 … 0
RESERVED	DATA
0	NR
r	rw

7: 0 Receiver holding FIFO (read access) and Transmitter holding FIFO (write access). If the receiver FIFO is not empty, read accesses retrieve the next byte from the FIFO. Bytes written to this field are stored in the transmitter holding FIFO if it is not full.

17.5.2 PS/2 Status Register

*Table 112.*0x04 - STAT - PS/2 status register

31 27	26 22	6	5	4	3	2	1	0
RCNT	TCNT	RESERVED	TF	RF	KI	FE	PE	DR
0	0	0	0	0	0	0	0	0
r	r	r	r	r	rw	rw	rw	r

0: Data ready (DR) - indicates that new data is available in the receiver holding register (read only).

1: Parity error (PE) - indicates that a parity error was detected

2: Framing error (FE) - indicates that a framing error was detected.

3: Keyboard inhibit (KI) - indicates that the keyboard is inhibited.

4: Receiver buffer full (RF) - indicates that the output buffer (FIFO) is full (read only).

5: Transmitter buffer full (TF) - indicates that the input buffer (FIFO) is full (read only).

26: 22 Transmit FIFO count (TCNT) - shows the number of data frames in the transmit FIFO (read only).

31: 27 Receiver FIFO count (RCNT) - shows the number of data frames in the receiver FIFO (read only).

17.5.3 PS/2 Control Register

*Table 113.*0x08 - CTRL - PS/2 control register

31	4	3	2	1	0
RESERVED		TI	RI	TE	RE
0		0	0	0	0
r		rw	rw	rw	rw

0: Receiver enable (RE) - if set, enables the receiver.

1: Transmitter enable (TE) - if set, enables the transmitter.

2: Keyboard interrupt enable (RI) - if set, interrupts are generated when a frame is received.

3: Host interrupt enable (TI) - if set, interrupts are generated when a frame is transmitted.

17.5.4 PS/2 Timer Reload Register

*Table 114.*0x0C - TIMER - PS/2 reload register

31 17	16 0
RESERVED	TIMER RELOAD REG
0	*
r	rw*

16: 0 PS/2 timer reload register - Reset value determined by fktlz VHDL generic. Register only present it “fixed” VHDL generic is zero.

17.6 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x060. For a description of vendor and device identifiers see GRLIB IP Library User’s Manual.

17.7 Implementation

17.7.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User’s Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

17.7.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual).

17.8 Configuration options

Table 115 shows the configuration options of the core (VHDL generics).

*Table 115.*Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Index of the interrupt line.	0 - NAHBIRQ-1	0
fKHz	Frequency of APB clock in KHz. This value divided by 10 is the reset value of the timer reload register.	1 - 1310710	50000
fixed	Used fixed clock divider to generate PS/2 clock.	0 - 1	0
oepol	Output enable polarity	0 - 1	0

17.9 Signal descriptions

Table 116 shows the interface signals of the core (VHDL ports).

*Table 116.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
PS2I	PS2_CLK_I	Input	PS/2 clock input	-
	PS2_DATA_I	Input	PS/2 data input	-
PS2O	PS2_CLK_O	Output	PS/2 clock output	-
	PS2_CLK_OE	Output	PS/2 clock output enable	Low
	PS2_DATA_O	Output	PS/2 data output	-
	PS2_DATA_OE	Output	PS/2 data output enable	Low

* see GRLIB IP Library User's Manual

17.10 Library dependencies

Table 117 shows libraries used when instantiating the core (VHDL libraries).

*Table 117.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	APB signal definitions
GAISLER	MISC	Signals, component	PS/2 signal and component declaration

17.11 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.gencomp.all;

library gaisler;
use gaisler.misc.all;

entity apbps2_ex is
 port (
    rstn : in std_ulogic;
    clk : in std_ulogic;

    -- PS/2 signals
    ps2clk : inout std_ulogic;
    ps2data : inout std_ulogic
    );
end;

architecture rtl of apbuart_ex is

  -- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- PS/2 signals
  signal kbdi : ps2_in_type;
  signal kbdo : ps2_out_type;

begin

ps20 : apbps2 generic map(pindex => 5, paddr => 5, pirq => 4)
      port map(rstn, clkm, apbi, apbo(5), kbdi, kbdo);

kbdclk_pad : iopad generic map (tech => padtech)
      port map (ps2clk,kbdo.ps2_clk_o, kbdo.ps2_clk_oe, kbdi.ps2_clk_i);

kbdata_pad : iopad generic map (tech => padtech)
        port map (ps2data, kbdo.ps2_data_o, kbdo.ps2_data_oe, kbdi.ps2_data_i);

end;
```

17.12 Keboard scan codes

*Table 118.*Scan code set 2, 104-key keyboard

KEY	MAKE	BREAK	--	KEY	MAKE	BREAK	--	KEY	MAKE	BREAK
A	1C	F0,1C		9	46	F0,46		[	54	FO,54
B	32	F0,32			`0E	F0,0E		INSERT	E0,70	E0,F0,70
C	21	F0,21		-	4E	F0,4E		HOME	E0,6C	E0,F0,6C
D	23	F0,23		=	55	FO,55		PG UP	E0,7D	E0,F0,7D
E	24	F0,24		\	5D	F0,5D		DELETE	E0,71	E0,F0,71
F	2B	F0,2B		BKSP	66	F0,66		END	E0,69	E0,F0,69
G	34	F0,34		SPACE	29	F0,29		PG DN	E0,7A	E0,F0,7A
H	33	F0,33		TAB	0D	F0,0D		U ARROW	E0,75	E0,F0,75
I	43	F0,43		CAPS	58	F0,58		L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B		L SHFT	12	FO,12		D ARROW	E0,72	E0,F0,72
K	42	F0,42		L CTRL	14	FO,14		R ARROW	E0,74	E0,F0,74
L	4B	F0,4B		L GUI	E0,1F	E0,F0,1F		NUM	77	F0,77
M	3A	F0,3A		L ALT	11	F0,11		KP /	E0,4A	E0,F0,4A
N	31	F0,31		R SHFT	59	F0,59		KP *	7C	F0,7C
O	44	F0,44		R CTRL	E0,14	E0,F0,14		KP -	7B	F0,7B
P	4D	F0,4D		R GUI	E0,27	E0,F0,27		KP +	79	F0,79
Q	15	F0,15		R ALT	E0,11	E0,F0,11		KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D		APPS	E0,2F	E0,F0,2F		KP .	71	F0,71
S	1B	F0,1B		ENTER	5A	F0,5A		KP 0	70	F0,70
T	2C	F0,2C		ESC	76	F0,76		KP 1	69	F0,69
U	3C	F0,3C		F1	5	F0,05		KP 2	72	F0,72
V	2A	F0,2A		F2	6	F0,06		KP 3	7A	F0,7A
W	1D	F0,1D		F3	4	F0,04		KP 4	6B	F0,6B
X	22	F0,22		F4	0C	F0,0C		KP 5	73	F0,73
Y	35	F0,35		F5	3	F0,03		KP 6	74	F0,74
Z	1A	F0,1A		F6	0B	F0,0B		KP 7	6C	F0,6C
0	45	F0,45		F7	83	F0,83		KP 8	75	F0,75
1	16	F0,16		F8	0A	F0,0A		KP 9	7D	F0,7D
2	1E	F0,1E		F9	1	F0,01		]	5B	F0,5B
3	26	F0,26		F10	9	F0,09		;	4C	F0,4C
4	25	F0,25		F11	78	F0,78			52	F0,52
5	2E	F0,2E		F12	7	F0,07		,	41	F0,41
6	36	F0,36		PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12		.	49	F0,49
7	3D	F0,3D		SCROLL	7E	F0,7E		/	4A	F0,4A
8	3E	F0,3E		PAUSE	E1,14,77, E1,F0,14, F0,77	-NONE-				

Table 119. Windows multimedia scan codes

KEY	MAKE	BREAK
Next Track	E0, 4D	E0, F0, 4D
Previous Track	E0, 15	E0, F0, 15
Stop	E0, 3B	E0, F0, 3B
Play/Pause	E0, 34	E0, F0, 34
Mute	E0, 23	E0, F0, 23
Volume Up	E0, 32	E0, F0, 32
Volume Down	E0, 21	E0, F0, 21
Media Select	E0, 50	E0, F0, 50
E-Mail	E0, 48	E0, F0, 48
Calculator	E0, 2B	E0, F0, 2B
My Computer	E0, 40	E0, F0, 40
WWW Search	E0, 10	E0, F0, 10
WWW Home	E0, 3A	E0, F0, 3A
WWW Back	E0, 38	E0, F0, 38
WWW Forward	E0, 30	E0, F0, 30
WWW Stop	E0, 28	E0, F0, 28
WWW Refresh	E0, 20	E0, F0, 20
WWW Favor-ites	E0, 18	E0, F0, 18

Table 120. ACPI scan codes (Advanced Configuration and Power Interface)

KEY	MAKE	BREAK
Power	E0, 37	E0, F0, 37
Sleep	E0, 3F	E0, F0, 3F
Wake	E0, 5E	E0, F0, 5E

17.13 Keyboard commands

*Table 121.*Transmit commands:

Command	Description
0xED	Set status LED's - keyboard will reply with ACK (0xFA). The host follows this command with an argument byte*
0xEE	Echo command - expects an echo response
0xF0	Set scan code set - keyboard will reply with ACK (0xFA) and wait for another byte. 0x01-0x03 which determines the scan code set to use. 0x00 returns the current set.
0xF2	Read ID - the keyboard responds by sending a two byte device ID of 0xAB 0x83
0xF3	Set typematic repeat rate - keyboard will reply with ACK (0xFA) and wait for another byte which determines the typematic rate.
0xF4	Keyboard enable - clears the keyboards output buffer, enables keyboard scanning and returns an acknowledgement.
0xF5	Keyboard disable - resets the keyboard, disables keyboard scanning and returns an acknowledgement.
0xF6	Set default - load default typematic rate/delay (10.9cps/500ms) and scan code set 2
0xFE	Resend - upon receipt of the resend command the keyboard will retransmit the last byte
0xFF	Reset - resets the keyboard

* bit 0 controls the scroll lock, bit 1 the num lock, bit 2 the caps lock, bit 3-7 are ignored

*Table 122.*Receive commands:

Command	Description
0xFA	Acknowledge
0xAA	Power on self test passed (BAT completed)
0xEE	Echo respond
0xFE	Resend - upon receipt of the resend command the host should retransmit the last byte
0x00	Error or buffer overflow
0xFF	Error of buffer overflow

*Table 123.*The typematic rate/delay argument byte

MSB							LSB
0	DELAY	DELAY	RATE	RATE	RATE	RATE	RATE

Table 124. Typematic repeat rates

Bits 0-4	Rate (cps)	Bits 0-4	Rate (cps)	Bits 0-4	Rate (cps)	Bits 0-4	Rate (cps)
00h	30	08h	15	10h	7.5	18h	3.7
01h	26.7	09h	13.3	11h	6.7	19h	3.3
02h	24	0Ah	12	12h	6	1Ah	3
03h	21.8	0Bh	10.9	13h	5.5	1Bh	2.7
04h	20.7	0Ch	10	14h	5	1Ch	2.5
05h	18.5	0Dh	9.2	15h	4.6	1Dh	2.3
06h	17.1	0Eh	8.6	16h	4.3	1Eh	2.1
07h	16	0Fh	8	17h	4	1Fh	2

Table 125. Typematic delays

Bits 5-6	Delay (seconds)
00b	0.25
01b	0.5
10b	0.75
11b	1

18 APBUART - AMBA APB UART Serial Interface

18.1 Overview

The interface is provided for serial communications. The UART supports data frames with 8 data bits, one optional parity bit and one or two stop bits. The core supports both reception and transmission of break signals using a configurable break size via dedicated APB registers. To generate the bit-rate, each UART has a programmable 12-bit clock divider. Two FIFOs are used for data transfer between the APB bus and UART, when *fifosize* VHDL generic > 1. Two holding registers are used for data transfer between the APB bus and UART, when *fifosize* VHDL generic = 1. Hardware flow-control is supported through the RTSN/CTSN hand-shake signals, when *flow* VHDL generic is set. Parity is supported for all VHDL generic configurations.

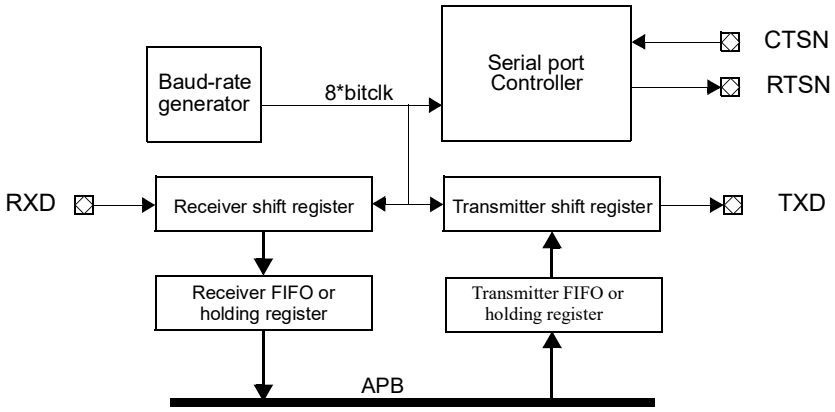


Figure 26. Block diagram

18.2 Operation

18.2.1 Transmitter operation

The transmitter is enabled through the TE bit in the UART control register. Data that is to be transferred is stored in the FIFO/holding register by writing to the data register. This FIFO is configurable to different sizes via the *fifosize* VHDL generic. When the size is 1, only a single holding register is used but in the following discussion both will be referred to as FIFOs. When ready to transmit, data is transferred from the transmitter FIFO/holding register to the transmitter shift register and converted to a serial stream on the transmitter serial output pin (TXD). It automatically sends a start bit followed by eight data bits, an optional parity bit, and one stop bit (figure 27). The least significant bit of the data is sent first. It is also possible to use two stop bits, this is configured via the control register.

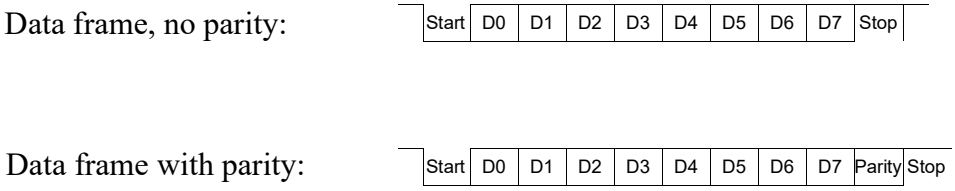


Figure 27. UART data frames

Following the transmission of the stop bit, if a new character is not available in the transmitter FIFO, the transmitter serial data output remains high and the transmitter shift register empty bit (TS) will be set in the UART status register. Transmission resumes and the TS is cleared when a new character is loaded into the transmitter FIFO. When the FIFO is empty the TE bit is set in the status register. If the transmitter is disabled, it will immediately stop any active transmissions including the character currently being shifted out from the transmitter shift register. The transmitter holding register may not be loaded when the transmitter is disabled or when the FIFO (or holding register) is full. If this is done, data might be overwritten and one or more frames are lost.

The discussion above applies to any FIFO configurations including the special case with a holding register (VHDL generic *fifosize* = 1). If FIFOs are used (VHDL generic *fifosize* > 1) some additional status and control bits are available. The TF status bit (not to be confused with the TF control bit) is set if the transmitter FIFO is currently full and the TH bit is set as long as the FIFO is *less* than half-full (less than half of entries in the FIFO contain data). The TF control bit enables FIFO interrupts when set. The status register also contains a counter (TCNT) showing the current number of data entries in the FIFO.

When flow control is enabled, the CTSN input must be low in order for the character to be transmitted. If it is deasserted in the middle of a transmission, the character in the shift register is transmitted and the transmitter serial output then remains inactive until CTSN is asserted again. If the CTSN is connected to a receivers RTSN, overrun can effectively be prevented.

18.2.2 Transmitter break process

A break signal is defined as a continuous ‘0’ bits on the TX line for a period of *breaksize* clock cycles followed by a single logical ‘1’ bit. A transmit break (TB) command via the UART control register together with the (TE) bit enabled, will issue a transmission of a break signal. The (TB) bit is self-clearing, meaning the user will need to issue a new transmit break command each time a break frame is desired. When a break is requested during active FIFO transmission, the UART completes the current word before initiating the break. Once the break character is sent, normal transmission from the FIFO automatically resumes from where it was interrupted.

The break size (BS) field in the UART control register allows users to configure the length of the break signal. This field explicitly sets the number of consecutive ‘0’ bits transmitted during a break transmission, not including the final ‘1’ that marks the end of the break.

18.2.3 Receiver operation

The serial input signal is first passed through a meta-stability filter. The first stage of this filter consists of two flip-flops connected in series that are clocked by the system clock. The output of the second flip-flop is then sampled into a three-bit shift register at a rate of 8 times the configured baudrate. Finally, the majority vote of the bits in the shift register is used as the serial data input for the rest of

the receiver. Loosely, this can be thought of as a low-pass filter with cut-off frequency of roughly 8/3 times the configured baudrate.

The receiver is enabled for data reception through the receiver enable (RE) bit in the UART control register. The receiver looks for a high to low transition of a start bit on the receiver serial data input. If a transition is detected, the state of the serial input is sampled a half bit clocks later. If the serial input is sampled high, the start bit is invalid and the search for a valid start bit continues. If the serial input is still low, a valid start bit is assumed and the receiver continues to sample the serial input at one bit time intervals (at the theoretical center of the bit) until the proper number of data bits and the optional parity bit have been assembled and one stop bit has been detected. If more than one stop bit is received then the all but the first are interpreted as the serial input being idle. In particular, the number of stop bits configured in the control register (via the NS bit) has no effect on receiver operation.

The receiver also has a configurable FIFO which is identical to the one in the transmitter. As mentioned in the transmitter part, both the holding register and FIFO will be referred to as FIFO.

During reception, the least significant bit is received first. The data is then transferred to the receiver FIFO and the data ready (DR) bit is set in the UART status register as soon as the FIFO contains at least one data frame. The parity, framing and overrun error bits are set at the received byte boundary, at the same time as the data ready bit would have been set. The data frame is not stored in the FIFO if an error is detected. Also, the new error status bits are or:ed with the old values before they are stored into the status register. Thus, they are not cleared until written to with zeros from the AMBA APB bus. If both the receiver FIFO and shift registers are full when a new start bit is detected, then the character held in the receiver shift register will be lost and the overrun bit will be set in the UART status register.

RTSN will be negated (high) when a valid start bit is detected and the receiver FIFO is full. When the holding register is read, the RTSN will automatically be reasserted again. This behavior applies regardless of the flow control configuration, i.e. even if the *flow* generic is set to 0 or the FL bit in the control register is set to 0.

When the VHDL generic *fifosize* > 1, which means that holding registers are not considered here, some additional status and control bits are available. The RF status bit (not to be confused with the RF control bit) is set when the receiver FIFO is full. The RH status bit is set when the receiver FIFO is half-full (at least half of the entries in the FIFO contain data frames). The RF control bit enables receiver FIFO interrupts when set. A RCNT field is also available showing the current number of data frames in the FIFO.

18.2.4 Receiver break process

The UART receiver is designed to detect break characters based on a configurable break size. When the receiver is enabled (RE), the receiver will begin the decoding process as for a normal character, and start searching for a break character when a logical ‘0’ is sampled instead of a logical ‘1’ at the stop bit position. It will continue until a sequence of consecutive ‘0’ bits that is equal to or larger than break size (BS) field in the UART control register is detected.

Upon detecting a valid break, a break received flag (BR) is indicated on the UART status register. The core will also disregard storing the corresponding zeros in the receiver FIFO to not be mixed with valid reception of 0x00 data.

The receiver’s parity setting is fully supported during break detection. The core adjusts its internal counters regardless of the parity configuration to correctly identify break frames according to the configured break size. A framing error will occur if the detected break is smaller than the selected break size in the UART control register. A break interrupt can be generated when a valid break character is detected when the break interrupt (BI) bit is set.

18.3 Baud-rate generation

Each UART contains a 12-bit down-counting scaler to generate the desired baud-rate, the number of scaler bits can be increased with VHDL generic *sbits*. The scaler is clocked by the system clock and generates a UART tick each time it underflows. It is reloaded with the value of the UART scaler reload register after each underflow. The resulting UART tick frequency should be 8 times the desired baud-rate. One appropriate formula to calculate the scaler value for a desired baud rate, using integer division where the remainder is discarded, is:

*scaler value = (system_clock_frequency) / (baud_rate * 8 + 7).*

To calculate the exact required scaler value use:

*scaler value = (system_clock_frequency) / (baud_rate * 8) - 1*

If the EC bit is set, the ticks will be generated with the same frequency as the external clock input instead of at the scaler underflow rate. In this case, the frequency of external clock must be less than half the frequency of the system clock.

18.4 Loop back mode

If the LB bit in the UART control register is set, the UART will be in loop back mode. In this mode, the transmitter output is internally connected to the receiver input and the RTSN is connected to the CTSN. It is then possible to perform loop back tests to verify operation of receiver, transmitter and associated software routines. In this mode, the outputs remain in the inactive state, in order to avoid sending out data.

18.5 FIFO debug mode

FIFO debug mode is entered by setting the debug mode bit in the control register. In this mode it is possible to read the transmitter FIFO and write the receiver FIFO through the FIFO debug register. The transmitter output is held inactive when in debug mode. A write to the receiver FIFO generates an interrupt if receiver interrupts are enabled.

FIFO debug mode requires that the hardware implementation supports flow control (VHDL generic *flow* = 1).

18.6 Interrupt generation

Interrupts are generated differently when a holding register is used (VHDL generic *fifosize* = 1) and when FIFOs are used (VHDL generic *fifosize* > 1). When holding registers are used, the UART will generate an interrupt under the following conditions: when the transmitter is enabled, the transmitter interrupt is enabled and the transmitter holding register moves from full to empty; when the receiver is enabled, the receiver interrupt is enabled and the receiver holding register moves from empty to full; when the receiver is enabled, the receiver interrupt is enabled and a character with either parity, framing or overrun error is received.

For FIFOs, two different kinds of interrupts are available: normal interrupts and FIFO interrupts. Normal interrupts from the transmitter are generated when transmitter interrupts are enabled (TI), the transmitter is enabled and the transmitter FIFO goes from containing data to being empty. For the receiver normal interrupts are generated when receiver interrupts are enabled (RI), the receiver is enabled and a character is received. The interrupt is generated if the character is correct and stored in the receive FIFO or if an error, such as parity; framing or overrun occurred.

Transmitter FIFO interrupts are generated when the transmitter FIFO interrupts are enabled (TF), transmissions are enabled (TE) and the UART is less than half-full (that is, whenever the TH status bit is set). This is a level interrupt and the interrupt signal is continuously driven high as long as the condition prevails. Receiver FIFO interrupts are generated when receiver FIFO interrupts are enabled

(RF), the receiver is enabled and the FIFO is half-full. The interrupt signal is continuously driven high as long as the receiver FIFO is half-full (at least half of the entries contain data frames).

Note that the processor acknowledges and clears the corresponding interrupt pending register but for FIFO interrupts the interrupt signal from the UART is continuously driven high, resulting in a new pending interrupt immediately being set in the interrupt controller. If FIFO interrupts are used for controlling FIFO handling, an interrupt handler need to check that there is room in the transmit FIFO before writing and that characters are available in the receive FIFO before reading.

To reduce interrupt occurrence a delayed receiver interrupt is available. It is enabled using the delayed interrupt enable (DI) bit. When enabled a timer is started each time a character is received and an interrupt is only generated if another character has not been received within 4 character + 4 bit times. If receiver FIFO interrupts are enabled a pending character interrupt will be cleared when the FIFO interrupt is active since the character causing the pending irq state is already in the FIFO and is noticed by the driver through the FIFO interrupt. In order to not take one additional interrupt (due to the interrupt signal being driven continuously high as described above), software should clear the corresponding pending bit in the interrupt controller after the FIFO has been emptied.

There is also a separate interrupt for break characters. When enabled an interrupt will always be generated immediately when a break character is received even when delayed receiver interrupts are enabled. When break interrupts are disabled no interrupt will be generated for break characters when delayed interrupts are enabled.

When delayed interrupts are disabled the behavior is the same for the break interrupt bit except that an interrupt will be generated for break characters if receiver interrupt enable is set even if break interrupt is disabled.

An interrupt can also be enabled for the transmitter shift register. When enabled the core will generate an interrupt each time the shift register goes from a non-empty to an empty state.

18.7 Registers

The core is controlled through registers mapped into APB address space.

Table 126. UART registers

APB address offset	Register
0x0	UART Data register
0x4	UART Status register
0x8	UART Control register
0xC	UART Scaler register
0x10	UART FIFO Debug register
0x14	UART FIFO Debug Control register
0x18	UART Capability register

18.7.1 UART Data Register

Table 127. 0x00 - DATA - UART data register

31 … 8	7 … 0
RESERVED	DATA
	NR
	rw

7: 0 Receiver holding register or FIFO (read access)

7: 0 Transmitter holding register or FIFO (write access)

18.7.2 UART Status Register

Table 128. 0x04 - STAT - UART status register

31 … 26	25 … 20	19 … 11	10	9	8	7	6	5	4	3	2	1	0
RCNT	TCNT	RESERVED	RF	TF	RH	TH	FE	PE	OV	BR	TE	TS	DR
0	0	0	0	0	0	0	0	0	0	0	1	1	0
r	r	r	r	r	r	r	rw	rw	rw	rw	r	r	r

31: 26 Receiver FIFO count (RCNT) - shows the number of data frames in the receiver FIFO. Reset: 0

25: 20 Transmitter FIFO count (TCNT) - shows the number of data frames in the transmitter FIFO. Reset: 0

10 Receiver FIFO full (RF) - indicates that the receiver FIFO is full. Reset: 0

9 Transmitter FIFO full (TF) - indicates that the transmitter FIFO is full. Reset: 0

8 Receiver FIFO half-full (RH) -indicates that at least half of the FIFO is holding data. Reset: 0

7 Transmitter FIFO half-full (TH) - indicates that the FIFO is less than half-full. Reset: 0

6 Framing error (FE) - indicates that a framing error was detected. Reset: 0

5 Parity error (PE) - indicates that a parity error was detected. Reset: 0

4 Overrun (OV) - indicates that one or more character have been lost due to overrun. Reset: 0

3 Break received (BR) - indicates that a BREAK has been received. Reset: 0

2 Transmitter FIFO empty (TE) - indicates that the transmitter FIFO is empty. Reset: 1

1 Transmitter shift register empty (TS) - indicates that the transmitter shift register is empty. Reset: 1

0 Data ready (DR) - indicates that new data is available in the receiver holding register. Reset: 0

18.7.3 UART Control Register

Table 129. 0x08 - CTRL - UART control register

31	30 21	20	19 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FA	RESERVED	TB	BS	NS	SI	DI	BI	DB	RF	TF	EC	LB	FL	PE	PS	TI	RI	TE	RE
0	0	0	0xA	NR	NR	NR	0	0	NR	NR	0	NR	0	NR	NR	NR	NR	0	0
r	r	rw	rw	rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31 FIFOs available (FA) - Set to 1 when receiver and transmitter FIFOs are available. When 0, only holding register are available.

30: 21 RESERVED

20 Transmit break (TB) - When set, and TE bit is enabled, the core will transmit a break signal with the size specified in the BS field. This bit is self-clearing once break operation completes.

19: 16 Break size (BS) - The value of this field determines the length in bits of a break transfer on the UART transmitter. It also indicates the expected break size for the UART receiver. The actual break size is calculated as BS+1, meaning possible break character sizes of 11-16 bits.

15 Number of stop bits (NS) - When set to '1' then two stop bits will be used, otherwise one stop bit will be used.

14 Transmitter shift register empty interrupt enable (SI) - When set, an interrupt will be generated when the transmitter shift register becomes empty. See section 18.6 for more details.

13 Delayed interrupt enable (DI) - When set, delayed receiver interrupts will be enabled and an interrupt will only be generated for received characters after a delay of 4 character times + 4 bits if no new character has been received during that interval. This is only applicable if receiver interrupt enable is set. See section 18.6 for more details.

12 Break interrupt enable (BI) - When set, an interrupt will be generated each time a break character is received. See section 18.6 for more details.

11 FIFO debug mode enable (DB) - read only, writable in the FIFO Debug Control register.

10 Receiver FIFO interrupt enable (RF) - when set, Receiver FIFO level interrupts are enabled.

9 Transmitter FIFO interrupt enable (TF) - when set, Transmitter FIFO level interrupts are enabled.

8 External Clock (EC) - if set, the UART scaler will be clocked by UARTI.EXTCLK.

7 Loop back (LB) - if set, loop back mode will be enabled.

6 Flow control (FL) - if set, enables flow control using CTS/RTS (when implemented).

5 Parity enable (PE) - if set, enables parity generation and checking.

4 Parity select (PS) - selects parity polarity (0 = even parity, 1 = odd parity) (when PE=1).

3 Transmitter interrupt enable (TI) - if set, interrupts are generated when characters are transmitted (see section 18.6 for details).

2 Receiver interrupt enable (RI) - if set, interrupts are generated when characters are received (see section 18.6 for details).

1 Transmitter enable (TE) - if set, enables the transmitter.

0 Receiver enable (RE) - if set, enables the receiver.

18.7.4 UART Scaler Register

Table 130. 0x0C - SCALER - UART scaler reload register

31 sbits	sbits-1 0
RESERVED	SCALER RELOAD VALUE
0	NR
r	rw

sbits-1:0 Scaler reload value

18.7.5 UART FIFO Debug Register

Table 131. 0x10 - DEBUG - UART FIFO debug register

31 8	7 0
RESERVED	DATA
0	NR
r	rw

7: 0 Transmitter holding register or FIFO (read access)

7: 0 Receiver holding register or FIFO (write access)

18.7.6 UART FIFO Debug Control Register

Table 132. 0x14 - DCTRL - UART FIFO debug control register

31 1	0
RESERVED	DB
0	0
r	rw

0 FIFO debug mode enable (DB) - when set, it is possible to read and write the FIFO debug register.

18.7.7 UART Capability Register

Table 133. 0x18 - CAP - UART capability register

31 7	6	5 0
RESERVED	FL	FIFO
0	*	*
r	r	r

6 Flow control (FL) - This filed signals if the core has been implemented with flow control support.

5: 0 FIFO size (FIFO) - This field contains the depth of the core's internal FIFOs. When FIFO size is 1 then holding registers are used in the core.

18.8 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x00C. For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

18.9 Implementation

18.9.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

18.9.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual).

18.9.3 ProASIC3-specific issues

The core does not explicitly instantiate any RAM blocks. However, when the core is synthesized for ProASIC3 using Synplify, the tool may still implement the transmitter FIFO in a RAM block. This is known to occur with the following Synplify versions:

- Synplify Pro ME L-2016.09M-SP1-5
- Synplify Pro ME P-2019.03A-SP1

In such an implementation, a write to the data register that occurs on the same cycle as a scaler tick will result in simultaneous read and write of one RAM address if the transmitter FIFO is empty. This can result in incorrect data being transmitted in the core's serial data output.

Implementation of the transmitter FIFO in a RAM block can be prevented by adding the following constraint (e.g. to the file "synplify.sdc"):

```
define_attribute {v:gaisler.apbuart*} {syn_ramstyle} {registers}
```

With this constraint in place, Synplify will still infer that the transmitter FIFO is logically a RAM. But the RAM will be implemented in registers and logic instead of in a RAM block.

18.10 Configuration options

Table 134 shows the configuration options of the core (VHDL generics).

*Table 134.*Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
console	When 1, prints output from the UART on console during VHDL simulation. Additionally speeds up simulation by making reads of the status register always return values consistent with the transmitter FIFO and transmitter shift register being empty. Does not affect synthesis.	0 - 1	0
pirq	Index of the interrupt line.	0 - NAHBIRQ-1	0
parity	Currently unused.	0 - 1	1
flow	Enables flow control. Flow control must be implemented for FIFO debug mode to be supported. Setting this generic to 0 also disables FIFO debug mode.	0 - 1	1
fifosize	Selects the size of the Receiver and Transmitter FIFOs	1, 2, 4, 8, 16, 32	1
abits	Selects the number of APB address bits used to decode the register addresses	3 - 8	8
sbits	Selects the number of bits in the scaler	12-32	12

18.11 Signal descriptions

Table 135 shows the interface signals of the core (VHDL ports).

*Table 135.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
UARTI	RXD	Input	UART receiver data	-
	CTSN	Input	UART clear-to-send	Low
	EXTCLK	Input	Use as alternative UART clock	-
UARTO	RTSN	Output	UART request-to-send	Low
	TXD	Output	UART transmit data	-
	SCALER	Output	UART scaler value	-
	TXEN	Output	Output enable for transmitter	High
	FLOW	Output	Unused	-
	RXEN	Output	Receiver enable	High

* see GRLIB IP Library User's Manual

18.12 Signal definitions and reset values

The signals and their reset values are described in table 136.

*Table 136.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
txd[]	Output	UART transmit data line	-	Logical 1
rtsn[]	Output	Ready To Send	Low	Logical 1
rxd[]	Input	UART receive data line	-	-
ctsn[]	Input	Clear To Send	Low	-

18.13 Timing

The timing waveforms and timing parameters are shown in figure 28 and are defined in table 137.

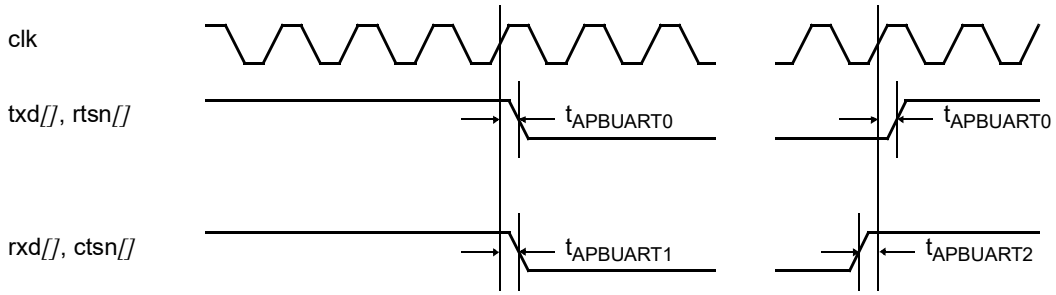


Figure 28. Timing waveforms

*Table 137.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{APBUART0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns
$t_{APBUART1}$	input to clock hold	rising *clk* edge	-	-	ns
$t_{APBUART2}$	input to clock setup	rising *clk* edge	-	-	ns

Note: The *ctsn[]* and *rxd[]* inputs are re-synchronized internally. These signals do not have to meet any setup or hold requirements.

18.14 Library dependencies

Table 138 shows libraries that should be used when instantiating the core.

*Table 138.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	APB signal definitions
GAISLER	UART	Signals, component	Signal and component declaration

18.15 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.uart.all;

entity apbuart_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- UART signals
    rxd   : in  std_ulogic;
    txd   : out std_ulogic
    );
end;

architecture rtl of apbuart_ex is
```

```
  -- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- UART signals
  signal uarti : uart_in_type;
  signal uarto : uart_out_type;

begin

  -- AMBA Components are instantiated here
   ...


  -- APB UART
  uart0 : apbuart
  generic map (pindex => 1, paddr => 1,  pirq => 2,
console => 1, fifosize => 1)
  port map (rstn, clk, apbi, apbo(1), uarti, uarto);

  -- UART input data
  uarti.rxd <= rxd;

  -- APB UART inputs not used in this configuration
  uarti.ctsn <= '0'; uarti.extclk <= '0';

  -- connect APB UART output to entity output signal
  txd <= uarto.txd;

end;
```

19 APBUART16550 - AMBA APB UART 16550 Serial Interface

19.1 Overview

The interface is designed for serial communications. The APBUART16550 is an adaptation of the previous APBUART, now fully compliant with the industry-standard 16550 UART. The register map has been modified to align with the 16550 standard, while some additional registers have been added at higher memory positions to retain certain functionalities from the original APBUART.

The UART supports data frames with 8 data bits, one optional parity bit, and one or two stop bits. To generate the bit-rate, each UART has a programmable 16-bit clock divider. Data transfer between the APB bus and the UART can utilize two FIFOs when the *fifomode* VHDL generic is set to 1. When *fifomode* is set to 0, or if FIFOs are disabled through software, two holding registers are used for data transfer.

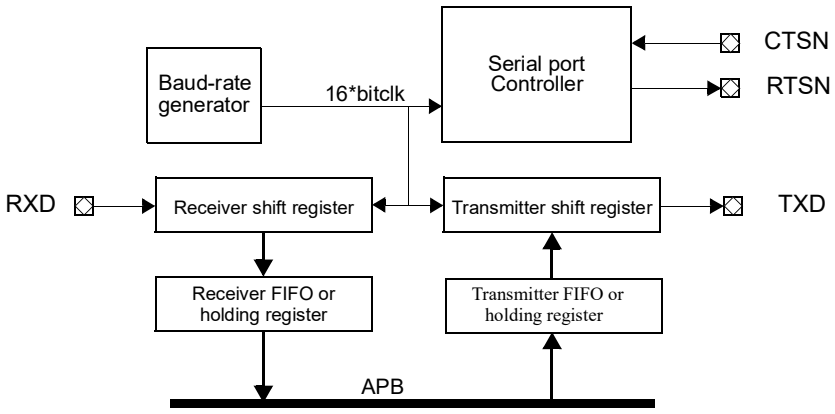


Figure 29. Block diagram

19.2 Operation

When the *fifomode* VHDL generic is set to 1, an 8-bit, 16-position FIFO is instantiated, allowing the UART to enable or disable the receiver and transmitter FIFOs by setting the FCR0 bit. When FIFOs are disabled, only a single holding register is used. Whenever the UART has its FIFOs active, we will refer to it as being in FIFO mode. Conversely, when the FIFOs are disabled, we will refer to it as being in 16450 mode.

When the *fifomode* VHDL generic is set to 0, no FIFO is instantiated, and writing to FCR0 has no effect. In this situation, the UART will remain permanently in 16450 mode, making it compatible only with the 16450 standard but not with the 16550 standard.

Whether the UART is configured to use FIFOs or holding registers, both terms will be used interchangeably throughout this document.

19.2.1 Transmitter operation

After reset, the transmitter is enabled to ensure compatibility with the 16550 standard. However, the activation of the transmitter can be controlled through the Transmitter Enable (TE) bit in the UART Custom Configuration Register. Data to be transferred is stored in the FIFO or holding register by writing to the data register.

When ready to transmit, data is transferred from the transmitter FIFO/holding register to the transmitter shift register and converted to a serial stream on the transmitter serial output pin (TXD). It automatically sends a start bit followed by eight data bits, an optional parity bit, and one stop bit (figure 30). The least significant bit of the data is sent first. It is also possible to use two stop bits, this is configured via the Line Control Register.

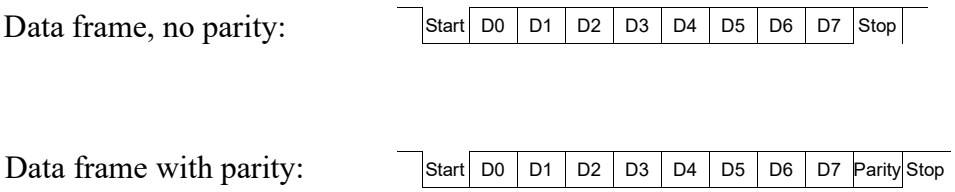


Figure 30. UART data frames

Following the transmission of the stop bit, if a new character is not available in the transmitter FIFO, the transmitter serial data output remains high, and the Transmitter Empty (TEMT) bit will be set in the UART Line Status Register. Transmission resumes and the TEMT bit is cleared when a new character is loaded into the transmitter FIFO. When the FIFO is empty, the THRE bit is set in the Line Status Register even if the transmitter shift register is full.

If the transmitter is disabled, it will immediately stop any active transmissions, including the character currently being shifted out from the transmitter shift register. The transmitter holding register should not be loaded when the FIFO (or holding register) is full, as this may cause data to be overwritten. The discussion above applies to both FIFO mode and 16450 mode (using only the holding register).

There is an extra register called the Transmitter FIFO Count, which is not part of the 16550 standard. This register contains a counter (TCNT) showing the current number of data entries in the transmitter FIFO. In 16450 mode, this count is limited to 1.

19.2.2 Receiver operation

The serial input signal is first passed through a meta-stability filter. The first stage of this filter consists of two flip-flops connected in series that are clocked by the system clock. The output of the second flip-flop is then sampled into a three-bit shift register at a rate of 16 times the configured baudrate. Finally, the majority vote of the bits in the shift register is used as the serial data input for the rest of the receiver.

After reset, the receiver is enabled to ensure compatibility with the 16550 standard. However, the activation of the receiver can be controlled through the Receiver Enable (RE) bit in the UART Custom Configuration Register. The receiver looks for a high to low transition of a start bit on the receiver serial data input. If a transition is detected, the state of the serial input is sampled a half bit clocks later. If the serial input is sampled, high the start bit is invalid and the search for a valid start bit continues. If the serial input is still low, a valid start bit is assumed and the receiver continues to sample the serial input at one bit time intervals (at the theoretical center of the bit) until the proper number of data bits and the parity bit have been assembled and one stop bit has been detected. If more than one stop bit is received then the all but the first are interpreted as the serial input being idle. In particular, the number of stop bits configured in the control register (via the NS bit) has no effect on receiver operation.

During reception, the least significant bit is received first. The data is then transferred to the receiver FIFO and the receiver Data Ready (DR) bit is set in the Line Status Register as soon as the FIFO contains at least one data frame.

There is an extra register called the Receiver FIFO Count, which is not part of the 16550 standard. This register contains a counter (RCNT) showing the current number of data entries in the receiver FIFO. In 16450 mode, this count is limited to 1.

19.3 Baud-rate generation

Each UART contains a 16-bit down-counting scaler to generate the desired baud-rate, the number of scaler bits is set to 16 for compatibility with the UART 16550 standard, but this can be modified with VHDL generic *sbits*. The scaler is clocked by the system clock and generates a UART tick each time it underflows. It is reloaded with the value of the UART scaler reload register after each underflow. The resulting UART tick frequency should be 16 times the desired baud-rate. One appropriate formula to calculate the scaler value for a desired baud rate, using integer division where the remainder is discarded, is:

*scaler value = (system_clock_frequency) / (baud_rate * 16 + 15).*

To calculate the exact required scaler value use:

*scaler value = (system_clock_frequency) / (baud_rate * 16) - 1*

If the EC bit is set, the ticks will be generated with the same frequency as the external clock input instead of at the scaler underflow rate. In this case, the frequency of external clock must be less than half the frequency of the system clock.

19.4 Flow Control Signals

The VHDL generic flow enables UART flow control. When the flow VHDL generic is set to 0, the CTSN input and RTSN outputs are internally set to 0, keeping them always asserted. This configuration is not compliant with the 16550 UART standard. When the flow VHDL generic is set to 1, the RTSN signal is controlled via bit 1 of the MODEM Control Register. The CTSN value can be retrieved by reading bit 4 of the MODEM Status Register. Changes in the CTSN signal are reflected in bit 0 of the MODEM Status Register, and if bit 3 of the Interrupt Enable Register is set to 1, an interrupt will also be raised to reflect changes in CTSN. The CTSN input has no effect on the transmitter.

19.5 Loop back mode

If the LB bit in the Modem Control Register is set, the UART will be in loop back mode. In this mode, the transmitter output is internally connected to the receiver input and the RTSN is connected to the CTSN. It is then possible to perform loop back tests to verify operation of receiver, transmitter and associated software routines. In this mode, the outputs remain in the inactive state, in order to avoid sending out data.

19.6 FIFO debug mode

FIFO debug mode is entered by setting the debug mode bit (DB) in the Debug Mode Enable Register. In this mode it is possible to read the transmitter FIFO and write the receiver FIFO through the FIFO debug register. The transmitter output is held inactive when in debug mode. A write to the receiver FIFO generates an interrupt if receiver interrupts are enabled.

FIFO debug mode requires that the hardware implementation supports flow control (VHDL generic *flow* = 1).

19.7 Error detection

The UART detects three sources of errors: overrun error, parity error, and framing error. Each of these errors is indicated by its respective bit in the Line Status Register (LSR). Additionally, there is a bit (BI) in the LSR that is set when a break condition occurs. Each of these bits is explained further in Table 149. When the Receiver Line Status interrupt is enabled via bit 2 of the Interrupt Enable Register, an interrupt will be raised each time any of the mentioned bits is set.

19.8 Interrupt generation

The UART supports five different types of interrupts. Interrupts can be enabled or disabled through bits 0 to 3 of the Interrupt Enable Register (IER). Setting each of these bits to 1 will enable the corresponding type of interrupt, except for bit 0, which enables both the Received Data Available Interrupt and timeout interrupts if FIFO mode is enabled.

The five types of interrupts, ordered by priority from highest to lowest, are: Receiver Line Status interrupt, Received Data Available and Character Timeout interrupts (which share the same priority level), Transmitter Holding Register interrupt, and MODEM Status interrupt. To determine the highest enabled and pending interrupt, the Interrupt Identification Register (IIR) should be read. Table 139 shows how to interpret the content of the IIR to identify the highest priority interrupt source and the actions required to clear the interrupt.

Table 139. Interrupt Control

Interrupt Identification Register				Interrupts			
Bit 3	Bit2	Bit1	Bit 0	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset
0	0	0	1	-	None	None	-
0	1	1	0	Highest	Receiver Line Status	Break Interrupt or Framing Error or Parity Error or Overrun Error	Reading the Line Status Register
0	1	0	0	Second	Received Data Available	Receiver Data Available or Trigger Level Reached	Reading the receiver holding register or the FIFO falls below the trigger level
1	1	0	0	Second	Character Timeout Indication (Only with FIFO mode active)	No characters have been removed from or input to the receiver FIFO during the last four character times, and there is at least one character present in the FIFO during this period.	Reading the receiver FIFO
0	0	1	0	Third	Transmitter Holding Register/FIFO Empty	Transmitter Holding Register/FIFO Empty	Reading the IIR Register (if it is the interrupt source) or Writing into the Transmitter Holding Register/FIFO
0	0	0	0	Fourth	MODEM Status	Delta Clear to Send (DCTS) bit is set to a logic 1.	Reading the MODEM Status Register

When receiver interrupts are enabled (IER0=1), the receive data available interrupt will notify the CPU when a character is transferred from the shift register to the receiver holding register or when the FIFO reaches its specified trigger level. This interrupt is cleared once the receiver holding register is read or the FIFO level drops below the trigger level. Similarly, the IIR receive data available indication is triggered under the same conditions and cleared in the same way. The data ready bit (LSR0) is set as soon as a character moves from the shift register to the receiver holding register or FIFO and is reset when the holding register or FIFO is empty.

When the receiver FIFO and interrupts are active, a FIFO timeout interrupt will be generated if:

• There is at least one character in the FIFO.

• No serial character has been received for more than 4 continuous character times (including the second stop bit if 2 stop bits are configured).

• The FIFO has not been read by the CPU for more than 4 continuous character times.

This timeout interrupt is cleared, and the timer is reset when the CPU reads a character from the receiver FIFO. If no timeout interrupt has occurred, the timeout timer is reset when a new character is

received or after the CPU reads the receiver FIFO. If both timeout and trigger level reached interrupts are pending simultaneously, IIR3 will be set to 1, indicating a timeout interrupt until it is cleared.

When transmitter interrupts are enabled (IER1=1), an interrupt will be triggered when the transmitter holding register or FIFO becomes empty. This interrupt is cleared when data is written to the transmitter holding register or FIFO.

19.9 Registers

The core is controlled through registers mapped into the APB address space. Table 140 lists the registers implemented as defined by the 16550 standard. Table 141 lists additional registers that have been added at higher memory positions to retain certain functionalities from the original APBUART.

Note that the value of the DLAB bit in the Line Control Register affects access to the APB UART registers mapped to addresses 0x00 and 0x04. The DLAB must be set high by the system software to access the Baud Generator Divisor Latches.

*Table 140.*Standard 16550 UART registers

APB address offset	Register
0x0 (DLAB = 0)	Receiver Buffer (read) / Transmitter Holding (write) Register
0x4 (DLAB = 0)	Interrupt Enable Register
0x8	Interrupt Identification (read) / FIFO Control (write) Register
0xC	Line Control Register
0x10	MODEM Control Register
0x14	Line Status Register
0x18	MODEM Status Register
0x1C	Scratch Register
0x0 (DLAB = 1)	Divisor Latch (LS)
0x4 (DLAB = 1)	Divisor Latch (MS)

*Table 141.*Custom UART registers

APB address offset	Register
0x20	Custom Control Register
0x24	Receiver FIFO Count Register
0x28	Transmitter FIFO Count Register
0x2C	FIFO Debug Mode Register
0x30	FIFO Debug Register

19.9.1 Receiver Buffer Register (RBR) / Transmitter Holding Register (THR)

Table 142. 0x00 (DLAB = 0) - RBR/THR

31 … 8	7 … 0
RESERVED	DATA
	0x00
	rw

7: 0 Receiver Buffer Register (read access)

7: 0 Transmitter Holding Register (write access)

19.9.2 Interrupt Enable Register (IER)

Table 143. 0x04 (DLAB = 0) - IER

31 8	7 4	3	2	1	0
RESERVED	RESERVED	IER3	IER2	IER1	IER0
		0	0	NR	NR
		rw	rw	rw	rw

7: 4 RESERVED

3 Setting this bit to logic 1 activates the MODEM Status Interrupt.

2 Setting this bit to logic 1 activates the Receiver Line Status Interrupt.

1 Setting this bit to logic 1 activates the Transmitter Holding Register Empty Interrupt.

0 Setting this bit to logic 1 activates the Received Data Available Interrupt and the timeout interrupts in FIFO mode.

19.9.3 FIFO control Register (FCR)

Table 144. 0x08 (write only) - FCR

31 8	7 6	5 3	2	1	0
RESERVED	FCR7-6	RESERVED	FCR2	FCR1	FCR0
	0		0	0	0
	w		w	w	w

7: 6 FCR6 and FCR7 are used to set the trigger level for the receiver FIFO interrupt. The different possible values are shown in table 145.

5: 3 RESERVED

2 Setting FCR2 to 1 clears all bytes in the transmitter FIFO and resets its counter logic to 0. The shift register remains unaffected. The bit automatically resets itself after being set.

1 Setting FCR1 to 1 clears all bytes in the receiver FIFO and resets its counter logic to 0. The shift register remains unaffected. The bit automatically resets itself after being set.

0 Setting FCR0 to 1 enables both the transmitter and receiver FIFOs. Resetting FCR0 clears all bytes in both FIFOs. When switching between FIFO Mode and 16450 Mode, the FIFOs are automatically cleared. This bit must be set to 1 before modifying other FCR bits, or those bits will not be programmed.

*Table 145.*Receiver FIFO Trigger Level configuration

FCR7	FCR6	Receiver FIFO Trigger Level (Bytes)
0	0	1
0	1	4
1	0	8
1	1	14

19.9.4 Interrupt Identification Register (IIR)

Table 146. 0x08 (read only) - IIR

31 8	7 6	5 4	3	2 1	0
RESERVED	IIR7-6	RESERVED	IIR3	IIR2-1	IIR0
	0		0	0	1
	r		r	r	r

7: 6 These bits are set when FCR0 is 1

5: 4 RESERVED

3 In 16450 Mode, this bit is always 0. In FIFO mode, this bit is set along with bit 2 when a timeout interrupt is pending.

2: 1 These bits are used to identify the highest priority interrupt that is pending, as shown in (Table 139).

0 This bit indicates whether an interrupt is pending in a prioritized interrupt environment. A logic 0 means an interrupt is pending, and the contents of the IIR can be used to point to the appropriate interrupt service routine. A logic 1 means no interrupt is pending.

19.9.5 UART Line Control Register (LCR)

Table 147. 0x0C - LCR

31 8	7	6	5	4	3	2	1 0
RESERVED	DLAB	BC	RES	EPS	PE	NS	CL
	0	0		NS	NS	0	0b11
	rw	rw		rw	rw	rw	r

7 Divisor Latch Access Bit (DLAB) - It must be set to logic 1 to access the Divisor Latches of the Baud Generator during a Read or Write operation. It must be set low to access the Receiver Buffer, the Transmitter Holding Register, or the Interrupt Enable Register.

6 Break Control (BC) bit - Setting this bit to logic 1 forces the serial output to the Spacing (logic 0) state, initiating a break condition. Clearing BC (setting it to logic 0) disables the break. The Break Control bit affects only the serial output and does not impact the transmitter logic.

5 RESERVED

4 Even Parity Select (EPS) bit - When bit 3 is set to logic 1, if bit EPS is set to logic 0, the Parity bit is chosen to make the total number of 1s odd. If bit EPS is set to logic 1, the total number of 1s is even.

3 Parity Enable (PE) bit - Setting this bit to logic 1 causes a Parity bit to be generated (for transmitted data) or checked (for received data) between the last data bit and the Stop bit.

2 Number of Stop Bits (NS) bit - Determines the transmitted and received Stop bits in each serial character. If bit SB is a logic 0, one Stop bit is generated in the transmitted data. If SB is a logic 1 two Stop bits are generated.

1: 0 These two bits specify the character length (CL) in each transmitted or received serial character. This field is fixed to the binary value “11”, which permanently sets the character length to 8 bits. Writes to this field have no effect.

19.9.6 UART MODEM Control Register (MCR)

Table 148. 0x10 - MCR

31 8	7 5	4	3 2	1	0
RESERVED	RSERVED	LB	RESERVED	RTS	RES
		NS		0	
		rw		rw	

7: 5 RESERVED

4 Loop back (LB) - if set, loop back mode will be enabled.

3: 2 RESERVED

1 When this bit is set to logic 1, the RTSN output is driven to logic 0. When this bit is cleared to logic 0, the RTSN output is driven to logic 1.

0 RESERVED

19.9.7 Line Status Register (LSR)

Table 149. 0x14 - LSR

31 8	7	6	5	4	3	2	1	0
RESERVED	LRSR7	TEMT	THRE	BI	FE	PE	OE	DR
	0	1	1	0	0	0	0	0
	r	r	r	r	r	r	r	r

7 In the 16450 Mode, this bit is always 0. In FIFO mode, bit 7 (LSR7) is set when there is at least one parity error, framing error, or break indication present in the FIFO. LSR7 is cleared when the CPU reads the Line Status Register, provided there are no additional errors in the FIFO.

6 Transmitter Empty (TEMT) bit - is set to logic 1 when both the Transmitter Holding Register (THR) and the Transmitter Shift Register (TSR) are empty. The bit is reset to logic 0 whenever either the THR or TSR contains a data character. In FIFO mode, this bit is set to one when both the transmitter FIFO and the shift register are empty.

5 Transmitter Holding Register Empty (THRE) bit - signifies that the UART is ready to accept a new character for transmission. When the Transmit Holding Register Empty Interrupt enable is set to high, this bit also triggers an interrupt to the CPU. The THRE bit is set to logic 1 when a character is transferred from the Transmitter Holding Register to the Transmitter Shift Register. It is reset to logic 0 when the CPU loads a new character into the Transmitter Holding Register. In FIFO mode, this bit is set when the transmission FIFO is empty and is cleared when at least one byte is written to the transmission FIFO.

4 Break Interrupt (BI) bit - is set to logic 1 when the received data input is held in the Spacing (logic 0) state for longer than the time required to transmit a full word (including Start bit, data bits, Parity bit, and Stop bits). The BI indicator is reset to logic 0 when the CPU reads the Line Status Register. In FIFO mode, this error is linked to the specific character in the FIFO, and the CPU is informed when the character reaches the top of the FIFO. When a break occurs, only one zero character is loaded into the FIFO. The next character transfer is enabled once serial input returns to the marking state and a valid start bit is received.

3 Framing Error (FE) bit - is set to logic 1 when the received character lacks a valid Stop bit. Specifically, it is set when the Stop bit following the last data or parity bit is detected as a logic 0 (Spacing level). The FE indicator is reset to logic 0 when the CPU reads the Line Status Register. In FIFO mode, this error is tied to the specific character in the FIFO, and the CPU is notified when the character reaches the top of the FIFO.

2 Parity Error (PE) bit - is set to logic 1 if the received data character does not have the correct parity (even or odd) as specified by the parity settings. The PE bit is reset to logic 0 when the CPU reads the Line Status Register. In FIFO mode, this error is linked to the specific character in the FIFO, and the CPU is informed of the error when the character reaches the top of the FIFO.

1 Overrun Error (OE) bit - is set to logic 1 when data in the Receiver Buffer Register is not read by the CPU before the next character arrives, causing the previous character to be overwritten. The OE indicator is reset to logic 0 when the CPU reads the Line Status Register. In FIFO mode, an overrun error occurs only after the FIFO is full and the next character is completely received in the shift register. The CPU is immediately notified of the overrun condition, and the character in the shift register is overwritten but not transferred to the FIFO.

0 Data Ready (DR) bit - Serves as the Receiver Data Ready indicator. It is set to logic 1 when a complete incoming character has been successfully received and moved into the Receiver Buffer Register or FIFO. Reading all the data in the Receiver Buffer Register or FIFO resets this bit to logic 0.

19.9.8 MODEM Status Register (MSR)

Table 150. 0x18 - MSR

31 8	7 5	4	3 1	0
RESERVED	RESERVED	CTS	RESERVED	DCTS
		0		0
		r		r

7: 5 RESERVED

4 Clear to Send (CTS) - represents the inverse of the CTSN input. When LB (loop back) of the Modem Control Register (MCR) is set to 1, this bit mirrors the Ready to Send (RTS) signal in the MCR.

3: 1 RESERVED

0 Delta Clear to Send (DCTS) bit - signifies that the Clear to Send (CTS) input to the UART has changed state since it was last read by the CPU. Reading the MSR register automatically clears this bit.

19.9.9 Scratch Register (SCR)

Table 151. 0x1C - SCR

31 8	7 0
RESERVED	DATA
	0
	rw

7: 0 This register does not influence the operation of the UART. It is designed to serve as a temporary storage register for the programmer, allowing for the temporary holding of data.

19.9.10 Divisor Latch LS (DLL)

Table 152. 0x00 (DLAB = 1) - DLL

31 8	7 0
RESERVED	SCALER RELOAD VALUE(LS)
	NR
	rw

7: 0 Least significant byte of the scaler reload value

19.9.11 Divisor Latch MS (DLM)

Table 153. 0x04 (DLAB = 1) - DLM

31 8	7 0
RESERVED	SCALER RELOAD VALUE(MS)
	NR
	rw

7: 0 Most significant byte of the scaler reload value

19.9.12 Custom Configuration Register (CCR)

Table 154. 0x20 - CCR

31 ... 8	7 ... 3	2	1	0
RESERVED	RESERVED	EC	TE	RE
		NR	1	1
		rw	rw	rw

7: 3 RESERVED

2 External Clock (EC) - if set, the UART scaler will be clocked by UARTI.EXTCLK.

1 Transmitter enable (TE) - if set, enables the transmitter.

0 Receiver enable (RE) - if set, enables the receiver.

19.9.13 Receiver FIFO Count (RFC)

Table 155. 0x24 - RFC

31 ... 8	7 ... 0
RESERVED	RCNT
	0
	r

7: 0 Receiver FIFO count (RCNT) - shows the number of data frames in the receiver FIFO.

19.9.14 Transmitter FIFO Count (TFC)

Table 156. 0x28 - TFC

31 ... 8	7 ... 0
RESERVED	TCNT
	0
	r

7: 0 Transmitter FIFO count (TCNT) - shows the number of data frames in the transmitter FIFO.

19.9.15 Debug Mode Enable Register (DMR)

Table 157. 0x2C - DMR

31 ... 8	7 ... 1	0
RESERVED		DB
		NR
		rw

7: 1 RESERVED

0 FIFO debug mode enable (DB) - when set, it is possible to read and write the FIFO debug register.

19.9.16 FIFO Debug Register

Table 158. 0x10 - DEBUG - UART FIFO debug register

31 ... 8	7 ... 0
RESERVED	DATA
0	0x00
r	rw

7: 0 Transmitter holding register or FIFO (read access)

7: 0 Receiver holding register or FIFO (write access)

19.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x0D5. For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

19.11 Implementation

19.11.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

19.11.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual).

19.11.3 ProASIC3-specific issues

The core does not explicitly instantiate any RAM blocks. However, when the core is synthesized for ProASIC3 using Synplify, the tool may still implement the transmitter FIFO in a RAM block. This is known to occur with the following Synplify versions:

- Synplify Pro ME L-2016.09M-SP1-5
- Synplify Pro ME P-2019.03A-SP1

In such an implementation, a write to the data register that occurs on the same cycle as a scaler tick will result in simultaneous read and write of one RAM address if the transmitter FIFO is empty. This can result in incorrect data being transmitted in the core's serial data output.

Implementation of the transmitter FIFO in a RAM block can be prevented by adding the following constraint (e.g. to the file "synplify.sdc"):

```
define_attribute {v:gaisler.apbuart*} {syn_ramstyle} {registers}
```

With this constraint in place, Synplify will still infer that the transmitter FIFO is logically a RAM. But the RAM will be implemented in registers and logic instead of in a RAM block.

19.12 Configuration options

Table 159 shows the configuration options of the core (VHDL generics).

*Table 159.*Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
console	When 1, prints output from the UART on console during VHDL simulation. Additionally speeds up simulation by making reads of the status register always return values consistent with the transmitter FIFO and transmitter shift register being empty. Does not affect synthesis.	0 - 1	0
pirq	Index of the interrupt line.	0 - NAHBIRQ-1	0
parity	Currently unused.	0 - 1	1
flow	Enables flow control. Flow control must be implemented for FIFO debug mode to be supported. Setting this generic to 0 also disables FIFO debug mode.	0 - 1	1
fifomode	When set to 1, it instantiates receiver and transmitter FIFOs. When set to 0, the UART is not compatible with the UART 16550 standard and can only operate in 16450 mode using holding registers	0 - 1	1
abits	Selects the number of APB address bits used to decode the register addresses	3 - 8	6
sbits	Selects the number of bits in the scaler.	12-16	16

19.13 Signal descriptions

Table 160 shows the interface signals of the core (VHDL ports).

*Table 160.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
UARTI	RXD	Input	UART receiver data	-
	CTSN	Input	UART clear-to-send	Low
	EXTCLK	Input	Use as alternative UART clock	-
UARTO	RTSN	Output	UART request-to-send	Low
	TXD	Output	UART transmit data	-
	SCALER	Output	UART scaler value	-
	TXEN	Output	Output enable for transmitter	High
	FLOW	Output	Unused	-
	RXEN	Output	Receiver enable	High

* see GRLIB IP Library User's Manual

19.14 Signal definitions and reset values

The signals and their reset values are described in table 161.

*Table 161.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
txd[]	Output	UART transmit data line	-	Logical 1
rtsn[]	Output	Ready To Send	Low	Logical 1
rxd[]	Input	UART receive data line	-	-
ctsn[]	Input	Clear To Send	Low	-

19.15 Timing

The timing waveforms and timing parameters are shown in figure 31 and are defined in table 162.

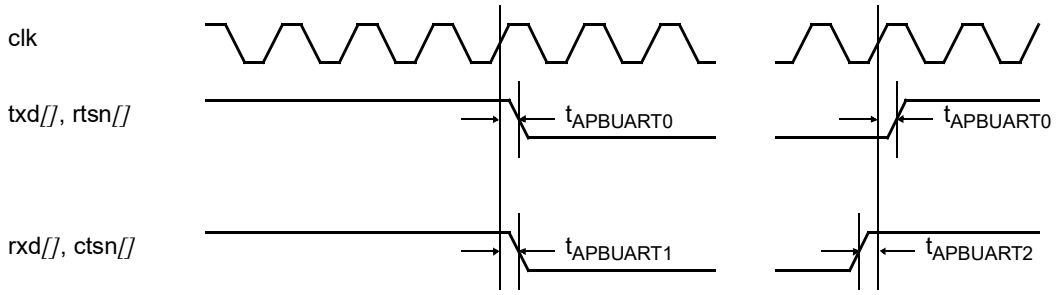


Figure 31. Timing waveforms

*Table 162.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{APBUART0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns
$t_{APBUART1}$	input to clock hold	rising *clk* edge	-	-	ns
$t_{APBUART2}$	input to clock setup	rising *clk* edge	-	-	ns

Note: The *ctsn[]* and *rxd[]* inputs are re-synchronized internally. These signals do not have to meet any setup or hold requirements.

19.16 Library dependencies

Table 163 shows libraries that should be used when instantiating the core.

*Table 163.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	APB signal definitions
GAISLER	UART	Signals, component	Signal and component declaration

19.17 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
```

```
library gaisler;
use gaisler.uart.all;

entity apbuart_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- UART signals
    rxd   : in  std_ulogic;
    txd   : out std_ulogic
    );
end;

architecture rtl of apbuart_ex is

  -- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- UART signals
  signal uarti : uart_in_type;
  signal uarto : uart_out_type;

begin

  -- AMBA Components are instantiated here
   ...


  -- APB UART
  uart0 : apbuart_16550
  generic map (pindex => 1, paddr => 1,  pirq => 2,
console => 1, fifomode => 1)
  port map (rstn, clk, apbi, apbo(1), uarti, uarto);

  -- UART input data
  uarti.rxd <= rxd;

  -- APB UART inputs not used in this configuration
  uarti.ctsn <= '0'; uarti.extclk <= '0';

  -- connect APB UART output to entity output signal
  txd <= uarto.txd;

end;
```

20 APBVGA - VGA controller with APB interface

20.1 Introduction

The APBVGA core is a text-only video controller with a resolution of 640x480 pixels, creating a display of 80x37 characters. The controller consists of a video signal generator, a 4 Kbyte text buffer, and a ROM for character pixel information. The video controller is controlled through an APB interface.

A block diagram for the data path is shown in figure 32.

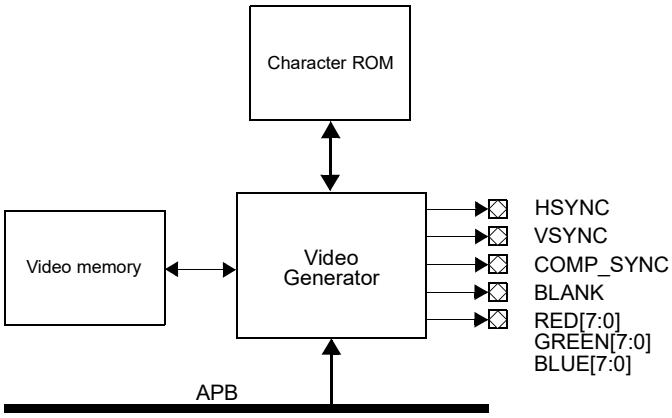


Figure 32. APBVGA block diagram

20.2 Operation

The video timing of APBVGA is fixed to generate a 640x480 display with 60 Hz refresh rate. The text font is encoded using 8x13 pixels. The display is created by scanning a segment of 2960 characters of the 4 Kbyte text buffer, rasterizing the characters using the character ROM, and sending the pixel data to an external video DAC using three 8-bit color channels. The required pixel clock is 25.175 MHz, which should be provided on the VGACLK input.

Writing to the video memory is made through the VGA data register. Bits [7:0] contains the character to be written, while bits [19:8] defines the text buffer address. Foreground and background colours are set through the background and foreground registers. These 24 bits corresponds to the three pixel colors, RED, GREEN and BLUE. The eight most significant bits defines the red intensity, the next eight bits defines the green intensity and the eight least significant bits defines the blue intensity. Maximum intensity for a color is received when all eight bits are set and minimum intensity when none of the bits are set. Changing the foreground color results in that all characters change their color, it is not possible to just change the color of one character. In addition to the color channels, the video controller generates HSYNC, VSYNC, CSYNC and BLANK. Togetherm the signals are suitable to drive an external video DAC such as ADV7125 or similar.

APBVGA implements hardware scrolling to minimize processor overhead. The controller monitors maintains a reference pointer containing the buffer address of the first character on the top-most line. When the text buffer is written with an address larger than the reference pointer + 2960, the pointer is incremented with 80. The 4 Kbyte text buffer is sufficient to buffer 51 lines of 80 characters. To simplify hardware design, the last 16 bytes (4080 - 4095) should not be written. When address 4079 has been written, the software driver should wrap to address 0. Sofware scrolling can be implemented by only using the first 2960 address in the text buffer, thereby never activating the hardware scolling mechanism.

20.3 Registers

The APB VGA is controlled through three registers mapped into APB address space.

Table 164. APB VGA registers

APB address offset	Register
0x0	VGA Data register (write-only, reads will return 0x00000000).
0x4	VGA Background color (write-only, reads will return 0x00000000).
0x8	VGA Foreground color (write-only, reads will return 0x00000000).

20.3.1 VGA Data Register

Table 165. 0x00 - DATA - VGA data register

31 20	19 8	7 0
RESERVED	ADDRESS	DATA
0	0	0
r	w	w

19: 8 Video memory address (write access)

7: 0 Video memory data (write access)

20.3.2 VGA Background Color

Table 166. 0x04 - BGCOL - VGA background register

31 24	23 16	15 8	7 0
RESERVED	RED	GREEN	BLUE
0			
r	w	w	w

23: 16 Video background color red.

15: 8 Video background color green.

7: 0 Video background color blue.

20.3.3 VGA Foreground Color

Table 167. 0x00 - FGCOL - VGA foreground register

31 24	23 16	15 8	7 0
RESERVED	RED	GREEN	BLUE
0			
r	w	w	w

23: 16 Video foreground color red.

15: 8 Video foreground color green.

7: 0 Video foreground color blue.

20.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x061. For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

20.5 Implementation

20.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

20.5.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual).

20.6 Configuration options

Table 168 shows the configuration options of the core (VHDL generics).

*Table 168.*Configuration options

Generic	Function	Allowed range	Default
memtech	Technology to implement on-chip RAM	0 - NTECH	2
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#

20.7 Signal descriptions

Table 169 shows the interface signals of the core (VHDL ports).

*Table 169.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
VGACLK	N/A	Input	VGA Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
VGAO	HSYNC	Output	Horizontal synchronization	High
	VSYNC		Vertical synchronization	High
	COMP_SYNC		Composite synchronization	Low
	BLANK		Blanking	Low
	VIDEO_OUT_R[7:0]		Video out, color red	-
	VIDEO_OUT_G[7:0]		Video out, color green	-
	VIDEO_OUT_B[7:0]		Video out, color blue	-
	BITDEPTH[1:0]		Constant High	-

* see GRLIB IP Library User's Manual

20.8 Library dependencies

Table 170 shows libraries used when instantiating the core (VHDL libraries).

*Table 170.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	APB signal definitions
GAISLER	MISC	Signals, component	VGA signal and component declaration

20.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

.
.


architecture rtl of apbuart_ex is

signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
signal vgao  : apbvga_out_type;

begin
  -- AMBA Components are instantiated here
   ...

  -- APB VGA
  vga0 : apbvga
  generic map (memtech => 2, pindex => 6, paddr => 6)
  port map (rstn, clk, vgaclk, apbi, apbo(6), vgao);
end;
```

21 AXI2AHB - AXI4 to AHB2 Bridge

21.1 Overview

The AXI4 to AHB2 bridge enables the usage of AXI4 masters in an AHB system. It has an AXI4 slave port as an input and an AHB master port as output. The IP also provides static endianness inversion to help integrating little endian components into a big endian system.

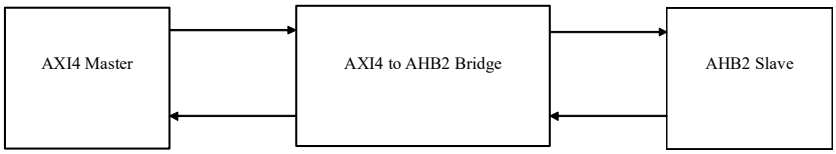


Figure 33. Example application

21.2 AXI4 Limitations

The AXI4 to AHB2 bridge currently does not support all AXI4 features. Incremental transfers are the only transfer type supported. This means that the bridge will not function as intended for the fixed and wrapping transfer types.

The following AXI4 signals are ignored by the bridge:

- AxCACHE
- AxREGION
- AxQOS
- AxLOCK
- AxPROT
- AxUSER
- xUSER

This means that the functions in the AXI4 protocol relying on these signals are not supported. It also means that there is no matching of features relying on these signals between the AXI4 and AHB2 protocols.

Response reporting only provides two possible response codes: OKAY and SLVERR. Any response from the AHB bus which is not OKAY are translated to SLVERR.

21.3 AHB2 Details

Due to the differences in the protocols, some AHB2 features can not be fully expressed. The HPROT signals is fixed to "0011", which translates into: not cachable, not bufferable, privileged data accesses.

21.4 Operation

There are no runtime control signals to the bridge. Thus the only way to operate the bridge is using the AXI4 slave interface. Please refer the AXI4 protocol standard document from Arm for AXI4 details. Also please do observe the limitations of the bridge, described by the AXI4 Limitations sections of this documentation.

The AXI4 to AHB2 bridge is limited to having matching data bus sizes of the AXI4 and AHB2 buses. If the AXI4 data bus size is greater than the AHB2 data bus size, the AXI4 resizer IP can be used to reduce the data bus size; before connecting it to the AXI4 to AHB2 bridge IP.

21.5 Implementation

21.5.1 Reset

The AXI4 to AHB2 bridge resets all its registers using an active low reset. The registers are reset synchronously.

21.5.2 Endianness

The endianness of the bridge is controlled using two different sources. The slave port endianness is controlled via the axi_endian configuration option, while the master port endianness is controlled via the abhmi.endian signal. The signal is expected to be static after reset. The AXI4 to AHB2 bridge will convert between the different endiannesses with the goal to keep the data consistent in regard to the transaction size.

21.5.3 Configuration Options

The AXI4 to AHB2 bridge can be configured at elaboration time using the generics described in this section.

Table 171 shows the configuration options of the core (VHDL generics).

*Table 171.*Configuration options

Generic name	Function	Allowed range	Default
memtech	Controls the memory technology used to implement the write data buffer.	0 - NTECH	inferred (0)
hindex	The master index for AHB master. Used to select which HGRANT to observe.	0	0
dbuffer	Controls the depth of the data buffer. Expressed as the number of words to buffer. Recommended to be a power of two.	4 -256	4
wordsize	The word size of the data words, in the number of bytes. The AXI to AHB2 bridge will consider the bus to only be as wide as expressed by this generic.	AHBDW	AHBDW
axi_endian	Controls the endianness expected from the AXI4 interface. 0: Big endian; 1: Little endian;	0 - 1	0
sub-_bus_width_address_inversion	In some cases it might be desirable to align narrow accesses to the width of the bus. Eg. a 16-bit access on a 32-bit bus. The address provided via the AXI interface is 0X0. The resulting address would in this case become 0X2.	0 - 1	0
mask	The 12 most significant bits of the address bus will be XORed with the provided mask. The mask must thus be within the range of [0. 4095].	16#000#	16#000#
vendorid	The AHB2 vendor ID to be used by the AHB2 master interface.	VENDOR_GAISLER	
deviceid	The AHB2 device ID to be used by the AHB2 master interface.	0	0
scantest	Enable scan for the memory variant for the read data FIFO.	0	0
memory_ft	Use a fault tolerant memory variant for the read data FIFO.	0	0

21.6 Signal Descriptions

Table 172 shows the interface signals of the bridge (VHDL ports).

*Table 172.*Signal descriptions

Signal name	Field	Type	Function	Active
resetn	N/A	Input	Reset	Low
clk	N/A	Input	AHB and AXI Clock	-
axisi	*	Input	AXI4 slave input signals. (not all signals are included please refer to the AXI4 limitations section for more information.)	-
axiso	*	Output	AXI4 slave output signals.	-
ahbmi	*	Input	AHB2 master input signals.	-
ahbmo	*	Output	AHB2 master output signals.	-
custom*	*	Input/ Output	MBIST related signals.	-

21.7 Library Dependencies

Table 173 shows the libraries used when instantiating the core (VHDL libraries).

*Table 173.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TECHMAP	gencomp	Components	Synchronous FIFO component.
GAISLER	axi	Signals	AXI4 related helper functions.

21.8 Instantiation

The following code snippet is intended to show how the core is instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all; -- For types and AXIDW

library gaisler;
use gaisler.axi.all; -- AXI4 to AHB2 bridge component

entity axi2ahb_example is
  port (
    clk : in std_logic;
    rstn : in std_logic;

    -- AXI4 slave signals
    s_axi4_awaddr : in std_logic_vector(31 downto 0);
    -- ... Rest of slave port declarations omitted.

    -- AHB2 master signals
    m_ahb2_haddr : out std_logic_vector(31 downto 0);
    -- ... Rest of master port declartions omitted.
    m_ahb2_hgrant : in std_logic
  );
```

```
end entity;

architecture ex of axi2ahb_example is
  constant hgrant_index : natural := 2;

  signal axisi : axi4_mosi_type;
  signal axiso : axi_somi_type;
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_type;
begin
  -- Assign slave inputs to record.
  axisi.aw.addr <= s_axi4_awaddr;
  -- ... Rest of AXI4 assignments omitted.

  -- Assign master inputs to record.
  ahbmi.hgrant <= (hgrant_index => m_ahb2_hgrant, others => '0');
  -- ... Rest of input assignments omitted.

  -- Assign master outputs to ports.
  m_ahb2_haddr <= ahbmo.haddr;
  -- ... Rest of output assignments omitted.

  axi2ahb_0 : axi2ahb
    generic map(
      memtech => 0, -- Generic memory implementation
      hindex => hgrant_index,
      dbuffer  => 64, -- Data buffer capable of storing 64 data words
      wordsize => AXIDW, -- To match record type width
      axi_endian => 1, -- Little endian AXI4 interface
      sub_bus_width_address_inversion => 0, -- Do not modify addresses
      mask => 16#000#, -- Do not mask the address
      vendorid => YOUR_VENDOR_ID,
      deviceid => YOUR_DEVICE_ID,
      scantest => 0, -- No SCAN support
      memory_ft => 0 -- Use a regular memory
    )
    port map(
      clk               => clk,
      resetn            => rstn,
      ahbmi             => ahbmi,
      ahbmo             => ahbmo,
      axisi             => aximo,
      axiso             => aximi
    );
end architecture;
```

22 AXI4_RESIZE - AXI4 Transaction Resize

22.1 Overview

The AXI4 resize component provides the capability of resizing AXI4 transactions. Resizing in this context means to manage a difference in data bus widths between the master and the slave. For slave widths greater or equal to the master width the module is transparent. It does not try to widen the transaction width.

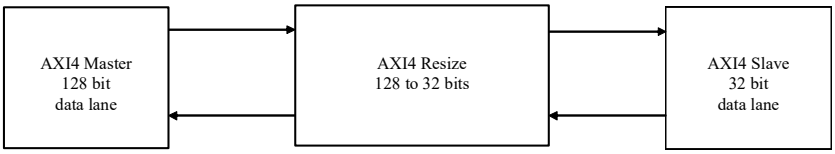


Figure 34. Figure: AXI4 resize usage example

22.2 Operation

Please refer to the AXI4 protocol standard document from Arm (IHI0022E) for AXI4 details.

The AXI4 resize component will automatically adjust the AxLEN and AxSIZE fields for any transaction exceeding the slave's capabilities. The AxSIZE field will be reduced to the maximum width allowed by the slave, as configured by the wl_m_data generic. If a resized transaction exceeds the limit imposed upon AxLEN, then multiple transactions will be created. Data flow and flags are managed as appropriate by the AXI4 resize component.

22.3 Implementation

22.3.1 Reset

The AXI4 resize component resets all its registers using an active low reset. The registers are either reset synchronously or asynchronously, depending on the GRLIB grlib_async_reset_enable configuration constant.

22.3.2 Endianness

AXI4 resize is endianness agnostic. It does not modify the endianness in any way.

22.3.3 Configuration Options

AXI4 resize can be configured at elaboration time using the generics described in this section. Table 594 shows the configuration options of the core (VHDL generics).

*Table 174.*Configuration options

Generic name	Function	Allowed range	Default
wl_s_data	Input data word length, controls the amount of data bits in the data input. The generic applies to both read and write channels.		128
wl_m_data	Output data word length, controls the amount of data bits in the data output. The generic applies to both read and write channels.		32
wl_user	Width of the AxUSER fields. In case of the creation of multiple transactions, the same user data will be added to the created transactions. The xUSER fields are not supported.		0
wl_id	Width of the different AXI4 id fields.		0

22.4 Signal Descriptions

Table 175 shows the interface signals (VHDL ports).

*Table 175.*Signal descriptions

Signal name	Field	Type	Function	Active
rst	N/A	Input	Reset	Low
clk	N/A	Input	AXI Clock	-
s_axi4_*	*	Input	Complete AXI4 slave interface with the exception of the WUSER and RUSER signals. (For a complete listing, please refer to the AXI4 standard or the instantiation example)	-
m_axi4_*	*	Output	Complete AXI4 master interface with the exception of the WUSER and RUSER signals. (For a complete listing, please refer to the AXI4 standard or the instantiation example)	-

22.5 Library Dependencies

Table 176 shows the libraries used when instantiating the core (VHDL libraries).

*Table 176.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	config, config_type	Signals	Configuration definition and support.

22.6 Instantiation

The following code snippet is intended to show how the core is instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library gaisler;
use gaisler.axi.all; -- AXI4 resize component

entity axi4_resize_example is
```

```
  port (
    clk : in std_logic;
    rstn : in std_logic
  );
end entity;

architecture ex of axis_geabox_example is
  constant slave_data_width : natural := 128;
  constant master_data_width : natural := 32;
  constant wl_id : natural := 4;
  constant wl_user : natural := 2;

  -- Slave ports
  signal s_axi4_awvalid : std_logic := '0';
  signal s_axi4_awaddr : std_logic_vector(31 downto 0);
  signal s_axi4_awsize : std_logic_vector(2 downto 0);
  signal s_axi4_awburst : std_logic_vector(1 downto 0);
  signal s_axi4_awlen : std_logic_vector(7 downto 0);
  signal s_axi4_awcache : std_logic_vector(3 downto 0) := (others => '0');
  signal s_axi4_awregion : std_logic_vector(3 downto 0) := (others => '0');
  signal s_axi4_awqos : std_logic_vector(3 downto 0) := (others => '0');
  signal s_axi4_awprot : std_logic_vector(2 downto 0) := (others => '0');
  signal s_axi4_awlock : std_logic_vector(1 downto 0) := (others => '0');
  signal s_axi4_awid : std_logic_vector(wl_id - 1 downto 0) := (others => '0');
  signal s_axi4_awuser : std_logic_vector(wl_user - 1 downto 0) := (others => '0');
  signal s_axi4_awready : std_logic;

  signal s_axi4_wdata : std_logic_vector(slave_data_width - 1 downto 0);
  signal s_axi4_wstrb : std_logic_vector(slave_data_width/8 - 1 downto 0);
  signal s_axi4_wlast : std_logic;
  signal s_axi4_wvalid : std_logic := '0';
  signal s_axi4_wready : std_logic;

  signal s_axi4_bvalid : std_logic;
  signal s_axi4_bresp : std_logic_vector(1 downto 0);
  signal s_axi4_bid : std_logic_vector(wl_id - 1 downto 0);
  signal s_axi4_buser : std_logic_vector(wl_user - 1 downto 0);
  signal s_axi4_bready : std_logic := '0';

  signal s_axi4_arvalid : std_logic := '0';
  signal s_axi4_araddr : std_logic_vector(31 downto 0);
  signal s_axi4_arsize : std_logic_vector(2 downto 0);
  signal s_axi4_arburst : std_logic_vector(1 downto 0);
  signal s_axi4_arlen : std_logic_vector(7 downto 0);
  signal s_axi4_arcache : std_logic_vector(3 downto 0) := (others => '0');
  signal s_axi4_arregion : std_logic_vector(3 downto 0) := (others => '0');
  signal s_axi4_arqos : std_logic_vector(3 downto 0) := (others => '0');
  signal s_axi4_arprot : std_logic_vector(2 downto 0) := (others => '0');
  signal s_axi4_arlock : std_logic_vector(1 downto 0) := (others => '0');
  signal s_axi4_arid : std_logic_vector(wl_id - 1 downto 0) := (others => '0');
  signal s_axi4_aruser : std_logic_vector(wl_user - 1 downto 0) := (others => '0');
  signal s_axi4_arready : std_logic;

  signal s_axi4_rdata : std_logic_vector(slave_data_width - 1 downto 0);
  signal s_axi4_rresp : std_logic_vector(1 downto 0);
  signal s_axi4_rid : std_logic_vector(wl_id - 1 downto 0);
  signal s_axi4_rlast : std_logic;
  signal s_axi4_rvalid : std_logic;
  signal s_axi4_rready : std_logic := '0';

  -- Master ports
  signal m_axi4_awvalid : std_logic;
  signal m_axi4_awaddr : std_logic_vector(31 downto 0);
  signal m_axi4_awsize : std_logic_vector(2 downto 0);
  signal m_axi4_awburst : std_logic_vector(1 downto 0);
  signal m_axi4_awlen : std_logic_vector(7 downto 0);
  signal m_axi4_awcache : std_logic_vector(3 downto 0);
  signal m_axi4_awregion : std_logic_vector(3 downto 0);
  signal m_axi4_awqos : std_logic_vector(3 downto 0);
  signal m_axi4_awprot : std_logic_vector(2 downto 0);
  signal m_axi4_awlock : std_logic_vector(1 downto 0);
```

```
  signal m_axi4_awid : std_logic_vector(wl_id - 1 downto 0);
  signal m_axi4_awuser : std_logic_vector(wl_user - 1 downto 0);
  signal m_axi4_awready : std_logic := '0';

  signal m_axi4_wdata : std_logic_vector(master_data_width - 1 downto 0);
  signal m_axi4_wstrb : std_logic_vector(master_data_width/8 - 1 downto 0);
  signal m_axi4_wlast : std_logic;
  signal m_axi4_wvalid : std_logic;
  signal m_axi4_wready : std_logic := '0';

  signal m_axi4_bvalid : std_logic := '0';
  signal m_axi4_bresp : std_logic_vector(1 downto 0) := (others => '0');
  signal m_axi4_bid : std_logic_vector(wl_id - 1 downto 0) := (others => '0');
  signal m_axi4_buser : std_logic_vector(wl_user - 1 downto 0) := (others => '0');
  signal m_axi4_bready : std_logic;

  signal m_axi4_arvalid : std_logic;
  signal m_axi4_araddr : std_logic_vector(31 downto 0);
  signal m_axi4_arsize : std_logic_vector(2 downto 0);
  signal m_axi4_arburst : std_logic_vector(1 downto 0);
  signal m_axi4_arlen : std_logic_vector(7 downto 0);
  signal m_axi4_arcache : std_logic_vector(3 downto 0);
  signal m_axi4_arregion : std_logic_vector(3 downto 0);
  signal m_axi4_arqos : std_logic_vector(3 downto 0);
  signal m_axi4_arprot : std_logic_vector(2 downto 0);
  signal m_axi4_arlock : std_logic_vector(1 downto 0);
  signal m_axi4_arid : std_logic_vector(wl_id - 1 downto 0);
  signal m_axi4_aruser : std_logic_vector(wl_user - 1 downto 0);
  signal m_axi4_arready : std_logic;

  signal m_axi4_rdata : std_logic_vector(master_data_width - 1 downto 0);
  signal m_axi4_rresp : std_logic_vector(1 downto 0);
  signal m_axi4_rid : std_logic_vector(wl_id - 1 downto 0);
  signal m_axi4_rlast : std_logic;
  signal m_axi4_rvalid : std_logic := '0';
  signal m_axi4_rready : std_logic;
begin
  -- Rest of the design goes here.

  axi4_resize_i : axi4_resize
    generic map (
      wl_s_data => slave_data_width,
      wl_m_data => master_data_width,
      wl_id => wl_id,
      wl_user => wl_user
    )
    port map (
      clk   => clk,
      rst   => rstn,
      -- Slave ports
      s_axi4_awvalid => s_axi4_awvalid,
      s_axi4_awaddr => s_axi4_awaddr,
      s_axi4_awsize => s_axi4_awsize,
      s_axi4_awburst => s_axi4_awburst,
      s_axi4_awlen => s_axi4_awlen,
      s_axi4_awcache => s_axi4_awcache,
      s_axi4_awregion => s_axi4_awregion,
      s_axi4_awqos => s_axi4_awqos,
      s_axi4_awprot => s_axi4_awprot,
      s_axi4_awlock => s_axi4_awlock,
      s_axi4_awid => s_axi4_awid,
      s_axi4_awuser => s_axi4_awuser,
      s_axi4_awready => s_axi4_awready,

      s_axi4_wdata => s_axi4_wdata,
      s_axi4_wstrb => s_axi4_wstrb,
      s_axi4_wlast => s_axi4_wlast,
      s_axi4_wvalid => s_axi4_wvalid,
      s_axi4_wready => s_axi4_wready,

      s_axi4_bvalid => s_axi4_bvalid,
```

```
s_axi4_bresp => s_axi4_bresp,
s_axi4_bid => s_axi4_bid,
s_axi4_buser => s_axi4_buser,
s_axi4_bready => s_axi4_bready,

s_axi4_arvalid => s_axi4_arvalid,
s_axi4_araddr => s_axi4_araddr,
s_axi4_arsize => s_axi4_arsize,
s_axi4_arburst => s_axi4_arburst,
s_axi4_arlen => s_axi4_arlen,
s_axi4_arcache => s_axi4_arcache,
s_axi4_arregion => s_axi4_arregion,
s_axi4_arqos => s_axi4_arqos,
s_axi4_arprot => s_axi4_arprot,
s_axi4_arlock => s_axi4_arlock,
s_axi4_arid => s_axi4_arid,
s_axi4_aruser => s_axi4_aruser,
s_axi4_arready => s_axi4_arready,

s_axi4_rdata => s_axi4_rdata,
s_axi4_rresp => s_axi4_rresp,
s_axi4_rid => s_axi4_rid,
s_axi4_rlast => s_axi4_rlast,
s_axi4_rvalid => s_axi4_rvalid,
s_axi4_rready => s_axi4_rready,

-- Master ports
m_axi4_awvalid => m_axi4_awvalid,
m_axi4_awaddr => m_axi4_awaddr,
m_axi4_awsize => m_axi4_awsize,
m_axi4_awburst => m_axi4_awburst,
m_axi4_awlen => m_axi4_awlen,
m_axi4_awcache => m_axi4_awcache,
m_axi4_awregion => m_axi4_awregion,
m_axi4_awqos => m_axi4_awqos,
m_axi4_awprot => m_axi4_awprot,
m_axi4_awlock => m_axi4_awlock,
m_axi4_awid => m_axi4_awid,
m_axi4_awuser => m_axi4_awuser,
m_axi4_awready => m_axi4_awready,

m_axi4_wdata => m_axi4_wdata,
m_axi4_wstrb => m_axi4_wstrb,
m_axi4_wlast => m_axi4_wlast,
m_axi4_wvalid => m_axi4_wvalid,
m_axi4_wready => m_axi4_wready,

m_axi4_bvalid => m_axi4_bvalid,
m_axi4_bresp => m_axi4_bresp,
m_axi4_bid => m_axi4_bid,
m_axi4_buser => m_axi4_buser,
m_axi4_bready => m_axi4_bready,

m_axi4_arvalid => m_axi4_arvalid,
m_axi4_araddr => m_axi4_araddr,
m_axi4_arsize => m_axi4_arsize,
m_axi4_arburst => m_axi4_arburst,
m_axi4_arlen => m_axi4_arlen,
m_axi4_arcache => m_axi4_arcache,
m_axi4_arregion => m_axi4_arregion,
m_axi4_arqos => m_axi4_arqos,
m_axi4_arprot => m_axi4_arprot,
m_axi4_arlock => m_axi4_arlock,
m_axi4_arid => m_axi4_arid,
m_axi4_aruser => m_axi4_aruser,
m_axi4_arready => m_axi4_arready,

m_axi4_rdata => m_axi4_rdata,
m_axi4_rresp => m_axi4_rresp,
m_axi4_rid => m_axi4_rid,
m_axi4_rlast => m_axi4_rlast,
```

```
            m_axi4_rvalid => m_axi4_rvalid,
            m_axi4_rready => m_axi4_rready
          );

    end architecture;
```

23 AXIS_BUFFER - AXI4-Stream Buffer

23.1 Overview

The AXI4-S buffer provides full timing decoupling to an AXI4-Stream interface. It can be considered a synchronous FIFO of depth 2. This buffer type is also known as a skid buffer.

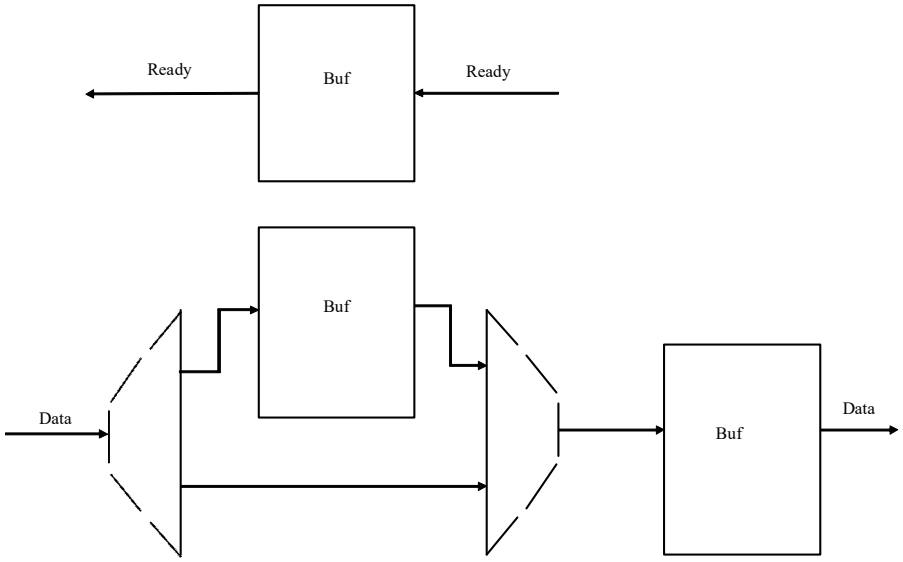


Figure 35. Figure: AXI-S buffer architecture

23.2 Operation

There are no runtime control signals to the bridge. Please refer the AXI4-Stream protocol standard document from Arm (IHI0022E) for AXI4-Stream details.

23.3 Implementation

23.3.1 Reset

The AXI4-Stream buffer resets all its registers using an active low reset. The registers are either reset synchronously or asynchronously, depending on the GRLIB grlib_async_reset_enable configuration constant.

23.3.2 Endianness

The AXI4-Stream buffer is endianness agnostic, no transformations of the input data is performed.

23.3.3 Configuration Options

The AXI4-Stream buffer can be configured at elaboration time using the generics described in this section.

*Table 177.*Configuration options

Generic name	Function	Allowed range	Default
wl	Data word length, controls the amount of data bits in the buffers. As the buffer only accepts tdata and no tuser, the user data has to be aggregated into the data stream.		1

23.4 Signal Descriptions

Table 178 shows the interface signals (VHDL ports).

*Table 178.*Signal descriptions

Signal name	Field	Type	Function	Active
rst	N/A	Input	Reset	Low
clk	N/A	Input	AXI Clock	-
s_axis_t*	*	Input/ Output	AXI4-Stream slave interface. Includes the inputs data and valid. Includes the output ready.	-
m_axis_t*	*	Input/ Output	AXI4-Stream masterinterface. Includes the input ready. Includes the outputs data and valid.	-

23.5 Library Dependencies

Table 179 shows the libraries used when instantiating the core (VHDL libraries).

*Table 179.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	config, config_type	Signals	Configuration definition and support.

23.6 Instantiation

The following code snippet is intended to show how the core is instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library gaisler;
use gaisler.axi.all; -- AXI4-Stream component

entity axis_buffer_example is
  port (
    clk : in std_logic;
    rstn : in std_logic
  );
end entity;

architecture ex of axis_buffer_example is
  constant data_width : natural := 32;

  signal s_axis_tdata : std_logic_vector(data_width - 1 downto 0);
  signal s_axis_tvalid : std_logic := '0';
  signal s_axis_tready : std_logic;
```

```
  signal m_axis_tdata : std_logic_vector(data_width - 1 downto 0);
  signal m_axis_tvalid : std_logic;
  signal m_axis_tready : std_logic := '1';
begin
  -- Rest of the design goes here.

  skid_buffer : axis_buffer
    generic map (
      wl => data_width
    )
    port map (
      clk => clk,
      rst => rstn,

      s_axis_tdata => s_axis_tdata,
      s_axis_tvalid => s_axis_tvalid,
      s_axis_tready => s_axis_tready,

      m_axis_tdata => m_axis_tdata,
      m_axis_tvalid => m_axis_tvalid,
      m_axis_tready => m_axis_tready
    );
end architecture
```

24 AXIS_GEARBOX - AXI4-Stream Gearbox

24.1 Overview

The AXI4-Stream gearbox can convert the data width of the input and output streams. The gearbox handles any input to output ratios. The tlast input can be asserted to flush the output at any time. The gearbox operates from the lowest output bits to the highest. An early flush due to an asserted tlast input might cause stale data to be present in the higher bits of the output.

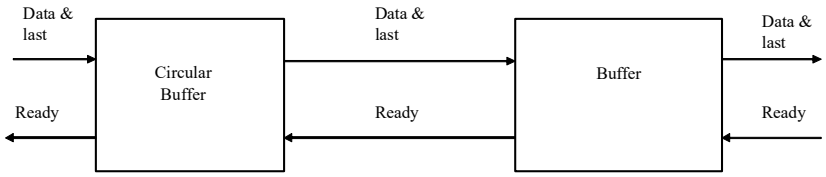


Figure 36. Figure: AXI4-S gearbox architecture

24.2 Operation

Please refer the AXI4-Stream protocol standard document from Arm (IHI0022E) for AXI4-Stream details. The gearbox operates from the lowest output bits to the highest. This means that the lowest input bits will become the first output in cases where the input is larger than the output. When the output is larger than the input, the first input will be placed in the lowest bits of the output. The gearbox offers early flushing of the circular buffer using the tlast input. An assertion of the tlast input will force the gearbox to output whatever it has in the circular buffer to the output buffer (given that the output buffer is available). The output tlast flag will be asserted for the flushed data. When a flush is performed, the circular buffer is reset and cleared. When the output is smaller than the input, the tlast flag will be asserted on the last output word created from the input.

24.3 Implementation

24.3.1 Reset

The AXI4-Stream gearbox resets all its registers using an active low reset. The registers are either reset synchronously or asynchronously, depending on the GRLIB grlib_async_reset_enable configuration constant.

24.3.2 Endianness

The AXI4-Stream gearbox is endianness agnostic.

24.3.3 Resource Usage

The circular buffer is sized as wl_in + wl_out bits. The output buffer is sized as wl_out bits. In total the gearbox consumes wl_in + 2 * wl_out register bits as a part of the data storage. Further registers are used for write and read pointers, the number depends on the wl_in to wl_out ratio.

24.4 Configuration Options

The AXI4-Stream gearbox can be configured at elaboration time using the generics described in this section.

*Table 180.*Configuration options

Generic name	Function	Allowed range	Default
wl_in	Input data word length, controls the amount of data bits in the data input.		1
wl_out	Output data word length, controls the amount of data bits in the data output.		1

24.5 Signal Descriptions

Table 181 shows the interface signals (VHDL ports).

*Table 181.*Signal descriptions

Signal name	Field	Type	Function	Active
rst	N/A	Input	Reset	Low
clk	N/A	Input	AXI Clock	-
s_axis_t*	*	Input/ Output	AXI4-Stream slave interface. Includes the inputs data, valid and last. Includes the output ready.	-
m_axis_t*	*	Input/ Output	AXI4-Stream master interface. Includes the input ready. Includes the outputs data, valid and last.	-

24.6 Library Dependencies

Table 182 shows the libraries used when instantiating the core (VHDL libraries).

*Table 182.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	config, config_type	Signals	Configuration definition and support.

24.7 Instantiation

The following code snippet is intended to show how the core is instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library gaisler;
use gaisler.axi.all; -- AXI4-Stream component

entity axis_gearbox_example is
  port (
    clk : in std_logic;
    rstn : in std_logic
  );
end entity;

architecture ex of axis_geabox_example is
  constant indata_width : natural := 32;
  constant outdata_width : natural := 8;
```

```
  signal s_axis_tdata : std_logic_vector(indata_width - 1 downto 0);
  signal s_axis_tvalid : std_logic := '0';
  signal s_axis_tlast : std_logic;
  signal s_axis_tready : std_logic;

  signal m_axis_tdata : std_logic_vector(outdata_width - 1 downto 0);
  signal m_axis_tvalid : std_logic;
  signal m_axis_tlast : std_logic;
  signal m_axis_tready : std_logic := '1';
begin
  -- Rest of the design goes here.

  gearbox : axis_gearbox
    generic map (
      wl_in  => indata_width,
      wl_out => outdata_width
    )
    port map (
      clk => clk,
      rst => rstn,

      s_axis_tdata => s_axis_tdata,
      s_axis_tvalid => s_axis_tvalid,
      s_axis_tlast => s_axis_tlast,
      s_axis_tready => s_axis_tready,

      m_axis_tdata => m_axis_tdata,
      m_axis_tvalid => m_axis_tvalid,
      m_axis_tlast => m_axis_tlast,
      m_axis_tready => m_axis_tready
    );
end architecture;
```

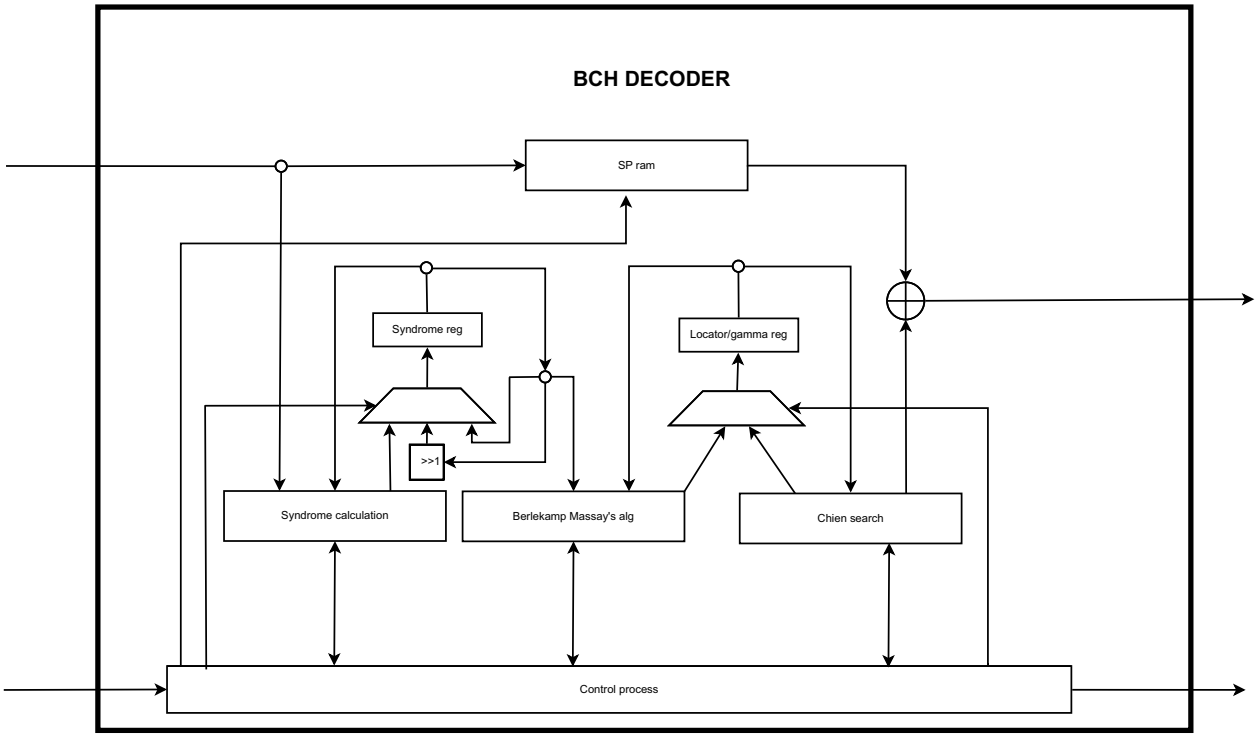
25 BCH_DECODER - Highly configurable systematic BCH decoder

25.1 Overview

The BCH decoder is expected to be used in a system together with the BCH_ENCODER core.

The BCH decoder reads an encoded data frame, including checkbits and corrects any errors. If the checkbits are computed using the BCH encoder with appropriate parametrization, it is guaranteed that the component will be able to correct all bit flips, up to a number, configured through VHDL generics.

Should there be more errors than the correctable amount, the device will identify this. In that case, the data and estimates of the number of errors should be viewed as corrupted.



25.2 Operation

25.2.1 Transaction overview

The decoder works on a transaction basis, consisting of several input and output cycles. A transaction consists of the following steps:

Table 183. Transaction overview

Step	Input signalling	Output signalling	Behaviour
1. Start transaction	bchi.start = '1' All other signals = '0'	All control outputs are '0'	Restarts the device.
2. Input data	bchi.valid = '1' bchi.codeword is the data to be decoded All other signals = '0'	All control outputs are '0'	The device accepts the inputted data and computes the syndrome. The device keeps track of how many inputs have been received and automatically starts the decoding process.
	bchi.valid = '0' All other signals = '0'	All control outputs are '0'	The device is idle, waiting for next data input.
3. Data processing	bchi.enable is don't-care. All other signals = '0'	All control outputs are '0'	Berlekamp-Massay's algorithm is running.
4.Output is ready	bchi.enable is don't-care. All other signals = '0'	bcho.out.start = '1' All other control outputs are '0'	Berlkamp-Massay's algorithm is finished and the device notifies it's ready to output decoded data.
5. Output data bits	bchi.enable = '1' All other signals = '0'	Same cycle: bcho.valid = '1' bcho.data is the corrected data bits All other control outputs are '0'	Performs Chien search on data bits. Outputs corrected data bits.
	bchi.enable = '0' All other signals = '0'	All control outputs are '0'	The device is idle, waiting for the host to accept next output.
6. Data processing	bchi.enable is don't-care. All other signals = '0'	All control outputs are '0'	Chien search is being performed on the check bits
7. End transaction	bchi.enable is don't-care All other signals = '0'	bcho.done = '1' If the decoding was successful, then bcho.nr_errors is the number of detected errors in the entire frame. bcho.cw_ok is high if and only if the frame was correctly decoded.	Ends the transaction and outputs decoding information.

25.3 Algorithm overview

The decoding is done in three steps. First step in reading input data and simultaneously compute the syndrome. In the second step the syndrome is used to compute a locator polynomial. This is done with Berlekamp-Massey's algorithm. The zeros of the locator polynomial are computed using Chien search. Because of the iterative nature of said algorithm, the corrected data is output simultaneously

as the error positions are computed. Finally, the decoding is deemed successful if and only if the number of unique zeros found in the locator polynomial is the same as the degree of said polynomial.

For resource reduction purposes, the syndrome registers are located on top level and shared between the Syndrome and Berlekamp-Massay subcomponents and the locator registers are located on top level and shared between the Berlekamp-Massay and Chien subcomponents.

25.3.1 Primitive polynomial

All data processing computations are done in a Galois field, with the size determined by the generic GF_SIZE. The primitive polynomial is chosen in the following manner

Table 184. Primitive polynomials

GF_SIZE	Max code length	Polynomial
13	8191	$x^{13} + x^8 + x^6 + x + 1$
14	16383	$x^{14} + x^4 + x^3 + x + 1$

25.3.2 Syndrome computation

Since the underlying field is the same as in the BCH encoder and the roots of the generator polynomial are $\alpha^i, i \in [1, 2 \times \text{ECC_CAPABILITY}]$, the code polynomial will be evaluated in the same points.

25.3.3 Berlekamp Massay's algorithm

The Berlekamp-Massay algorithm is used to compute the locator polynomial from the syndrome polynomial. An instantiation which can correct t errors will use $2t$ clock cycles for this computation.

25.3.4 Chien search

Since there exists no know way of efficiently finding the roots of a polynomial over a Galois field, an exhaustive search is performed. Each cycle WORD_SIZE number of possible polynomial zeros are checked. The locator polynomial is also updated in order to make it possible to reuse the exact same circuitry to check for the next WORD_SIZE values.

25.4 Implementation

25.4.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

25.5 Configuration options

Table 185 shows the configuration options of the core (VHDL generics).

Table 185. List of generics

Generic name	Function	Allowed range	Default
WORD_SIZE	The number of bits processed per cycle. Applies to bch_encode_in.data and bch_encode_out.codeword.	1 to 32	8
ECC_CAPABILITY	The maximum number of errors the code should be able to correct.	1 to 60	60
GF_SIZE	The logarithm of the size of the underlying field.	13,14	14
DATA_BITS	Number of databits (i.e. not counting checkbits)	1 to $2^{GF_SIZE} - GF_SIZE \times ECC_CAPABILITY - 1$	8192
TECH	Target technology	0 to NTECH	INFERRED(0)

The number of expected check bits will be $GF_SIZE \times ECC_CAPABILITY$. Should the number of code bits not be a multiple of WORD_SIZE, then the last input is expected to be zeropadded.

Note that the number of data bits does not have to be a multiple of WORD_SIZE. If it isn't, then check bits and data bits will be in the same input word.

If this component is used in conjunction with BCH_ENCODER, all three generics are expected to match with the corresponding generics in the decoder block.

The total number of bits, including both data and check bits, must be no more than $2^{GF_SIZE} - 1$.

25.6 Signal description

Table 186 shows the interface signals of the core (VHDL ports).

Table 186: Signal list

Signal name	Field	Type	Function	Active
rstn	N/A	Input	Reset	Low
clk	N/A	Input	Clock	-
BCHI	VALID	Input	Valid data	High
	START	Input	Starts a new transaction	High
	ENABLE	Input	Enables output of check bits	High
	CODEWORD(31:0)	Input	Data input. Lower indices used if WORD_SIZE < 32	-
BCHO	CW_OK	Output	All errors were correctable	High
	NR_ERRORS	Output	Number of detected errors in the codeword	
	VALID	Output	Output data is valid	High
	START	Output	First data indicator	High
	DONE	Output	Transaction completed	High
	DATA(31:0)	Output	Data output. Lower indices used if WORD_SIZE < 32	-

25.7 Library dependencies

The table below shows the required libraries for instatntiation fo hte BCH_DECODER.

Table 187. Library dependencies

Libary	Package	Imported units	Description
GAISLER	BCH_EXTERNAL_PKG	Signals	Port declarations
GAISLER	BCH_INTERNAL_PKG	Constants, functions	Support functions, constants, type declarations

25.8 Instantiation

This is an example of how BCH_ENCODER and BCH_DECODER may be used together.

```
signal enc_in  : gaisler.bch_external_pkg.bch_encoder_in_type;
signal enc_out : gaisler.bch_external_pkg.bch_encoder_out_type;
signal dec_in  : gaisler.bch_external_pkg.bch_decoder_in_type;
signal dec_out : gaisler.bch_external_pkg.bch_decoder_out_type;

--------------
--------------
--------------

begin
------------- Generate
------------- data
------------- here
encoder_0: entity gaisler.bch_encoder
generic map (
        GF_SIZE => 14,
        WORD_SIZE => 8,
        ECC_CAPABILITY => 60
)
port map (
        clk         => clk,
        rstn        => rstn,
        bchi => enc_in,
        bcho => enc_out
);

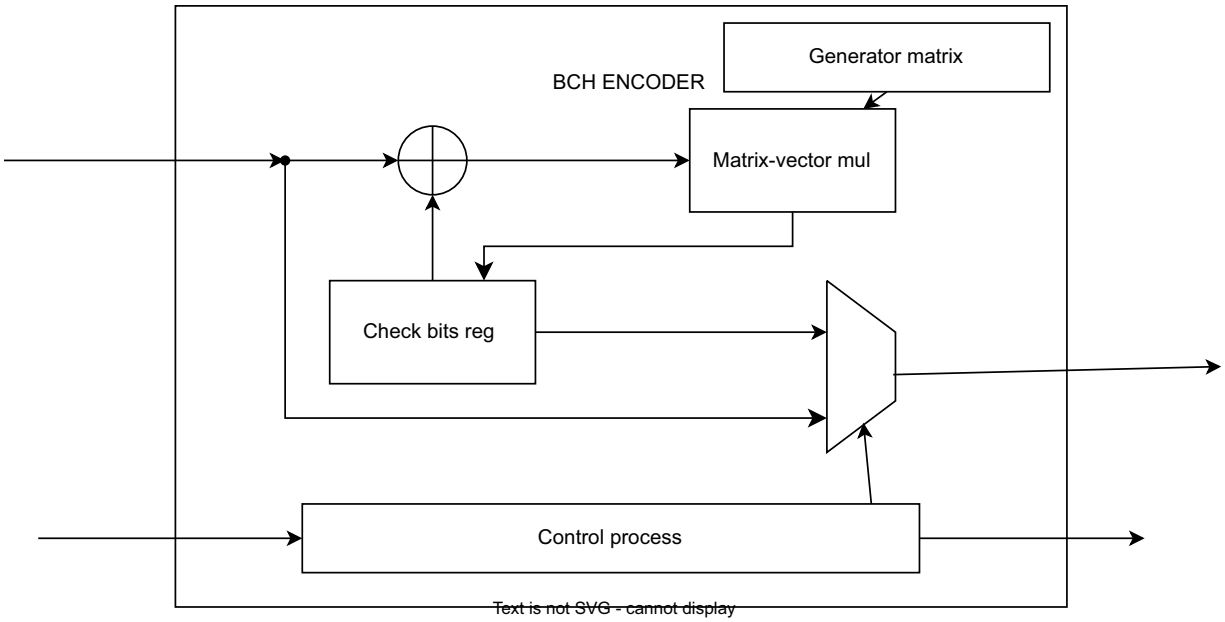
dec_in <= (
        valid   => enc_out.valid,
        start   => enc_in.start,
        enable  => '1',
        codeword => enc_out.codeword
);

decoder_0 : entity gaisler.bch_decoder
generic map (
        GF_SIZE          => 14,
        WORD_SIZE        => 8,
        ECC_CAPABILITY   => 60,
        DATA_BITS => 8192
)
port map (
        rstn               => rstn,
        clk                => clk,
        bchi => dec_in,
        bch0 => dec_out
);
```

26 BCH_ENCODER - Highly configurable systematic BCH encoder

26.1 Overview

The BCH encoder accepts a data frame of configurable length and computes check bits. The check bits are computed in such a way that it is guaranteed that BCH_DECODER can correct up to and including a number of independent bit flips, which is configured at implementation time via VHDL generics.



26.2 Operation

26.2.1 Transaction overview

The encoder works on a transaction basis, consisting of several input and output cycles. A transaction consists of the following steps:

Table 188. Transaction overview

Step	Input signalling	Output signalling	Behaviour
1.Start	bchi.start = '1' All other signals = '0'	All outputs are '0'	Restarts the device.
2. Input data	bchi.valid = '1' bchi.data is the data to be encoded bchi.enable is don't-care All other signals = '0'	Next cycle: bcho.valid = '1' bcho.codeword is the data from the previous cycle All other signals = '0'	The device accepts the input data. During the next clk cycle, the same data will be outputted regardless of other signaling.
	bchi.valid = '0' bchi.data and bchi.enable are don't-care. All other signals = '0'	Next cycle: bcho.valid = '0' bcho.codeword is undefined All other signals = '0'	The device ignores the input data this cycle.
3. Stop input	bchi.stop = '1' All other signals = '0'	Possible valid and codeword from input in the previous cycle.	Marks end of data. After this, the device will be ready to output check bits.
4. Output check bits	bchi.enable = '1' All other signals = '0'	Same cycle: bcho.valid = '1' bcho.codeword is checkbits computed by the device.	Outputs checkbits, controlled by the enable flag on the input.
	bchi.enable = '0' All other signals = '0'		The holds off transmitting check bits until next time bch_encoder_in.enable is high.
5. End transaction	bchi.enable is don't-care All other signals = '0'	bch_encoder_out.done = '1'	Ends the transaction

All signals will be sampled on every clock cycle, i.e. bch_encoder_in.start and stop should be high for exactly one cycle to start or stop input. In a similar manner, bch_encoder_out.done will be high for exactly one cycle to signal finished transaction.

The component computes checkbits in accordance with a BCH coding scheme.

26.2.2 Input restrictions

Data may only be input during a transaction. In practice this means:

All bchi.valid pulses must come after bchi.start and before bchi.stop.

Should bchi.start be high in the middle of a transaction, all calculated checkbits will be discarded and a new transaction initialized.

26.2.3 Primitive polynomial

The underlying field and hence the corresponding primitive polynomial is chosen based on the generic GF_SIZE in the following manner

Table 189. Primitive polynomial

GF_SIZE	Max code length	Polynomial
13	8191	$x^{13} + x^8 + x^6 + x + 1$
14	16383	$x^{14} + x^4 + x^3 + x + 1$

26.2.4 Generator

The choice of generator polynomial is dependent on the underlying field and the error correction capability. If the code is designed to correct t errors, then $g(x) = LCM(m_1, m_2, ..., m_{2t})$ over GF(2) where $m_i(x)$ is the minimal polynomial of α^i where α is a primitive element in the field.

Since the code is systematic, the generator matrix $G = (I|A)$ where I is the quadratic identity matrix and A is chosen such that the code is equivalent to one generated by simply multiplying with the generator polynomial. Note that A is unique.

26.3 Implementation

26.3.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

26.4 Configuration options

Table 190 shows the configuration options of the core (VHDL generics).

Table 190. Generic options

Generic name	Function	Allowed range	Default
WORD_SIZE	The number of bits processed per cycle. Applies to bch_encode_in.data and bch_encode_out.codeword.	1-32	8
ECC_CAPABILITY	The maximum number of errors the code should be able to correct.	1-60	60
GF_SIZE	The logarithm of the size of the underlying field.	13,14	14

The number of check bits generated will be $\text{GF_SIZE} \times \text{ECC_CAPABILITY}$. Should this not be a multiple of WORD_SIZE, the bottom bits of the last output will be padded with zeros.

If this component is used in conjunction with BCH_DECODER, all three generics are expected to agree with the corresponding generics in the decoder block.

The total number of bits, including both data and check bits, must be no more than $2^{\text{GF_SIZE}} - 1$.

26.5 Signal description

Table 191 shows the interface signals of the core (VHDL ports).

Table 191: Signal list

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
BCHI	VALID	Input	Valid data	High
	START	Input	Starts a new transaction	High
	ENABLE	Input	Enables output of check bits	High
	STOP	Input	Indicates last input has been sent	High
	DATA(31:0)	Input	Data input. Lower indices used if WORD_SIZE < 32	-
BCHO	VALID	Output	Output codeword is valid	High
	DONE	Output	Last check data indicator	High
	CODEWORD(31:0)	Output	Data output. Lower indices used if WORD_SIZE < 32	-

26.6 Library dependencies

Table 192. Library dependencies

Libary	Package	Imported units	Description
GAISLER	BCH_EXTERNAL_PKG	Signals	Port declarations
GAISLER	BCH_INTERNAL_PKG	Constants, functions	Support functions, generator polynomial

26.7 Instantiation

This is an example of how BCH_ENCODER and BCH_DECODER may be used together.

```
signal enc_in  : gaisler.bch_external_pkg.bch_encoder_in_type;
signal enc_out : gaisler.bch_external_pkg.bch_encoder_out_type;
signal dec_in  : gaisler.bch_external_pkg.bch_decoder_in_type;
signal dec_out : gaisler.bch_external_pkg.bch_decoder_out_type;

--------------
--------------
--------------

begin
------------- Generate
------------- data
------------- here
encoder_0: entity gaisler.bch_encoder
generic map (
        GF_SIZE => 14,
        WORD_SIZE => 8,
        ECC_CAPABILITY => 60
)
port map (
        clk         => clk,
        rstn        => rstn,
        bchi => enc_in,
        bcho => enc_out
);

dec_in <= (
        valid   => enc_out.valid,
```

```
        start   => enc_in.start,
        enable  => '1',
        codeword => enc_out.codeword
);

decoder_0 : entity gaisler.bch_decoder
generic map (
        GF_SIZE          => 14,
        WORD_SIZE        => 8,
        ECC_CAPABILITY   => 60,
        DATA_BITS => 8192
)
port map (
        rstn               => rstn,
        clk                => clk,
        bchi => dec_in,
        bcho => dec_out
);
```

27 CAN_OC - GRLIB wrapper for OpenCores CAN Interface core

27.1 Overview

CAN_OC is GRLIB wrapper for the CAN core from Opencores. It provides a bridge between AMBA AHB and the CAN Core registers. The AHB slave interface is mapped in the AHB I/O space using the GRLIB plug&play functionality. The CAN core interrupt is routed to the AHB interrupt bus, and the interrupt number is selected through the *irq* generic. The FIFO RAM in the CAN core is implemented using the GRLIB parametrizable SYNCRAM_2P memories, assuring portability to all supported technologies.

This CAN interface implements the CAN 2.0A and 2.0B protocols. The functionality and operational modes of CAN_OC are based on the Philips SJA1000 CAN controller IC, and has a compatible register map with a few exceptions.

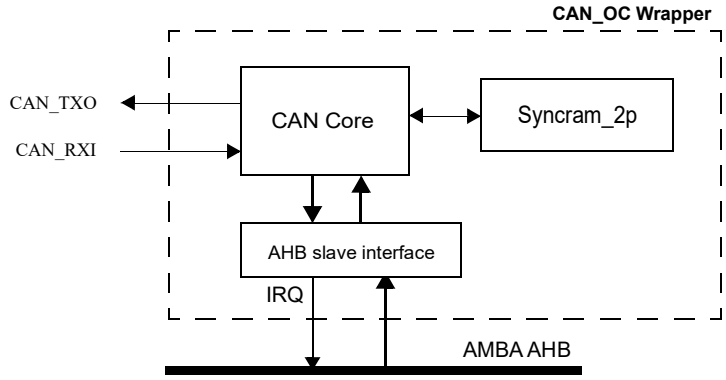


Figure 37. Block diagram

27.2 Opencores CAN controller overview

This CAN controller is based on the Philips SJA1000 and has a compatible register map with a few exceptions. It also supports both BasicCAN (PCA82C200 like) and PeliCAN mode. In PeliCAN mode the extended features of CAN 2.0B are supported. The mode of operation is chosen through the Clock Divider register.

This document lists the registers and their functionality. The Philips SJA1000 data sheet can be used as a reference if further clarification is necessary regarding the register map and the functionality implemented in the IP. However, other sections of the SJA1000 datasheet such as those related to timing and electrical parameters, pinout and package information do not apply to CAN_OC. The differences in functionality and register map between CAN_OC and SJA1000 can be found in the Design considerations section later in this document.

The register map and functionality is different between the two modes of operation. First the BasicCAN mode will be described followed by PeliCAN. Common registers (clock divisor and bus timing) are described in a separate chapter. The register map also differs depending on whether the core is in operating mode or in reset mode. When reset the core starts in reset mode awaiting configuration. Operating mode is entered by clearing the reset request bit in the command register. To re-enter reset mode set this bit high again.

27.3 AHB interface

All registers are one byte wide and the addresses specified in this document are byte addresses. Byte reads and writes should be used when interfacing with this core. The read byte is duplicated on all byte lanes of the AHB bus. The wrapper is big endian so the core expects the MSB at the lowest address.

The bit numbering in this document uses bit 7 as MSB and bit 0 as LSB.

The core is designed for big-endian systems.

27.4 BasicCAN mode

27.4.1 BasicCAN register map

Table 193. BasicCAN address allocation

Address	Operating mode		Reset mode	
	Read	**Write**	**Read**	**Write**
0	Control	Control	Control	Control
1	(0xFF)	Command	(0xFF)	Command
2	Status	-	Status	-
3	Interrupt	-	Interrupt	-
4	(0xFF)	-	Acceptance code	Acceptance code
5	(0xFF)	-	Acceptance mask	Acceptance mask
6	(0xFF)	-	Bus timing 0	Bus timing 0
7	(0xFF)	-	Bus timing 1	Bus timing 1
8	(0x00)	-	(0x00)	-
9	(0x00)	-	(0x00)	-
10	TX id1	TX id1	(0xFF)	-
11	TX id2, rtr, dlc	TX id2, rtr, dlc	(0xFF)	-
12	TX data byte 1	TX data byte 1	(0xFF)	-
13	TX data byte 2	TX data byte 2	(0xFF)	-
14	TX data byte 3	TX data byte 3	(0xFF)	-
15	TX data byte 4	TX data byte 4	(0xFF)	-
16	TX data byte 5	TX data byte 5	(0xFF)	-
17	TX data byte 6	TX data byte 6	(0xFF)	-
18	TX data byte 7	TX data byte 7	(0xFF)	-
19	TX data byte 8	TX data byte 8	(0xFF)	-
20	RX id1	-	RX id1	-
21	RX id2, rtr, dlc	-	RX id2, rtr, dlc	-
22	RX data byte 1	-	RX data byte 1	-
23	RX data byte 2	-	RX data byte 2	-
24	RX data byte 3	-	RX data byte 3	-
25	RX data byte 4	-	RX data byte 4	-
26	RX data byte 5	-	RX data byte 5	-
27	RX data byte 6	-	RX data byte 6	-
28	RX data byte 7	-	RX data byte 7	-
29	RX data byte 8	-	RX data byte 8	-
30	(0x00)	-	(0x00)	-
31	Clock divider	Clock divider	Clock divider	Clock divider

27.4.2 Control register

The control register contains interrupt enable bits as well as the reset request bit.

Table 194. Bit interpretation of control register (CR) (address 0)

Bit	Name	Description	Reset value
CR.7	-	reserved	0
CR.6	-	reserved	0
CR.5	-	reserved (reads as 1)	1
CR.4	Overrun Interrupt Enable	1 - enabled, 0 - disabled	0
CR.3	Error Interrupt Enable	1 - enabled, 0 - disabled	0
CR.2	Transmit Interrupt Enable	1 - enabled, 0 - disabled	0
CR.1	Receive Interrupt Enable	1 - enabled, 0 - disabled	0
CR.0	Reset request	Writing 1 to this bit aborts any ongoing transfer and enters reset mode. Writing 0 returns to operating mode.	1

27.4.3 Command register

Writing a one to the corresponding bit in this register initiates an action supported by the core.

Table 195. Bit interpretation of command register (CMR) (address 1)

Bit	Name	Description	Reset value
CMR.7	-	reserved	1
CMR.6	-	reserved	1
CMR.5	-	reserved	1
CMR.4	-	not used (go to sleep in SJA1000 core)	1
CMR.3	Clear data overrun	Clear the data overrun status bit	1
CMR.2	Release receive buffer	Free the current receive buffer for new reception	1
CMR.1	Abort transmission	Aborts a not yet started transmission.	1
CMR.0	Transmission request	Starts the transfer of the message in the TX buffer	1

A transmission is started by writing 1 to CMR.0. It can only be aborted by writing 1 to CMR.1 and only if the transfer has not yet started. If the transmission has started it will not be aborted when setting CMR.1 but it will not be retransmitted if an error occurs.

Giving the Release receive buffer command should be done after reading the contents of the receive buffer in order to release this memory. If there is another message waiting in the FIFO a new receive interrupt will be generated (if enabled) and the receive buffer status bit will be set again.

To clear the Data overrun status bit CMR.3 must be written with 1.

27.4.4 Status register

The status register is read only and reflects the current status of the core.

Table 196. Bit interpretation of status register (SR) (address 2)

Bit	Name	Description	Reset value
SR.7	Bus status	1 when the core is in bus-off and not involved in bus activities	0
SR.6	Error status	At least one of the error counters have reached or exceeded the CPU warning limit (96).	0
SR.5	Transmit status	1 when transmitting a message	0
SR.4	Receive status	1 when receiving a message	0
SR.3	Transmission complete	1 indicates the last message was successfully transferred.	1
SR.2	Transmit buffer status	1 means CPU can write into the transmit buffer	1
SR.1	Data overrun status	1 if a message was lost because no space in fifo.	0
SR.0	Receive buffer status	1 if messages available in the receive fifo.	0

Receive buffer status is cleared when the Release receive buffer command is given and set high if there are more messages available in the fifo.

The data overrun status signals that a message which was accepted could not be placed in the fifo because not enough space left. NOTE: This bit differs from the SJA1000 behavior and is set first when the fifo has been read out.

When the transmit buffer status is high the transmit buffer is available to be written into by the CPU. During an on-going transmission the buffer is locked and this bit is 0.

The transmission complete bit is set to 0 when a transmission request has been issued and will not be set to 1 again until a message has successfully been transmitted.

27.4.5 Interrupt register

The interrupt register signals to CPU what caused the interrupt. The interrupt bits are only set if the corresponding interrupt enable bit is set in the control register.

Table 197. Bit interpretation of interrupt register (IR) (address 3)

Bit	Name	Description	Reset value
IR.7	-	reserved (reads as 1)	1
IR.6	-	reserved (reads as 1)	1
IR.5	-	reserved (reads as 1)	1
IR.4	-	not used (wake-up interrupt of SJA1000)	0
IR.3	Data overrun interrupt	Set when SR.1 goes from 0 to 1.	0
IR.2	Error interrupt	Set when the error status or bus status are changed.	0
IR.1	Transmit interrupt	Set when the transmit buffer is released (status bit 0->1)	0
IR.0	Receive interrupt	This bit is set while there are more messages in the fifo.	0

This register is reset on read with the exception of IR.0. Note that this differs from the SJA1000 behavior where all bits are reset on read in BasicCAN mode. This core resets the receive interrupt bit when the release receive buffer command is given (like in PeliCAN mode).

Also note that bit IR.5 through IR.7 reads as 1 but IR.4 is 0.

27.4.6 Transmit buffer

The table below shows the layout of the transmit buffer. In BasicCAN only standard frame messages can be transmitted and received (EFF messages on the bus are ignored).

Table 198. Transmit buffer layout

Addr	Name	Bits							
		7	6	5	4	3	2	1	0
10	ID byte 1	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
11	ID byte 2	ID.2	ID.1	ID.0	RTR	DLC.3	DLC.2	DLC.1	DLC.0
12	TX data 1	TX byte 1							
13	TX data 2	TX byte 2							
14	TX data 3	TX byte 3							
15	TX data 4	TX byte 4							
16	TX data 5	TX byte 5							
17	TX data 6	TX byte 6							
18	TX data 7	TX byte 7							
19	TX data 8	TX byte 8							

If the RTR bit is set no data bytes will be sent but DLC is still part of the frame and must be specified according to the requested frame. Note that it is possible to specify a DLC larger than 8 bytes but should not be done for compatibility reasons. If DLC > 8 still only 8 bytes can be sent.

27.4.7 Receive buffer

The receive buffer on address 20 through 29 is the visible part of the 64 byte RX FIFO. Its layout is identical to that of the transmit buffer.

27.4.8 Acceptance filter

Messages can be filtered based on their identifiers using the acceptance code and acceptance mask registers. The top 8 bits of the 11 bit identifier are compared with the acceptance code register only comparing the bits set to zero in the acceptance mask register. If a match is detected the message is stored to the fifo.

27.5 PeliCAN mode

27.5.1 PeliCAN register map

Table 199. PeliCAN address allocation

#	Operating mode				Reset mode	
	Read		**Write**		**Read**	**Write**
0	Mode		Mode		Mode	Mode
1	(0x00)		Command		(0x00)	Command
2	Status		-		Status	-
3	Interrupt		-		Interrupt	-
4	Interrupt enable		Interrupt enable		Interrupt enable	Interrupt enable
5	reserved (0x00)		-		reserved (0x00)	-
6	Bus timing 0		-		Bus timing 0	Bus timing 0
7	Bus timing 1		-		Bus timing 1	Bus timing 1
8	(0x00)		-		(0x00)	-
9	(0x00)		-		(0x00)	-
10	reserved (0x00)		-		reserved (0x00)	-
11	Arbitration lost capture		-		Arbitration lost capture	-
12	Error code capture		-		Error code capture	-
13	Error warning limit		-		Error warning limit	Error warning limit
14	RX error counter		-		RX error counter	RX error counter
15	TX error counter		-		TX error counter	TX error counter
16	RX FI SFF	RX FI EFF	TX FI SFF	TX FI EFF	Acceptance code 0	Acceptance code 0
17	RX ID 1	RX ID 1	TX ID 1	TX ID 1	Acceptance code 1	Acceptance code 1
18	RX ID 2	RX ID 2	TX ID 2	TX ID 2	Acceptance code 2	Acceptance code 2
19	RX data 1	RX ID 3	TX data 1	TX ID 3	Acceptance code 3	Acceptance code 3
20	RX data 2	RX ID 4	TX data 2	TX ID 4	Acceptance mask 0	Acceptance mask 0
21	RX data 3	RX data 1	TX data 3	TX data 1	Acceptance mask 1	Acceptance mask 1
22	RX data 4	RX data 2	TX data 4	TX data 2	Acceptance mask 2	Acceptance mask 2
23	RX data 5	RX data 3	TX data 5	TX data 3	Acceptance mask 3	Acceptance mask 3
24	RX data 6	RX data 4	TX data 6	TX data 4	reserved (0x00)	-
25	RX data 7	RX data 5	TX data 7	TX data 5	reserved (0x00)	-
26	RX data 8	RX data 6	TX data 8	TX data 6	reserved (0x00)	-
27	FIFO	RX data 7	-	TX data 7	reserved (0x00)	-
28	FIFO	RX data 8	-	TX data 8	reserved (0x00)	-
29	RX message counter		-		RX msg counter	-
30	(0x00)		-		(0x00)	-
31	Clock divider		Clock divider		Clock divider	Clock divider

The transmit and receive buffers have different layout depending on if standard frame format (SFF) or extended frame format (EFF) is to be transmitted/received. See the specific section below.

27.5.2 Mode register

Table 200. Bit interpretation of mode register (MOD) (address 0)

Bit	Name	Description	Reset value
MOD.7	-	reserved	0
MOD.6	-	reserved	0
MOD.5	-	reserved	0
MOD.4	-	not used (sleep mode in SJA1000)	0
MOD.3	Acceptance filter mode	1 - single filter mode, 0 - dual filter mode	0
MOD.2	Self test mode	If set the controller is in self test mode	0
MOD.1	Listen only mode	If set the controller is in listen only mode	0
MOD.0	Reset mode	Writing 1 to this bit aborts any ongoing transfer and enters reset mode. Writing 0 returns to operating mode	1

Writing to MOD.1-3 can only be done when reset mode has been entered previously.

In Listen only mode the core will not send any acknowledgements. Note that unlike the SJA1000 the Opencores core does not become error passive and active error frames are still sent!

When in Self test mode the core can complete a successful transmission without getting an acknowledgement if given the Self reception request command. Note that the core must still be connected to a real bus, it does not do an internal loopback.

27.5.3 Command register

Writing a one to the corresponding bit in this register initiates an action supported by the core.

Table 201. Bit interpretation of command register (CMR) (address 1)

Bit	Name	Description	Reset value
CMR.7	-	reserved	0
CMR.6	-	reserved	0
CMR.5	-	reserved	0
CMR.4	Self reception request	Transmits and simultaneously receives a message	0
CMR.3	Clear data overrun	Clears the data overrun status bit	0
CMR.2	Release receive buffer	Free the current receive buffer for new reception	0
CMR.1	Abort transmission	Aborts a not yet started transmission.	0
CMR.0	Transmission request	Starts the transfer of the message in the TX buffer	0

A transmission is started by writing 1 to CMR.0. It can only be aborted by writing 1 to CMR.1 and only if the transfer has not yet started. Setting CMR.0 and CMR.1 simultaneously will result in a so called single shot transfer, i.e. the core will not try to retransmit the message if not successful the first time.

Giving the Release receive buffer command should be done after reading the contents of the receive buffer in order to release this memory. If there is another message waiting in the FIFO a new receive interrupt will be generated (if enabled) and the receive buffer status bit will be set again.

The Self reception request bit together with the self test mode makes it possible to do a self test of the core without any other cores on the bus. A message will simultaneously be transmitted and received and both receive and transmit interrupt will be generated.

27.5.4 Status register

The status register is read only and reflects the current status of the core.

*Table 202.*Bit interpretation of command register (SR) (address 2)

Bit	Name	Description	Reset value
SR.7	Bus status	1 when the core is in bus-off and not involved in bus activities	0
SR.6	Error status	At least one of the error counters have reached or exceeded the error warning limit.	0
SR.5	Transmit status	1 when transmitting a message	0
SR.4	Receive status	1 when receiving a message	0
SR.3	Transmission complete	1 indicates the last message was successfully transferred.	1
SR.2	Transmit buffer status	1 means CPU can write into the transmit buffer	1
SR.1	Data overrun status	1 if a message was lost because no space in fifo.	0
SR.0	Receive buffer status	1 if messages available in the receive fifo.	0

Receive buffer status is cleared when there are no more messages in the fifo. The data overrun status signals that a message which was accepted could not be placed in the fifo because not enough space left. NOTE: This bit differs from the SJA1000 behavior and is set first when the fifo has been read out.

When the transmit buffer status is high the transmit buffer is available to be written into by the CPU. During an on-going transmission the buffer is locked and this bit is 0.

The transmission complete bit is set to 0 when a transmission request or self reception request has been issued and will not be set to 1 again until a message has successfully been transmitted.

27.5.5 Interrupt register

The interrupt register signals to CPU what caused the interrupt. The interrupt bits are only set if the corresponding interrupt enable bit is set in the interrupt enable register.

*Table 203.*Bit interpretation of interrupt register (IR) (address 3)

Bit	Name	Description	Reset value
IR.7	Bus error interrupt	Set if an error on the bus has been detected	0
IR.6	Arbitration lost interrupt	Set when the core has lost arbitration	0
IR.5	Error passive interrupt	Set when the core goes between error active and error passive	0
IR.4	-	not used (wake-up interrupt of SJA1000)	0
IR.3	Data overrun interrupt	Set when data overrun status bit is set	0
IR.2	Error warning interrupt	Set on every change of the error status or bus status	0
IR.1	Transmit interrupt	Set when the transmit buffer is released	0
IR.0	Receive interrupt	Set while the fifo is not empty.	0

This register is reset on read with the exception of IR.0 which is reset when the fifo has been emptied.

27.5.6 Interrupt enable register

In the interrupt enable register the separate interrupt sources can be enabled/disabled. If enabled the corresponding bit in the interrupt register can be set and an interrupt generated. This register does not have a reset value.

*Table 204.*Bit interpretation of interrupt enable register (IER) (address 4)

Bit	Name	Description
IR.7	Bus error interrupt	1 - enabled, 0 - disabled
IR.6	Arbitration lost interrupt	1 - enabled, 0 - disabled
IR.5	Error passive interrupt	1 - enabled, 0 - disabled
IR.4	-	not used (wake-up interrupt of SJA1000)
IR.3	Data overrun interrupt	1 - enabled, 0 - disabled
IR.2	Error warning interrupt	1 - enabled, 0 - disabled.
IR.1	Transmit interrupt	1 - enabled, 0 - disabled
IR.0	Receive interrupt	1 - enabled, 0 - disabled

27.5.7 Arbitration lost capture register

*Table 205.*Bit interpretation of arbitration lost capture register (ALC) (address 11)

Bit	Name	Description	Reset value
ALC.7-5	-	reserved	0
ALC.4-0	Bit number	Bit where arbitration is lost	0

When the core loses arbitration the bit position of the bit stream processor is captured into arbitration lost capture register. The register will not change content again until read out.

27.5.8 Error code capture register

*Table 206.*Bit interpretation of error code capture register (ECC) (address 12)

Bit	Name	Description	Reset value
ECC.7-6	Error code	Error code number	0
ECC.5	Direction	1 - Reception, 0 - transmission error	0
ECC.4-0	Segment	Where in the frame the error occurred	0

When a bus error occurs the error code capture register is set according to what kind of error occurred, if it was while transmitting or receiving and where in the frame it happened. As with the ALC register the ECC register will not change value until it has been read out. The table below shows how to interpret bit 7-6 of ECC.

*Table 207.*Error code interpretation

ECC.7-6	Description
0	Bit error
1	Form error
2	Stuff error
3	Other

Bit 4 downto 0 of the ECC register is interpreted as below

*Table 208.*Bit interpretation of ECC.4-0

ECC.4-0	Description
0x03	Start of frame
0x02	ID.28 - ID.21
0x06	ID.20 - ID.18
0x04	Bit SRTR
0x05	Bit IDE
0x07	ID.17 - ID.13
0x0F	ID.12 - ID.5
0x0E	ID.4 - ID.0
0x0C	Bit RTR
0x0D	Reserved bit 1
0x09	Reserved bit 0
0x0B	Data length code
0x0A	Data field
0x08	CRC sequence
0x18	CRC delimiter
0x19	Acknowledge slot
0x1B	Acknowledge delimiter
0x1A	End of frame
0x12	Intermission
0x11	Active error flag
0x16	Passive error flag
0x13	Tolerate dominant bits
0x17	Error delimiter
0x1C	Overload flag

27.5.9 Error warning limit register

This registers allows for setting the CPU error warning limit. It defaults to 96. Note that this register is only writable in reset mode.

27.5.10 RX error counter register (address 14)

This register shows the value of the rx error counter. It is writable in reset mode. A bus-off event resets this counter to 0.

27.5.11 TX error counter register (address 15)

This register shows the value of the tx error counter. It is writable in reset mode. If a bus-off event occurs this register is initialized as to count down the protocol defined 128 occurrences of the bus-free signal and the status of the bus-off recovery can be read out from this register. The CPU can force a bus-off by writing 255 to this register. Note that unlike the SJA1000 this core will signal bus-off immediately and not first when entering operating mode. The bus-off recovery sequence starts when entering operating mode after writing 255 to this register in reset mode.

27.5.12 Transmit buffer

The transmit buffer is write-only and mapped on address 16 to 28. Reading of this area is mapped to the receive buffer described in the next section. The layout of the transmit buffer depends on whether a standard frame (SFF) or an extended frame (EFF) is to be sent as seen below.

Table 209.

#	Write (SFF)	Write(EFF)
16	TX frame information	TX frame information
17	TX ID 1	TX ID 1
18	TX ID 2	TX ID 2
19	TX data 1	TX ID 3
20	TX data 2	TX ID 4
21	TX data 3	TX data 1
22	TX data 4	TX data 2
23	TX data 5	TX data 3
24	TX data 6	TX data 4
25	TX data 7	TX data 5
26	TX data 8	TX data 6
27	-	TX data 7
28	-	TX data 8

TX frame information (this field has the same layout for both SFF and EFF frames)

*Table 210.*TX frame information address 16

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FF	RTR	-	-	DLC.3	DLC.2	DLC.1	DLC.0

Bit 7 - FF selects the frame format, i.e. whether this is to be interpreted as an extended or standard frame. 1 = EFF, 0 = SFF.
Bit 6 - RTR should be set to 1 for an remote transmission request frame.
Bit 5:4 - are don't care.
Bit 3:0 - DLC specifies the Data Length Code and should be a value between 0 and 8. If a value greater than 8 is used 8 bytes will be transmitted.

TX identifier 1 (this field is the same for both SFF and EFF frames)

*Table 211.*TX identifier 1 address 17

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Bit 7:0 - The top eight bits of the identifier.

TX identifier 2, SFF frame

*Table 212.*TX identifier 2 address 18

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.20	ID.19	ID.18	-	-	-	-	-

Bit 7:5 - Bottom three bits of an SFF identifier.
Bit 4:0 - Don't care.

TX identifier 2, EFF frame

Table 213. TX identifier 2 address 18

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Bit 7:0 - Bit 20 downto 13 of 29 bit EFF identifier.

TX identifier 3, EFF frame

Table 214. TX identifier 3 address 19

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

Bit 7:0 - Bit 12 downto 5 of 29 bit EFF identifier.

TX identifier 4, EFF frame

Table 215. TX identifier 4 address 20

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.4	ID.3	ID.2	ID.1	ID.0	-	-	-

Bit 7:3 - Bit 4 downto 0 of 29 bit EFF identifier
Bit 2:0 - Don’t care

Data field

For SFF frames the data field is located at address 19 to 26 and for EFF frames at 21 to 28. The data is transmitted starting from the MSB at the lowest address.

27.5.13 Receive buffer

Table 216.

#	Read (SFF)	Read (EFF)
16	RX frame information	RX frame information
17	RX ID 1	RX ID 1
18	RX ID 2	RX ID 2
19	RX data 1	RX ID 3
20	RX data 2	RX ID 4
21	RX data 3	RX data 1
22	RX data 4	RX data 2
23	RX data 5	RX data 3
24	RX data 6	RX data 4
25	RX data 7	RX data 5
26	RX data 8	RX data 6
27	RX FI of next message in fifo	RX data 7
28	RX ID1 of next message in fifo	RX data 8

RX frame information (this field has the same layout for both SFF and EFF frames)

*Table 217.*RX frame information address 16

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FF	RTR	0	0	DLC.3	DLC.2	DLC.1	DLC.0

Bit 7 - Frame format of received message. 1 = EFF, 0 = SFF.
Bit 6 - 1 if RTR frame.
Bit 5:4 - Always 0.
Bit 3:0 - DLC specifies the Data Length Code.

RX identifier 1(this field is the same for both SFF and EFF frames)

*Table 218.*RX identifier 1 address 17

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Bit 7:0 - The top eight bits of the identifier.

RX identifier 2, SFF frame

*Table 219.*RX identifier 2 address 18

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.20	ID.19	ID.18	RTR	0	0	0	0

Bit 7:5 - Bottom three bits of an SFF identifier.
Bit 4 - 1 if RTR frame.
Bit 3:0 - Always 0.

RX identifier 2, EFF frame

*Table 220.*RX identifier 2 address 18

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Bit 7:0 - Bit 20 downto 13 of 29 bit EFF identifier.

RX identifier 3, EFF frame

*Table 221.*RX identifier 3 address 19

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

Bit 7:0 - Bit 12 downto 5 of 29 bit EFF identifier.

RX identifier 4, EFF frame

*Table 222.*RX identifier 4 address 20

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.4	ID.3	ID.2	ID.1	ID.0	RTR	0	0

Bit 7:3 - Bit 4 downto 0 of 29 bit EFF identifier
Bit 2- 1 if RTR frame
Bit 1:0 - Don’t care

Data field

For received SFF frames the data field is located at address 19 to 26 and for EFF frames at 21 to 28.

27.5.14 Acceptance filter

The acceptance filter can be used to filter out messages not meeting certain demands. If a message is filtered out it will not be put into the receive fifo and the CPU will not have to deal with it.

There are two different filtering modes, single and dual filter. Which one is used is controlled by bit 3 in the mode register. In single filter mode only one 4 byte filter is used. In dual filter two smaller filters are used and if either of these signals a match the message is accepted. Each filter consists of two parts the acceptance code and the acceptance mask. The code registers are used for specifying the pattern to match and the mask registers specify don’t care bits. In total eight registers are used for the acceptance filter as shown in the table below. Note that they are only read/writable in reset mode.

Table 223. Acceptance filter registers

Address	Description
16	Acceptance code 0 (ACR0)
17	Acceptance code 1 (ACR1)
18	Acceptance code 2 (ACR2)
19	Acceptance code 3 (ACR3)
20	Acceptance mask 0 (AMR0)
21	Acceptance mask 1 (AMR1)
22	Acceptance mask 2 (AMR2)
23	Acceptance mask 3 (AMR3)

Single filter mode, standard frame

When receiving a standard frame in single filter mode the registers ACR0-3 are compared against the incoming message in the following way:

ACR0.7-0 & ACR1.7-5 are compared to ID.28-18
ACR1.4 is compared to the RTR bit.
ACR1.3-0 are unused.
ACR2 & ACR3 are compared to data byte 1 & 2.

The corresponding bits in the AMR registers selects if the results of the comparison doesn't matter. A set bit in the mask register means don't care.

Single filter mode, extended frame

When receiving an extended frame in single filter mode the registers ACR0-3 are compared against the incoming message in the following way:

ACR0.7-0 & ACR1.7-0 are compared to ID.28-13
ACR2.7-0 & ACR3.7-3 are compared to ID.12-0
ACR3.2 are compared to the RTR bit
ACR3.1-0 are unused.

The corresponding bits in the AMR registers selects if the results of the comparison doesn't matter. A set bit in the mask register means don't care.

Dual filter mode, standard frame

When receiving a standard frame in dual filter mode the registers ACR0-3 are compared against the incoming message in the following way:

Filter 1

ACR0.7-0 & ACR1.7-5 are compared to ID.28-18
ACR1.4 is compared to the RTR bit.
ACR1.3-0 are compared against upper nibble of data byte 1
ACR3.3-0 are compared against lower nibble of data byte 1

Filter 2

ACR2.7-0 & ACR3.7-5 are compared to ID.28-18
ACR3.4 is compared to the RTR bit.

The corresponding bits in the AMR registers selects if the results of the comparison doesn't matter. A set bit in the mask register means don't care.

Dual filter mode, extended frame

When receiving a standard frame in dual filter mode the registers ACR0-3 are compared against the incoming message in the following way:

Filter 1

ACR0.7-0 & ACR1.7-0 are compared to ID.28-13

Filter 2

ACR2.7-0 & ACR3.7-0 are compared to ID.28-13

The corresponding bits in the AMR registers selects if the results of the comparison doesn't matter. A set bit in the mask register means don't care.

27.5.15 RX message counter

The RX message counter register at address 29 holds the number of messages currently stored in the receive fifo. The top three bits are always 0.

27.6 Common registers

There are three common registers with the same addresses and the same functionality in both Basi-CAN and PeliCAN mode. These are the clock divider register and bus timing register 0 and 1.

27.6.1 Clock divider register

The only real function of this register in the GRLIB version of the Opencores CAN is to choose between PeliCAN and BasiCAN. The clkout output of the Opencore CAN core is not connected and it is its frequency that can be controlled with this register.

*Table 224.*Bit interpretation of clock divider register (CDR) (address 31)

Bit	Name	Description	Reset value
CDR.7	CAN mode	1 - PeliCAN, 0 - BasiCAN	0
CDR.6	-	unused (cbp bit of SJA1000)	0
CDR.5	-	unused (rxinten bit of SJA1000)	0
CDR.4	-	reserved	0
CDR.3	Clock off	Disable the clkout output	0
CDR.2-0	Clock divisor	Frequency selector	0

27.6.2 Bus timing 0

*Table 225.*Bit interpretation of bus timing 0 register (BTR0) (address 6)

Bit	Name	Description
BTR0.7-6	SJW	Synchronization jump width
BTR0.5-0	BRP	Baud rate prescaler

The CAN core system clock is calculated as:

$$t_{scl} = 2*t_{clk}*(BRP+1)$$

where t_{clk} is the system clock.

The sync jump width defines how many clock cycles (t_{scl}) a bit period may be adjusted with by one re-synchronization. This register does not have a reset value.

27.6.3 Bus timing 1

*Table 226.*Bit interpretation of bus timing 1 register (BTR1) (address 7)

Bit	Name	Description
BTR1.7	SAM	1 - The bus is sampled three times, 0 - single sample point
BTR1.6-4	TSEG2	Time segment 2
BTR1.3-0	TSEG1	Time segment 1

The CAN bus bit period is determined by the CAN system clock and time segment 1 and 2 as shown in the equations below:

$$t_{tseg1} = t_{scl} * (TSEG1+1)$$
$$t_{tseg2} = t_{scl} * (TSEG2+1)$$
$$t_{bit} = t_{tseg1} + t_{tseg2} + t_{scl}$$

The additional t_{scl} term comes from the initial sync segment. Sampling is done between TSEG1 and TSEG2 in the bit period. This register does not have a reset value.

27.7 Design considerations

This section lists known differences between this CAN controller and SJA1000 on which is it based:

- All bits related to sleep mode are unavailable
- Output control and test registers do not exist (reads 0x00)
- Clock divisor register bit 6 (CBP) and 5 (RXINTEN) are not implemented
- Overrun irq and status not set until fifo is read out

BasicCAN specific differences:

- The receive irq bit is not reset on read, works like in PeliCAN mode
- Bit CR.6 always reads 0 and is not a flip flop with no effect as in SJA1000

PeliCAN specific differences:

- Writing 256 to tx error counter gives immediate bus-off when still in reset mode
- Read Buffer Start Address register does not exist
- Addresses above 31 are not implemented (i.e. the internal RAM/FIFO access)
- The core transmits active error frames in Listen only mode

27.8 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x019. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

27.9 Implementation

27.9.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). See the description of the *syncrst* VHDL generic for further information.

27.9.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

27.10 Configuration options

Table 227 shows the configuration options of the core (VHDL generics).

*Table 227.*Configuration options

Generic	Function	Allowed range	Default
slvndx	AHB slave bus index	0 - NAHBSLV-1	0
ioaddr	The AHB I/O area base address. Compared with bit 19-8 of the 32-bit AHB address.	0 - 16#FFF#	16#FFF#
iomask	The I/O area address mask. Sets the size of the I/O area and the start address together with ioaddr.	0 - 16#FFF#	16#FF0#
irq	Interrupt number	0 - NAHBIRQ-1	0
memtech	Technology to implement on-chip RAM	0	0 - NTECH
syncrst	Reset implementation 0: Use asynchronous reset 1: Use synchronous reset, leave internal buffers without reset 2: Use synchronous reset, initialize internal buffers to zero at reset.	0 - 2	0
ft	Enable fault-tolerance	0 - 1	0

27.11 Signal descriptions

Table 228 shows the interface signals of the core (VHDL ports).

*Table 228.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK		Input	AHB clock	
RESETN		Input	Reset	Low
AHBSI	*	Input	AMBA AHB slave inputs	-
AHBSO	*	Input	AMBA AHB slave outputs	
CAN_RXI		Input	CAN receiver input	High
CAN_TXO		Output	CAN transmitter output	High
MTESTI**	FIFO	Input	Memory BIST input signal to fifo RAM	-
	INFO	Input	Memory BIST input signal to info RAM	-
MTESTO**	FIFO	Output	Memory BIST output signal from fifo RAM	-
	INFO	Output	Memory BIST output signal from info RAM	-
MTESTCLK**	N/A	Input	Memory BIST clock	-

*1) see AMBA specification

** not available in FPGA releases

27.12 Signal definitions and reset values

The signals and their reset values are described in table 229.

*Table 229.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
cantx[]	Output	CAN transmit data	Low	Logical 1
canen[]	Output	CAN transmit enabel	-	Logical 0
canrx[]	Input	CAN receive data	Low	-

27.13 Timing

The timing waveforms and timing parameters are shown in figure 38 and are defined in table 230.

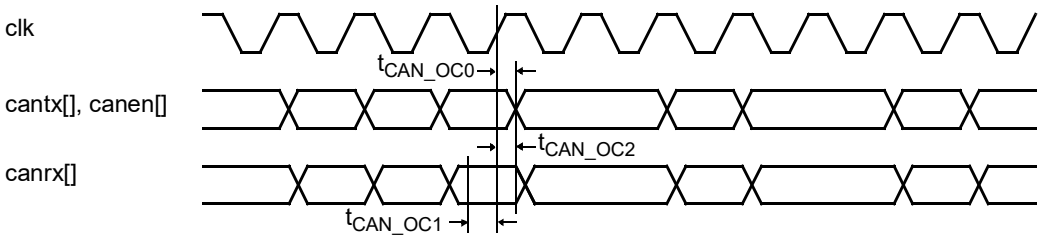


Figure 38. Timing waveforms

*Table 230.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{CAN_OC0}	clock to data output delay	rising *clk* edge	TBD	TBD	ns
t_{CAN_OC1}	data input to clock setup	rising *clk* edge	-	-	ns
t_{CAN_OC2}	data input from clock hold	rising *clk* edge	-	-	ns

Note: The *canrx[]* input is re-synchronized internally. The signal does not have to meet any setup or hold requirements.

27.14 Library dependencies

Table 231 shows libraries that should be used when instantiating the core.

*Table 231.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions
GAISLER	CAN	Component	Component declaration

27.15 Component declaration

```
library grlib;
use grlib.amba.all;
use gaisler.can.all;

component can_oc
   generic (
    slvndx    : integer := 0;
```

```
      ioaddr    : integer := 16#000#;
      iomask    : integer := 16#FF0#;
      irq       : integer := 0;
      memtech   : integer := 0);
    port (
      resetn  : in  std_logic;
      clk     : in  std_logic;
      ahbsi   : in  ahb_slv_in_type;
      ahbso   : out ahb_slv_out_type;
      can_rxi : in  std_logic;
      can_txo : out std_logic
);
    end component;
```

28 CLKGEN - Clock generation

28.1 Overview

The CLKGEN clock generator implements internal clock generation and buffering.

28.2 Technology specific clock generators

28.2.1 Overview

The core is a wrapper that instantiates technology specific primitives depending on the value of the *tech* VHDL generic. Each supported technology has its own subsection below. Table 232 lists the sub-section applicable for each technology setting. The table is arranged after the technology's numerical value in GRLIB. The subsections are ordered in alphabetical order after technology vendor.

*Table 232.*Overview of technology specific clock generator sections

Technology	Numerical value	Comment	Section
inferred	0	Default when no technology specific generator is available.	28.2.2
virtex	1		28.2.12
virtex2	2		28.2.13
memvirage	3	No technology specific clock generator available.	28.2.2
axcel	4		28.2.3
proasic	5		28.2.3
atc18s	6	No technology specific clock generator available.	28.2.2
altera	7		28.2.7
umc	8	No technology specific clock generator available.	28.2.2
rhumc	9		28.2.10
apa3	10		28.2.5
spartan3	11		28.2.11
ihp25	12	No technology specific clock generator available.	28.2.2
rhlib18t	13		28.2.9
virtex4	14		28.2.13
lattice	15	No technology specific clock generator available.	28.2.2
ut25	16	No technology specific clock generator available.	28.2.2
spartan3e	17		28.2.11
peregrine	18	No technology specific clock generator available.	28.2.2
memartisan	19	No technology specific clock generator available.	28.2.2
virtex5	20		28.2.14
custom1	21	No technology specific clock generator available.	28.2.2
ihp25rh	22	No technology specific clock generator available.	28.2.2
stratix1	23		28.2.7
stratix2	24		28.2.7
eclipse	25	No technology specific clock generator available.	28.2.2
stratix3	26		28.2.8
cyclone3	27		28.2.6
memvirage90	28	No technology specific clock generator available.	28.2.2
tsmc90	29	No technology specific clock generator available.	28.2.2
easic90	30		28.2.15
atc18rha	31	No technology specific clock generator available.	28.2.2

*Table 232.*Overview of technology specific clock generator sections

Technology	Numerical value	Comment	Section
smic013	32	No technology specific clock generator available.	28.2.2
tm65gpl	33	No technology specific clock generator available.	28.2.2
axdsp	34		28.2.3
spartan6	35		28.2.11
virtex6	36		28.2.14
actfus	37		28.2.17
stratix4	38		28.2.18
st65lp	39	No technology specific clock generator available.	28.2.2
st65gp	40	No technology specific clock generator available.	28.2.2
easic45	41		28.2.16
nx	62		20,2,19

28.2.2 Generic technology

This implementation is used when the clock generator does not support instantiation of technology specific primitives or when the inferred technology has been selected.

This implementation connects the input clock, CLKIN or PCICLKIN depending on the *pcien* and *pcisysclk* VHDL generic, to the SDCLK, CLK1XU, and CLK outputs. The CLKN output is driven by the inverted input clock. The PCICLK output is directly driven by PCICLKIN. Both clock lock signals are always driven to '1' and the CLK2X output is always driven to '0'.

In simulation, CLK, CLKN and CLK1XU transitions are skewed 1 ns relative to the SDRAM clock output.

28.2.3 ProASIC

Generics used in this technology:	pcisysclk
Instantiated technology primitives:	None
Signals not driven in this technology:	clk4x, clk1xu, clk2xu, clkb, clkc

This technology selection does not instantiate any technology specific primitives. The core's clock output, CLK, is driven by the CLKIN or PCICLKIN input depending on the value of VHDL generics *pcien* and *pcisysclk*.

The PCICLK is always directly connected to PCICLKIN. Outputs SDCLK, CLKN and CLK2X, are driven to ground. Both clock lock signals, CGO.CLKLOCK and CGO.PCILOCK, are always driven high.

28.2.4 Actel Axcelerator

Generics used in this technology:	pcisysclk, clk_mul, clk_div, pcien, freq
Instantiated technology primitives:	PLL
Signals not driven in this technology:	clk4x, clk1xu, clk2xu, clkb, clkc

This technology selection has two modes. The first one is used if VHDL generics *clk_mul* and *clk_div* are equal and does not instantiate any technology specific primitives. The core's clock output, CLK, is driven by the CLKIN or PCICLKIN input depending on the value of VHDL generics *pcien* and *pcisysclk*.

The second mode is used if VHDL generics *clk_mul* and *clk_div* are different and instantiates a PLL. The core's clock output CLK is either driven by the pciclkin input or the main output from the PLL

depending on the values of VHDL generics *pcien* and *pcisysclk*. When the PLL drives the CLK output the resulting frequency is the frequency of CLKIN multiplied by the VHDL generic *clk_mul* and divided by the VHDL generic *clk_div*. Clock buffers are not instantiated within the clock generator and has to be done externally.

For both modes the following applies:

The PCICLK is always directly connected to PCICLKIN. Outputs SDCLK, CLKN and CLK2X, are driven to ground. Both clock lock signals, CGO.CLKLOCK and CGO.PCILOCK, are always driven high.

28.2.5 Actel ProASIC3

Generics used in this technology:	clk_mul, clk_div, clk_odiv, pcisysclk, pcien, freq, clkb_odiv, clkc_odiv
Instantiated technology primitives:	PLLINT, PLL
Signals not driven in this technology:	clkn, sdclk, clk2x, clk4x, clk1xu, clk2xu

This technology instantiates a PLL and a PLLINT to generate the main clock. The instantiation of a PLLINT macro allows the PLL reference clock to be driven from an I/O that is routed through the regular FPGA routing fabric. Figure 39 shows the instantiated primitives, the PLL EXTFB input is not shown and the EXTFB port on the instantiated component is always tied to ground. The figure shows which of the core's output ports that are driven by the PLL. The PCICLOCK will directly connected to PCICLKIN if VHDL generic *pcien* is non-zero, while CGO.PCILOCK is always driven high. The VHDL generics *pcien* and *pcisysclk* are used to select the reference clock. The values driven on the PLL inputs are listed in tables 233 and 234.

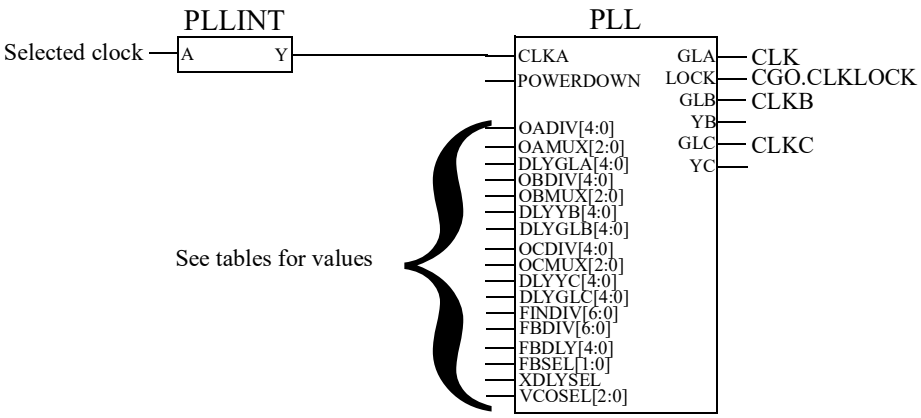


Figure 39. Actel ProASIC3 clock generation

*Table 233.*Constant input signals on Actel ProASIC3 PLL

Signal name	Value	Comment
OADIV[4:0]	VHDL generic *clk_odiv* - 1	Output divider
OAMUX[2:0]	0b100	Post-PLL MUXA
DLYGLA[4:0]	0	Delay on Global A
OBDIV[4:0]	VHDL generic *clkb_odiv* - 1 when *clkb_odiv* > 0, otherwise 0	Output divider
OBMUX[2:0]	0 when VHDL generic *clkb_odiv* = 0, otherwise 0b100	Post-PLL MUXB
DLYYB[4:0]	0	Delay on YB
DLYGLB[4:0]	0	Delay on Global B
OCDIV[4:0]	VHDL generic *clkc_odiv* - 1 when *clkc_odiv* > 0, otherwise 0	Output divider
OCMUX[2:0]	0 when VHDL generic *clkc_odiv* = 0, otherwise 0b100	Post-PLL MUXC
DLYYC[4:0]	0	Delay on YC
DLYGLC[4:0]	0	Delay on Global C
FINDIV[6:0]	VHDL generic *clk_div* - 1	Input divider
FBDIV[6:0]	VHDL generic *clk_mul* - 1	Feedback divider
FBDLY[4:0]	0	Feedback delay
FBSEL[1:0]	0b01	2-bit PLL feedback MUX
XDLYSEL	0	1-bit PLL feedback MUX
VCOSEL[2:0]	*See table 234 below*	VCO gear control. Selects one of four frequency ranges.

The PLL primitive has one parameter, VCOFREQUENCY, which is calculated with:

$$VCOFREQUENCY = \frac{freq \cdot clkmul}{clkdiv} / 1000$$

The calculations are performed with integer precision. This value is also used to determine the value driven on PLL input VCOSEL[2:0]. Table 234 lists the signal value depending on the value of VCOFREQUENCY.

*Table 234.*VCOSEL[2:0] on Actel ProASIC3 PLL

Value of VCOFREQUENCY	Value driven on VCOSEL[2:0]
< 44	0b000
< 88	0b010
< 175	0b100
>= 175	0b110

28.2.6 Altera Cyclone III

Generics used in this technology:	clk_mul, clk_div, sdramen, pcien, pcisysclk, freq, clk2xen
Instantiated technology primitives:	ALTPLL
Signals not driven in this technology:	clk4x, clk1xu, clk2xu, clkb, clkc

This technology instantiates an ALTPLL primitive to generate the required clocks, see figure 40. The ALTPLL attributes are listed in table 235. As can be seen in this table the attributes OPERATION_-MODE and COMPENSATE_CLOCK depend on the VHDL generic *sdramen*.

Table 235. Altera Cyclone III ALTPLL attributes

Attribute name*	**Value with *sdramen* = 1**	**Value with *sdramen* = 0**
INTENDED_DEVICE_FAMILY	“Cyclone III”	“Cyclone III”
OPERATION_MODE	“ZERO_DELAY_BUFFER”	“NORMAL”
COMPENSATE_CLOCK	“CLK1”	“clock0”
INCLK0_INPUT_FREQUENCY	1000000000 / (VHDL generic *freq*)	1000000000 / (VHDL generic *freq*)
WIDTH_CLOCK	5	5
CLK0_MULTIPLY_BY	VHDL generic *clk_mul*	VHDL generic *clk_mul*
CLK0_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*
CLK1_MULTIPLY_BY	VHDL generic *clk_mul*	VHDL generic *clk_mul*
CLK1_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*
CLK2_MULTIPLY_BY	VHDL generic *clk_mul * 2*	VHDL generic *clk_mul * 2*
CLK2_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*

*Any attributes not listed are assumed to have their default value

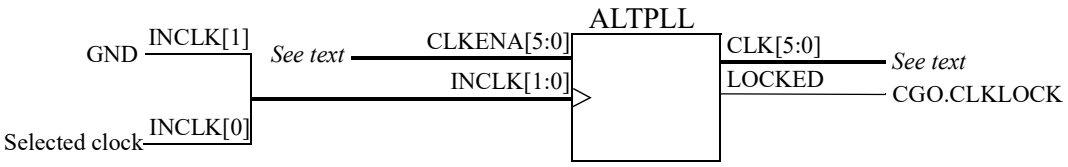


Figure 40. Altera Cyclone III ALTPLL

The value driven on the ALTPLL clock enable signal is dependent on the VHDL generics *clk2xen* and *sdramen*, table 236 lists the effect of these generics.

Table 236. Effect of VHDL generics *clk2xen* and *sdramen* on ALTPLL clock enable input

Value of sdramen	**Value of clk2xen**	**Value of CLKENA[5:0]**
0	0	0b000001
0	1	0b000101
1	0	0b000011
1	1	0b000111

Table 237 lists the connections of the core's input and outputs to the ALTPLL ports.

*Table 237.*Connections between core ports and ALTPLL ports

Core signal	Core direction	ALTPLL signal
CLKIN/PCICLKIN*	Input	INCLK[0]
CLK	Output	CLK[0]
CLKN	Output	$\overline{\text{CLK[0]}}$ (CLK[0] through an inverter)
CLK2X	Output	CLK[2]
SDCLK	Output	CLK[1]
CGO.CLKLOCK	Output	LOCKED

* Depending on VHDL generics PCIEN and PCISYSCLK, as described below.

The clocks can be generated using either the CLKIN input or the PCICLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. If *pcien* is 0 or *pcisysclk* is 0 the input clock to the ALTPLL will be CLKIN. If *pcien* is non-zero and *pcisysclk* is 1 the input to the ALTPLL will be PCICLKIN.

The PCICLK output will connected to the PCICLKIN input if VHDL generic *pcien* is non-zero. Otherwise the PCICLK output will be driven to ground. The CGO.PCILOCK signal is always driven high.

28.2.7 Altera Stratix 1/2

Generics used in this technology:	clk_mul, clk_div, sdramen, pcien, pcisysclk, freq, clk2xen
Instantiated technology primitives:	ALTPLL
Signals not driven in this technology:	clk4x, clk1xu, clk2xu, clkb, clkc

This technology instantiates an ALTPLL primitive to generate the required clocks, see figure 41. The ALTPLL attributes are listed in table 238. As can be seen in this table the OPERATION_MODE attribute depends on the VHDL generic *sdramen*.

*Table 238.*Altera Stratix 1/2 ALTPLL attributes

Attribute name*	Value with *sdramen* = 1	Value with *sdramen* = 0
OPERATION_MODE	"ZERO_DELAY_BUFFER"	"NORMAL"
INCLK0_INPUT_FREQUENCY	1000000000 / (VHDL generic *freq*)	1000000000 / (VHDL generic *freq*)
WIDTH_CLOCK	6	6
CLK0_MULTIPLY_BY	VHDL generic *clk_mul*	VHDL generic *clk_mul*
CLK0_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*
CLK1_MULTIPLY_BY	VHDL generic *clk_mul* * 2	VHDL generic *clk_mul* * 2
CLK1_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*
EXTCLK0_MULTIPLY_BY	VHDL generic *clk_mul*	VHDL generic *clk_mul*
EXTCLK0_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*

*Any attributes not listed are assumed to have their default value

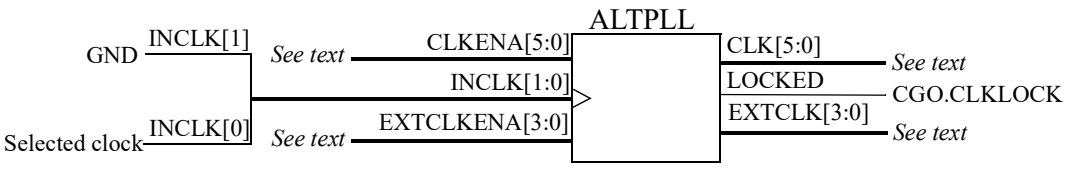


Figure 41. Altera Stratix 1/2 ALTPLL

The values driven on the ALTPLL clock enable signals are dependent on the VHDL generic *clk2xen*, table 239 lists the effect of *clk2xen*.

*Table 239.*Effect of VHDL generic *clk2xen* on ALTPLL clock enable inputs

Signal	**Value with *clk2xen* = 0**	**Value with *clk2xen* /= 0**
CLKENA[5:0]	0b000001	0b000011
EXTCLKENA[3:0]	0b0001	0b0011

Table 240 lists the connections of the core's input and outputs to the ALTPLL ports.

*Table 240.*Connections between core ports and ALTPLL ports

Core signal	**Core direction**	**ALTPLL signal**
CLKIN/PCICLKIN*	Input	INCLK[0]
CLK	Output	CLK[0]
CLKN	Output	$\overline{\text{CLK[0]}}$ (CLK[0] through an inverter)
CLK2X	Output	CLK[1]
SDCLK	Output	EXTCLK[0]
CGO.CLKLOCK	Output	LOCKED

* Depending on VHDL generics PCIEN and PCISYSCLK, as described below.

The clocks can be generated using either the CLKIN input or the PCICLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. If *pcien* is 0 or *pcisysclk* is 0 the input clock to the ALTPLL will be CLKIN. If *pcien* is non-zero and *pcisysclk* is 1 the input to the ALTPLL will be PCICLKIN.

The PCICLK output will connected to the PCICLKIN input if VHDL generic *pcien* is non-zero. Otherwise the PCICLK output will be driven to ground. The CGO.PCILOCK signal is always driven high.

28.2.8 Altera Stratix 3

This technology is not fully supported at this time.

28.2.9 RHLIB18t

Generics used in this technology:	clk_mul, clk_div
Instantiated technology primitives:	lfdll_top
Signals not driven in this technology:	-

Please contact Frontgrade Gaisler for information concerning the use of this clock generator.

28.2.10 RHUMC

Generics used in this technology:	None
Instantiated technology primitives:	pll_ip
Signals not driven in this technology:	-

Please contact Frontgrade Gaisler for information concerning the use of this clock generator.

28.2.11 Xilinx Spartan 3/3e/6

Generics used in this technology:	clk_mul, clk_div, sdramen, noclkfb, pcien, pcidll, pcisysclk, freq, clk2xen, clksel
Instantiated technology primitives:	BUFG, BUFMUX, DCM, BUFGDLL
Signals not driven in this technology:	clk4x, clkb, clkc

The main clock is generated with a DCM which is instantiated with the attributes listed in table 241. The input clock source connected to the CLKIN input is either the core's CLKIN input or the PCI-CLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. The main DCM's connections is shown in figure 42.

*Table 241.*Spartan 3/e DCM attributes

Attribute name*	**Value**
CLKDV_DIVIDE	2.0
CLKFX_DIVIDE	Determined by core's VHDL generic *clk_div*
CLKFX_MULTIPLY	Determined by core's VHDL generic *clk_mul*
CLKIN_DIVIDE_BY_2	false
CLKIN_PERIOD	10.0
CLKOUT_PHASE_SHIFT	"NONE"
CLK_FEEDBACK	"2X"
DESKEW_ADJUST	"SYSTEM_SYNCHRONOUS"
DFS_FREQUENCY_MODE	"LOW"
DLL_FREQUENCY_MODE	"LOW"
DSS_MODE	"NONE"
DUTY_CYCLE_CORRECTION	true
FACTORY_JF	X"C080"
PHASE_SHIFT	0
STARTUP_WAIT	false

*Any attributes not listed are assumed to have their default value

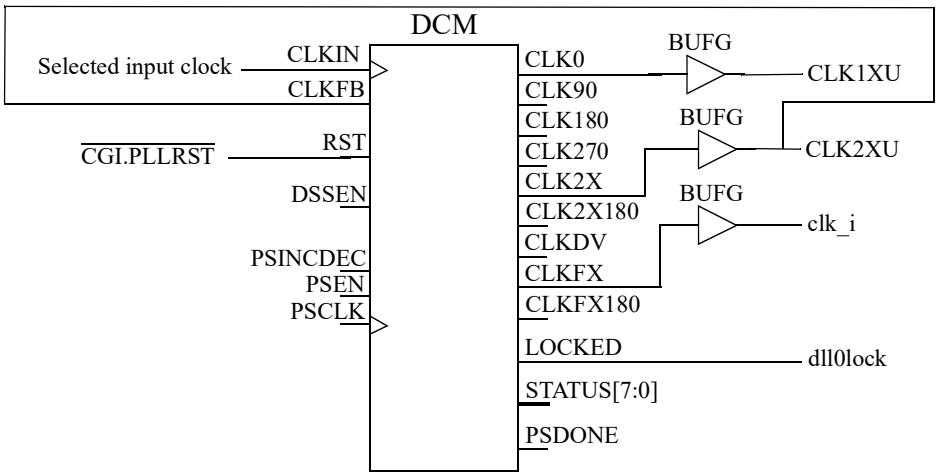


Figure 42. Spartan 3/e generation of main clock

If the VHDL generic *clk2xen* is non-zero the DCM shown in figure 43 is instantiated. The attributes of this DCM are the same as in table 241, except that the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes are both set to 2 and the CLK_FEEDBACK attribute is set to "1X". The dll0lock signal is connected to the LOCKED output of the main clock DCM. When this signal is low all the bits in the

shift register connected to the CLK2X DCM's RST input are set to '1'. When the dll0lock signal is asserted it will take four main clock cycles until the RST input is deasserted. Depending on the value of the *clksel* VHDL generic the core's CLK2X output is either driven by a BUFG or a BUFGMUX. Figure 44 shows the two alternatives and how the CGI.CLKSEL(0) input is used to selected between the CLK0 and CLK2X output of the CLK2X DCM.

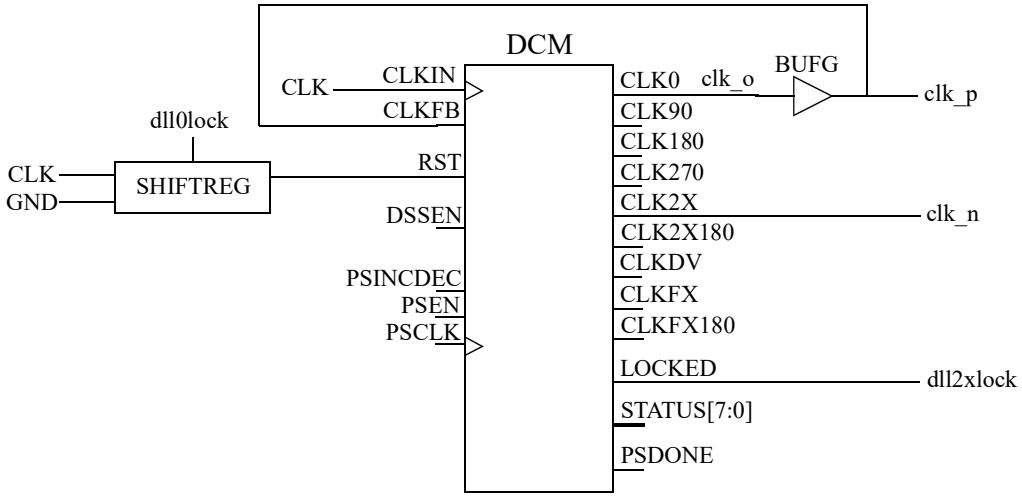


Figure 43. Spartan 3/e generation of CLK2X clock when VHDL generic *clk2xen* is non-zero

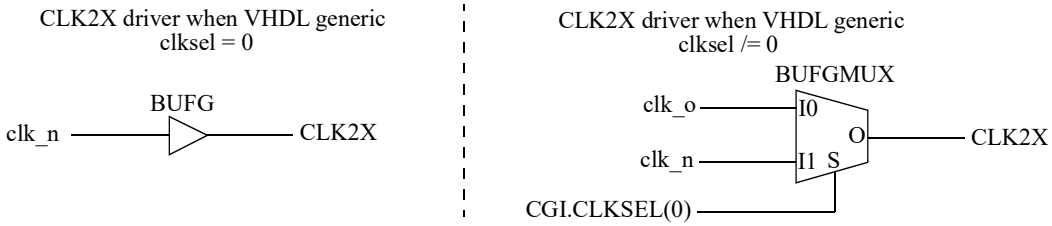


Figure 44. Spartan 3/e selection of CLK2X clock when VHDL generic *clk2xen* is non-zero

The value of the *clk2xen* VHDL generic also decides which output that drives the core's CLK output. If the VHDL generic is non-zero the CLK output is driven by the clk_p signal originating from the CLK2X DCM. Otherwise the CLK output is connected to the clk_i signal originating from the main clock DCM. The core's CLKN output is driven by the selected signal through an inverter. Figure 45 illustrates the connections.

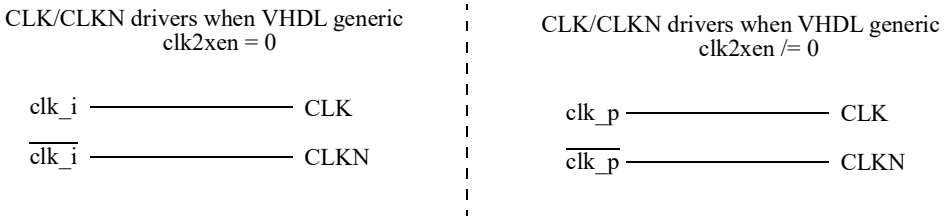


Figure 45. Spartan 3/e clock generator outputs CLK and CLKN

If the VHDL generic *clk2xen* is zero the dll0lock signal from the main clock DCM is either connected to the SDRAM DCM, described below, or if the SDRAM DCM is non-existent, to the core's CGO.CLKLOCK output. This setting also leads to the core's CLK2X output being driven by the main clock DCM's CLK2X output via a BUFG, please see figure 46.

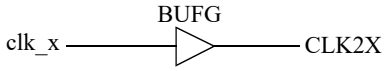


Figure 46. Spartan 3/e generation of CLK2X clock when VHDL generic *clk2xen* is zero

If the SDRAM clock is enabled, via the *sdramen* VHDL generic, and the clock generator is configured to use clock feedback the DCM shown in figure 47 is instantiated. This DCM has the same attributes as the CLK2X DCM. The input to the SDRAM DCM input clock is determined via the *clk2xen* VHDL generic. If the VHDL generic is set to 0 the input is the main CLK, if the generic is set to 1 the input is the clk_p out of the CLK2X DCM shown in figure 44. If the *clk2xen* VHDL generic is set to 2 the clock input to the SDRAM DCM depends on the *clksel* VHDL generic. The input in this last case is the CLK2X output shown in figure 46.

If the CLK2X DCM has been instantiated the SDRAM DCM RST input depends on the LOCKED output of the CLK2X DCM. If the CLK2X DCM has not been instantiated the SDRAM DCM RST input depends on the LOCKED output from the main clock DCM. The applicable LOCKED signal is utilized to keep the SDRAM DCM in reset until its input clock has been stabilized. This is done with a shift register with the same method used for the CLK2X DCM RST.

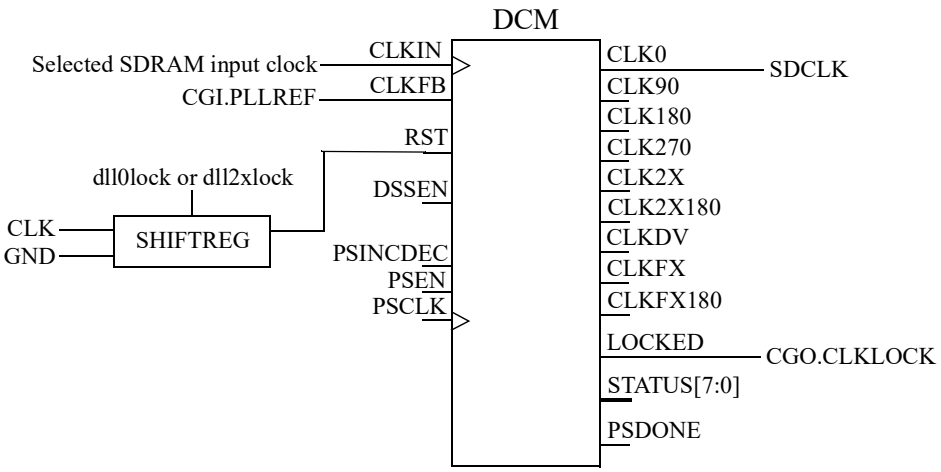


Figure 47. Spartan 3/e generation of SDRAM clock

If the SDRAM clock is disabled (*sdramen* VHDL generic set to 0) or the core has been configured not to use clock feedback (*noclockfb* VHDL generic set to 1) the driver of the core's SDCLK output is determined by the value of the *clk2xen* VHDL generic. If the *clk2xen* VHDL generic is set to 2, the SDRAM clock output is the same as the CLK2X output shown in figure 44, in other words it also depends on the *clksel* VHDL generic. If the *clk2xen* VHDL generic has any other value the SDCLK output is the same as the core's CLK output.

When the *sdramen* VHDL generic is set to 0 the core's CGO.CLKLOCK output is connected to the CLK2X DCM's LOCKED output, if the DCM exists, otherwise the CGO.CLKLOCK output is connected to the main clock DCM's LOCKED output.

If PCI clock generation is enabled via the *pcien* VHDL generic the core instantiates either a BUFG or a BUFGDLL as depicted in figure 48 below. Note that the PCI clock must be enabled if the main clock is to be driven by the PCICLKIN input. If the PCI clock is disabled the PCICLK output is driven to zero. The CGO.PCILOCK output is always driven high in all configurations.

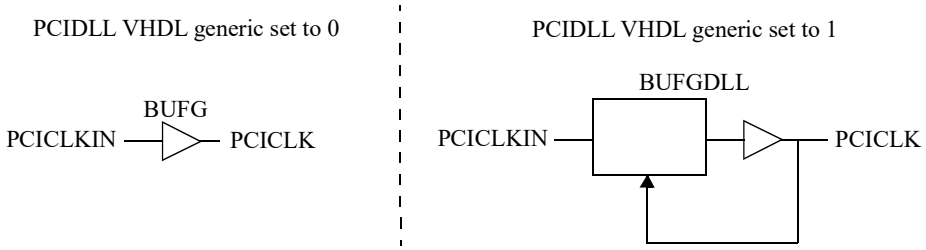


Figure 48. Spartan 3/e PCI clock generation

28.2.12 Xilinx Virtex

Generics used in this technology:	clk_mul, clk_div, sdramen, noclkfb, pcien, pcidll, pcisysclk
Instantiated technology primitives:	BUFG, BUFGDLL, CLKDLL
Signals not driven in this technology:	clk4x, clk1xu, clk2xu, clkb, clkc

The main clock is generated with the help of a CLKDLL. Figure 49 below shows how the CLKDLL primitive is connected. The input clock source is either the core's CLKIN input or the PCICLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. The figure shows three potential drivers of the BUFG driving the output clock CLK, the driver is selected via the VHDL generics *clk_mul* and *clk_div*. If *clk_mul/clk_div* is equal to 2 the CLK2X output is selected, if *clk_div/clk_mul* equals 2 the CLKDV output is selected, otherwise the CLK0 output drives the BUFG. The inverted main clock output, CLKN, is the BUFG output connected via an inverter.

The figure shows a dashed line connecting the CLKDLL's LOCKED output to the core output CGO.CLKLOCK. The driver of the CGO.CLKLOCK output depends on the instantiation of a CLK-DLL for the SDRAM clock. See description of the SDRAM clock below.

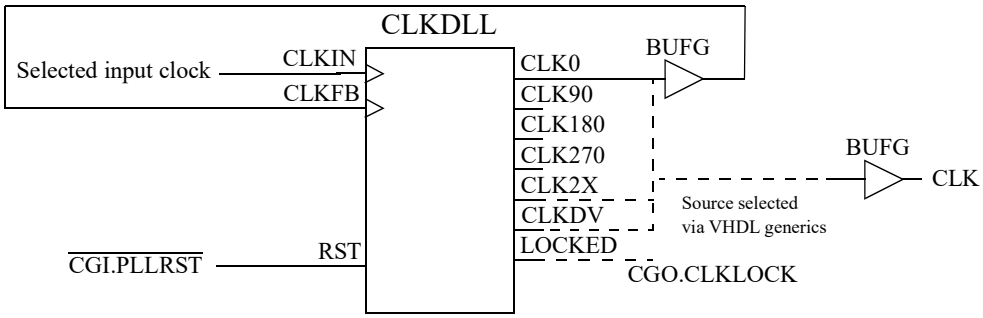


Figure 49. Virtex generation of main clock

If the SDRAM clock is enabled, via the *sdramen* VHDL generic, and the clock generator is configured to use clock feedback, VHDL generic *noclkfb* set to 0, a CLKDLL is instantiated as depicted in figure 50. Note how the CLKDLL's RST input is connected via a shift register clocked by the main clock. The shift register is loaded with all '1' when the LOCKED signal of the main clock CLKDLL is low. When the LOCKED signal from the main clock CLKDLL is asserted the SDRAM CLKDLL's RST input will be deasserted after four main clock cycles.

For all other configurations the SDRAM clock is driven by the main clock and the CGO.CLKLOCK signal is driven by the main clock CLKDLL's LOCKED output. The SDRAM CLKDLL must be present if the core's CLK2X output shall be driven.

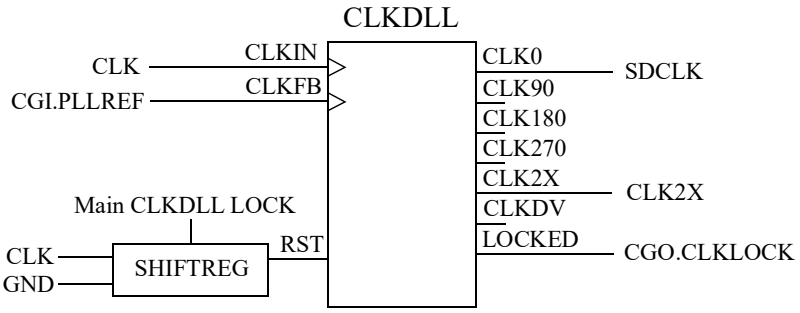


Figure 50. Virtex generation of SDRAM clock with feedback clock enabled

If PCI clock generation is enabled via the *pcien* VHDL generic the core instantiates either a BUFG or a BUFGDLL as depicted in figure 51 below. Note that the PCI clock must be enabled if the main clock is to be driven by the PCICLKIN input. If the PCI clock is disabled the PCICLK output is driven to zero. The CGO.PCILOCK output is always driven high in all configurations.

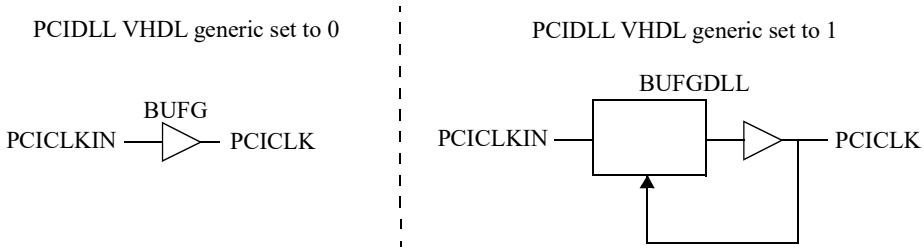


Figure 51. Virtex PCI clock generation

28.2.13 Xilinx Virtex 2/4

Generics used in this technology:	clk_mul, clk_div, sdramen, noclkfb, pcien, pcidll, pcisysclk, freq, clk2xen, clksel
Instantiated technology primitives:	BUFG, BUFMUX, DCM, BUFGDLL
Signals not driven in this technology:	clk4x, clkb, clkc

The main clock is generated with a DCM which is instantiated with the attributes listed in table 242. The input clock source connected to the CLKIN input is either the core's CLKIN input or the PCI-CLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. The main DCM's connections is shown in figure 52.

*Table 242.*Virtex 2/4 DCM attributes

Attribute name*	Value
CLKDV_DIVIDE	2.0
CLKFX_DIVIDE	Determined by core's VHDL generic *clk_div*
CLKFX_MULTIPLY	Determined by core's VHDL generic *clk_mul*
CLKIN_DIVIDE_BY_2	false
CLKIN_PERIOD	10.0
CLKOUT_PHASE_SHIFT	"NONE"
CLK_FEEDBACK	"1X"
DESKEW_ADJUST	"SYSTEM_SYNCHRONOUS"
DFS_FREQUENCY_MODE	"LOW"
DLL_FREQUENCY_MODE	"LOW"
DSS_MODE	"NONE"
DUTY_CYCLE_CORRECTION	true
FACTORY_JF	X"C080"
PHASE_SHIFT	0
STARTUP_WAIT	false

*Any attributes not listed are assumed to have their default value

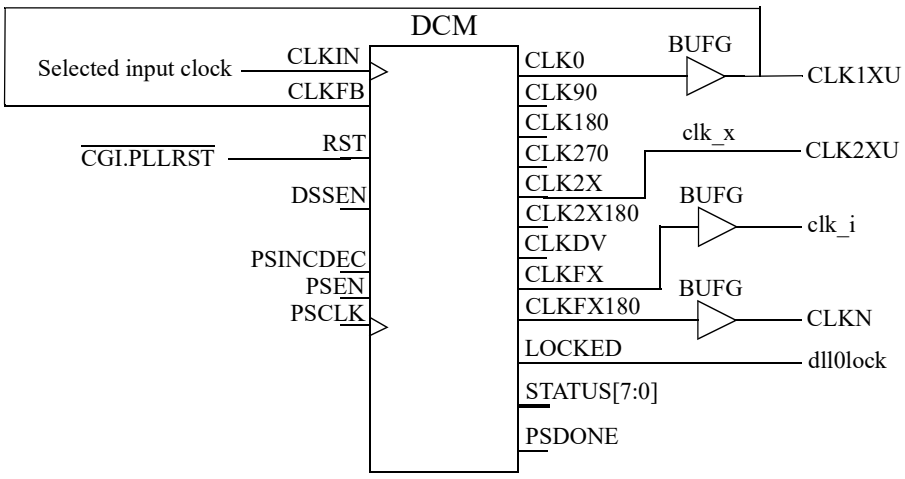


Figure 52. Virtex 2/4 generation of main clock

If the VHDL generic *clk2xen* is non-zero the DCM shown in figure 53 is instantiated. The attributes of this DCM are the same as in table 242, except that the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes are both set to 2. The dll0lock signal is connected to the LOCKED output of the main clock DCM. When this signal is low all the bits in the shift register connected to the CLK2X DCM's RST input are set to '1'. When the dll0lock signal is asserted it will take four main clock cycles until the RST input is deasserted. Depending on the value of the *clksel* VHDL generic the core's CLK2X output is either driven by a BUFG or a BUFGMUX. Figure 54 shows the two alternatives and how the CGI.CLKSEL(0) input is used to selected between the CLK0 and CLK2X output of the CLK2X DCM.

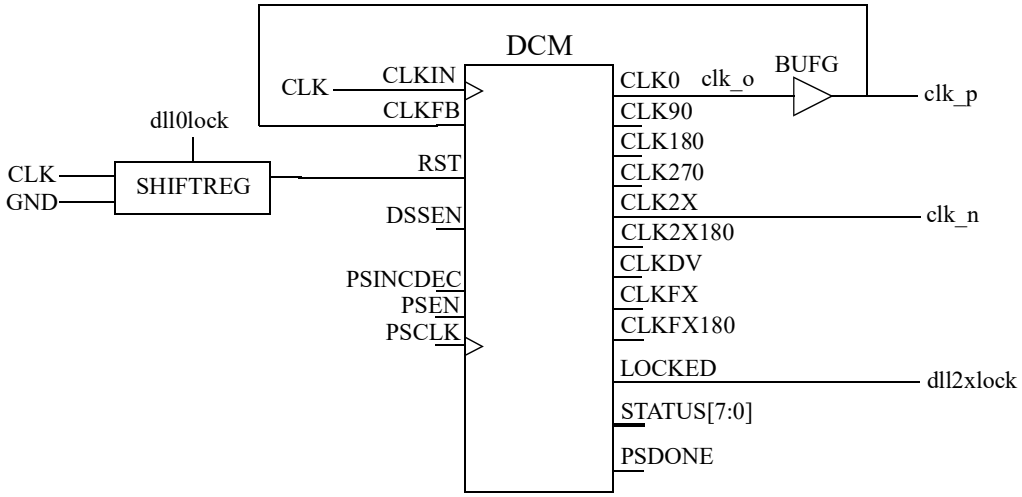


Figure 53. Virtex 2/4 generation of CLK2X clock when VHDL generic *clk2xen* is non-zero

The value of the *clk2xen* VHDL generic also decides which output that drives the core's CLK output. If the VHDL generic is non-zero the CLK output is driven by the clk_p signal originating from the CLK2X DCM. Otherwise the CLK output is connected to the clk_i signal originating from the main clock DCM. Note that the CLKN output always originates from the main clock DCM, as shown in figure 52.

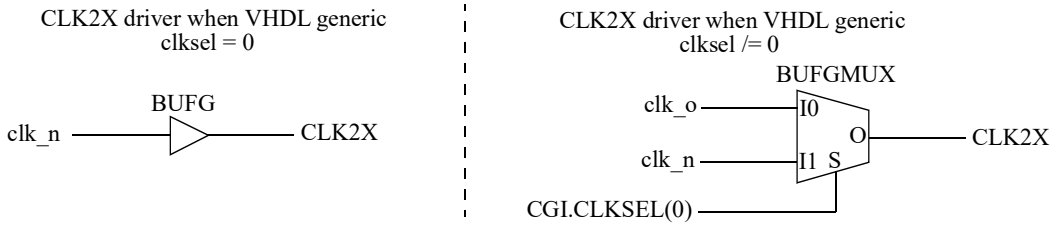


Figure 54. Virtex 2/4 selection of CLK2X clock when VHDL generic *clk2xen* is non-zero

If the VHDL generic *clk2xen* is zero the dll0lock signal from the main clock DCM is either connected to the SDRAM DCM, described below, or if the SDRAM DCM is non-existent, to the core's CGO.CLKLOCK output. This setting also leads to the core's CLK2X output being driven by the main clock DCM's CLK2X output via a BUFG, please see figure 55.

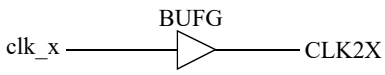


Figure 55. Virtex 2/4 generation of CLK2X clock when VHDL generic *clk2xen* is zero

If the SDRAM clock is enabled, via the *sdramen* VHDL generic, and the clock generator is configured to use clock feedback the DCM shown in figure 56. The input to the SDRAM DCM input clock is determined via the *clk2xen* VHDL generic. If the VHDL generic is set to 0 the input is the main CLK, if the generic is set to 1 the input is the clk_p out of the CLK2X DCM shown in figure 53. If the *clk2xen* VHDL generic is set to 2 the clock input to the SDRAM DCM depends on the *clksel* VHDL generic. The input in this last case is the CLK2X output shown in figure 54.

If the CLK2X DCM has been instantiated the SDRAM DCM RST input depends on the LOCKED output of the CLK2X DCM. If the CLK2X DCM has not been instantiated the SDRAM DCM RST input depends on the LOCKED output from the main clock DCM. The applicable LOCKED signal is

utilized to keep the SDRAM DCM in reset until its input clock has been stabilized. This is done with a shift register with the same method used for the CLK2X DCM RST.

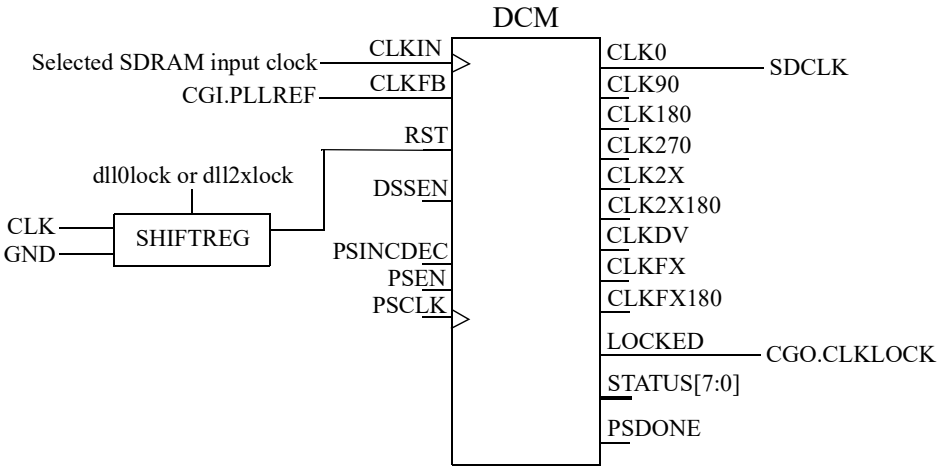


Figure 56. Virtex 2/4 generation of SDRAM clock

If the SDRAM clock is disabled (*sdramen* VHDL generic set to 0) or the core has been configured not to use clock feedback (*noclockfb* VHDL generic set to 1) the driver of the core's SDCLK output is determined by the value of the *clk2xen* VHDL generic. If the clk2xen VHDL generic is set to 2, the SDRAM clock output is the same as the CLK2X output shown in figure 54, in other words it also depends on the *clksel* VHDL generic. If the *clk2xen* VHDL generic has any other value the SDCLK output is the same as the core's CLK output.

When the *sdramen* VHDL generic is set to 0 the core's CGO.CLKLOCK output is connected to the CLK2X DCM's LOCKED output, if the DCM exists, otherwise the CGO.CLKLOCK output is connected to the main clock DCM's LOCKED output.

If PCI clock generation is enabled via the *pcien* VHDL generic the core instantiates either a BUFG or a BUFGDLL as depicted in figure 57 below. Note that the PCI clock must be enabled if the main clock is to be driven by the PCICLKIN input. If the PCI clock is disabled the PCICLK output is driven to zero. The CGO.PCILOCK output is always driven high in all configurations.

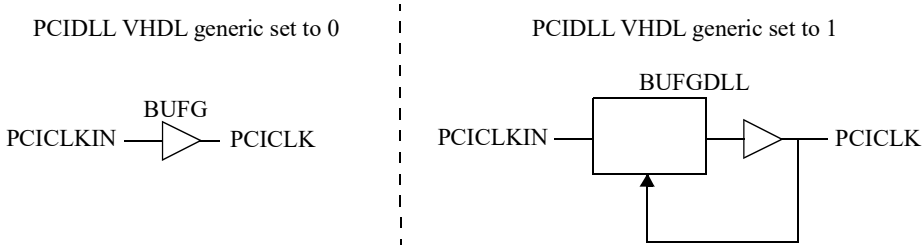


Figure 57. Virtex 2/4 PCI clock generation

28.2.14 Xilinx Virtex 5/6

Generics used in this technology:	clk_mul, clk_div, sdramen, noclkfb, pcien, pcidll, pcisysclk, freq, clk2xen, clksel
Instantiated technology primitives:	BUFG, BUFMUX, DCM, BUFGDLL
Signals not driven in this technology:	clk4x, clkb, clkc

The main clock is generated with a DCM which is instantiated with the attributes listed in table 243. The input clock source connected to the CLKIN input is either the core's CLKIN input or the PCI-

CLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. The main DCM's connections is shown in figure 58.

Table 243. Virtex 5 DCM attributes

Attribute name*	**Value**
CLKDV_DIVIDE	2.0
CLKFX_DIVIDE	Determined by core's VHDL generic *clk_div*
CLKFX_MULTIPLY	Determined by core's VHDL generic *clk_mul*
CLKIN_DIVIDE_BY_2	false
CLKIN_PERIOD	10.0
CLKOUT_PHASE_SHIFT	"NONE"
CLK_FEEDBACK	"1X"
DESKEW_ADJUST	"SYSTEM_SYNCHRONOUS"
DFS_FREQUENCY_MODE	"LOW"
DLL_FREQUENCY_MODE	"LOW"
DSS_MODE	"NONE"
DUTY_CYCLE_CORRECTION	true
FACTORY_JF	X"C080"
PHASE_SHIFT	0
STARTUP_WAIT	false

*Any attributes not listed are assumed to have their default value

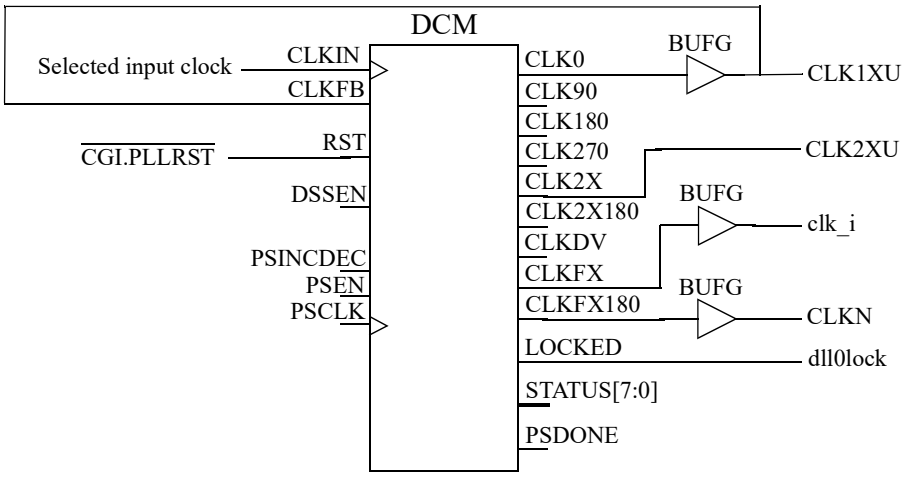


Figure 58. Virtex 5 generation of main clock

If the VHDL generic *clk2xen* is non-zero the DCM shown in figure 59 is instantiated. The attributes of this DCM are the same as in table 243, except that the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes are both set to 2. The dll0lock signal is connected to the LOCKED output of the main clock DCM. When this signal is low all the bits in the shift register connected to the CLK2X DCM's RST input are set to '1'. When the dll0lock signal is asserted it will take four main clock cycles until the RST input is deasserted. Depending on the value of the *clksel* VHDL generic the core's CLK2X output is either driven by a BUFG or a BUFGMUX. Figure 60 shows the two alternatives and how the CGI.CLKSEL(0) input is used to selected between the CLK0 and CLK2X output of the CLK2X DCM.

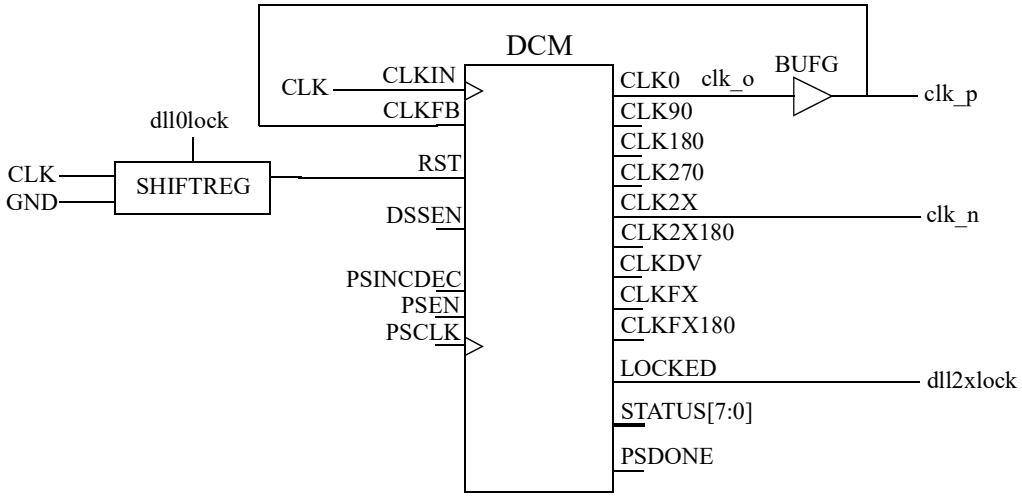


Figure 59. Virtex 5 generation of CLK2X clock when VHDL generic *clk2xen* is non-zero

The value of the *clk2xen* VHDL generic also decides which output that drives the core's CLK output. If the VHDL generic is non-zero the CLK output is driven by the clk_p signal originating from the CLK2X DCM. Otherwise the CLK output is connected to the clk_i signal originating from the main clock DCM. Note that the CLKN output always originates from the main clock DCM, as shown in figure 58.

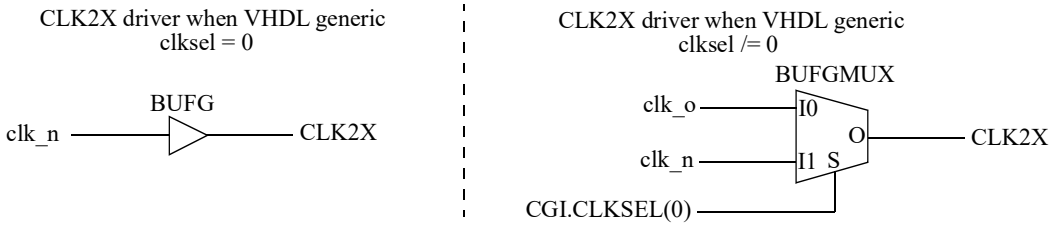


Figure 60. Virtex 5 selection of CLK2X clock when VHDL generic *clk2xen* is non-zero

If the VHDL generic *clk2xen* is zero the dll0lock signal from the main clock DCM is either connected to the SDRAM DCM, described below, or if the SDRAM DCM is non-existent, to the core's CGO.CLKLOCK output. This setting also leads to the core's CLK2X output being driven directly by the main clock DCM's CLK2X output.

If the SDRAM clock is enabled, via the *sdramen* VHDL generic, and the clock generator is configured to use clock feedback the DCM shown in figure 61. This DCM has the same attributes as the main clock DCM described in table 243, with the exceptions that CLKFX_MULTIPLY and CLKFX_DIVIDE are both set to 2 and DESKEW_ADJUST is set to "SOURCE_SYNCHRONOUS".

The input to the SDRAM DCM input clock is determined via the *clk2xen* VHDL generic. If the VHDL generic is set to 0 the input is the main CLK, if the generic is set to 1 the input is the clk_p out of the CLK2X DCM shown in figure 53. If the *clk2xen* VHDL generic is set to 2 the clock input to the SDRAM DCM depends on the *clksel* VHDL generic. The input in this last case is the CLK2X output shown in figure 60.

If the CLK2X DCM has been instantiated the SDRAM DCM RST input depends on the LOCKED output of the CLK2X DCM. If the CLK2X DCM has not been instantiated the SDRAM DCM RST input depends on the LOCKED output from the main clock DCM. The applicable LOCKED signal is utilized to keep the SDRAM DCM in reset until its input clock has been stabilized. This is done with a shift register with the same method used for the CLK2X DCM RST.

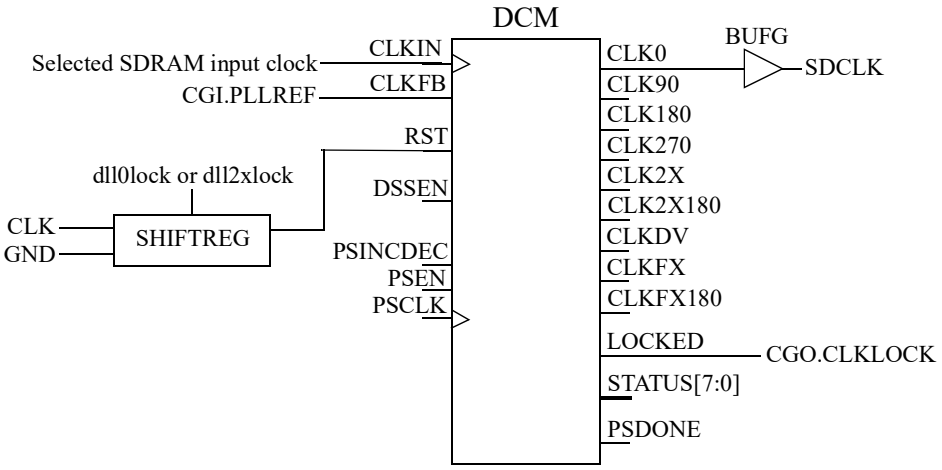


Figure 61. Virtex 5 generation of SDRAM clock

If the SDRAM clock is disabled (*sdramen* VHDL generic set to 0) or the core has been configured not to use clock feedback (*noclockfb* VHDL generic set to 1) the driver of the core's SDCLK output is determined by the value of the *clk2xen* VHDL generic. If the clk2xen VHDL generic is set to 2, the SDRAM clock output is the same as the CLK2X output shown in figure 60, in other words it also depends on the *clksel* VHDL generic. If the *clk2xen* VHDL generic has any other value the SDCLK output is the same as the core's CLK output.

When the *sdramen* VHDL generic is set to 0 the core's CGO.CLKLOCK output is connected to the CLK2X DCM's LOCKED output, if the DCM exists, otherwise the CGO.CLKLOCK output is connected to the main clock DCM's LOCKED output.

If PCI clock generation is enabled via the *pcien* VHDL generic the core instantiates either a BUFG or a BUFGDLL as depicted in figure 62 below. Note that the PCI clock must be enabled if the main clock is to be driven by the PCICLKIN input. If the PCI clock is disabled the PCICLK output is driven to zero. The CGO.PCILOCK output is always driven high in all configurations.

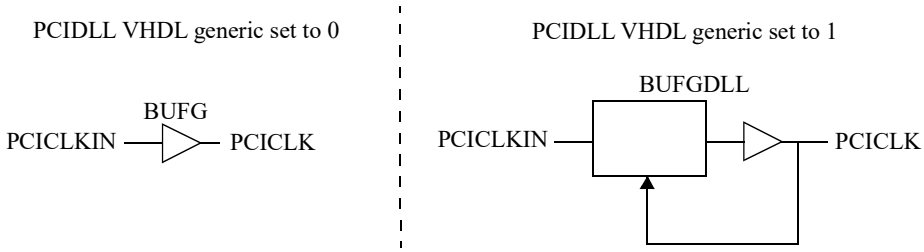


Figure 62. Virtex 5 PCI clock generation

28.2.15 eASIC90 (Nextreme)

Generics used in this technology:	clk_mul, clk_div, freq, pcisysclk, pcien
Instantiated technology primitives:	eclkgen
Signals not driven in this technology:	sdclk, pciclk, clk1xu, clk2xu, clkb, clkc

Please contact Frontgrade Gaisler for information concerning the use of this clock generator.

28.2.16 eASIC45 (Nextreme2)

Generics used in this technology:	clk_mul, clk_div, freq, pcisysclk, pcien, sdramen, clk2xen
Instantiated technology primitives:	eclkgen
Signals not driven in this technology:	clk1xu, clk2xu, clkb, clkc

An example instantiating eASIC's clock generator wrapper that generates clk, clkn and clk2x is provided. Note that the example does not instantiate buffers on the clock outputs. Please contact Frontgrade Gaisler for information concerning the use of this clock generator.

28.2.17 Actel Fusion

Generics used in this technology:	clk_mul, clk_div, clk_odiv, pcisysclk, pcien, freq, clkb_odiv, clkc_odiv
Instantiated technology primitives:	PLLINT, PLL
Signals not driven in this technology:	clkn, sdclk, clk2x, clk4x, clk1xu, clk2xu

This technology instantiates a PLL and a PLLINT to generate the main clock. The instantiation of a PLLINT macro allows the PLL reference clock to be driven from an I/O that is routed through the regular FPGA routing fabric. Figure 63 shows the instantiated primitives, the PLL EXTFB input is not shown and the EXTFB port on the instantiated component is always tied to ground. The OADIVRST port on the PLL is driven by CGI.PLLRST. The figure shows which of the core's output ports that are driven by the PLL. The PCICLOCK will directly connected to PCICLKIN if VHDL generic *pcien* is non-zero, while CGO.PCILOCK is always driven high. The VHDL generics *pcien* and *pcisysclk* are used to select the reference clock. The values driven on the PLL inputs are listed in tables 244 and 245.

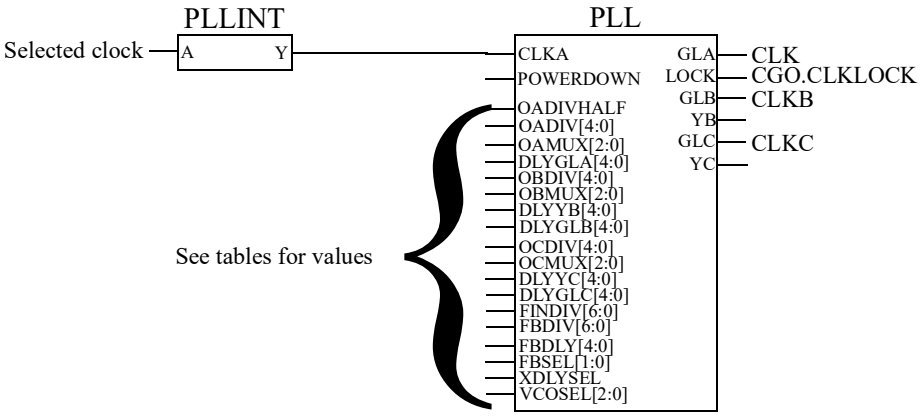


Figure 63. Actel Fusion clock generation

*Table 244.*Constant input signals on Actel Fusion PLL

Signal name	Value	Comment
OADIVHALF	0	Division by half
OADIV[4:0]	VHDL generic *clk_odiv* - 1	Output divider
OAMUX[2:0]	0b100	Post-PLL MUXA
DLYGLA[4:0]	0	Delay on Global A
OBDIV[4:0]	VHDL generic *clkb_odiv* - 1 when *clkb_odiv* > 0, otherwise 0	Output divider
OBMUX[2:0]	0 when VHDL generic *clkb_odiv* = 0, otherwise 0b100	Post-PLL MUXB

Table 244. Constant input signals on Actel Fusion PLL

Signal name	Value	Comment
DLYYB[4:0]	0	Delay on YB
DLYGLB[4:0]	0	Delay on Global B
OCDIV[4:0]	VHDL generic *clkc_odiv* - 1 when *clkc_odiv* > 0, otherwise 0	Output divider
OCMUX[2:0]	0 when VHDL generic *clkc_odiv* = 0, otherwise 0b100	Post-PLL MUXC
DLYYC[4:0]	0	Delay on YC
DLYGLC[4:0]	0	Delay on Global C
FINDIV[6:0]	VHDL generic *clk_div* - 1	Input divider
FBDIV[6:0]	VHDL generic *clk_mul* - 1	Feedback divider
FBDLY[4:0]	0	Feedback delay
FBSEL[1:0]	0b01	2-bit PLL feedback MUX
XDLYSEL	0	1-bit PLL feedback MUX
VCOSEL[2:0]	*See table 234 below*	VCO gear control. Selects one of four frequency ranges.

The PLL primitive has one parameter, VCOFREQUENCY, which is calculated with:

$$VCOFREQUENCY = \frac{freq \cdot clkmul}{clkdiv} / 1000$$

The calculations are performed with integer precision. This value is also used to determine the value driven on PLL input VCOSEL[2:0]. Table 234 lists the signal value depending on the value of VCOFREQUENCY.

Table 245. VCOSEL[2:0] on Actel Fusion PLL

Value of VCOFREQUENCY	Value driven on VCOSEL[2:0]
< 44	0b000
< 88	0b010
< 175	0b100
>= 175	0b110

28.2.18 Altera Stratix 4

This technology is not fully supported at this time.

28.2.19 NanoXplore Brave Medium

Generics used in this technology:	clk_mul,noclkfb, pcisysclk, pcien, freq, clksel, clk_odiv, clkb_odiv, clkc_odiv,
Instantiated technology primitives:	NX_CKB,NX_PLL,NX_WFG
Signals not driven in this technology:	clkn, sdclk, clk4x, clk1xu, clk2xu,clk8x

For this technology, the input clock frequency has to be in range 20-200 MHz.

For the correct use of the clkgen, it is necessary to specify the generic **freq**, which indicates the input clock frequency in KHz, and the generic **clksel**. In particular, the generic **clksel** selects if the clkgen will use the PLL internal feedback or the external feedback. In fact, the implementation makes use of

a NX_PLL primitive, which allows the user to select a feedback internal to the primitive itself or external to it. Since the implementation is quite different depending on the value of **clksel**, the two possibilities are described separately in the following sections.

clksel = 0

For this implementation, the NX_PLL makes use of the feedback path internal to the primitive. The clock generation is supported only if the condition

$$\text{freq} \times \text{clk_mul} \times 2 < 1200\ \text{MHz}$$

is satisfied.

The internal feedback allows the multiplication of the input clock frequency for any number from 1 to 15 but the generated clock won't be phase aligned with the reference clock.

In this case, three clocks are generated: ***clk***, ***clkb*** and ***clkc***. Their frequency depends on the values of the generics **freq**, **clk_mul**, **clk_odiv**, **clkb_odiv** and **clkc_odiv**. In particular:

if **freq** < 100 Mhz then

$$\left(\text{clk_freq} = \frac{freq \times \text{clk_mul} \times 2}{2^{\text{clk_odiv}}}\right) \quad \left(\text{clkb_freq} = \frac{freq \times \text{clk_mul} \times 2}{2^{\text{clkb_odiv}}}\right) \quad \left(\text{clkc_freq} = \frac{freq \times \text{clk_mul} \times 2}{2^{\text{clkc_odiv}}}\right)$$

if **freq** > 100 Mhz then

$$\left(\text{clk_freq} = \frac{freq \times \text{clk_mul}}{2^{\text{clk_odiv}}}\right) \quad \left(\text{clkb_freq} = \frac{freq \times \text{clk_mul}}{2^{\text{clkb_odiv}}}\right) \quad \left(\text{clkc_freq} = \frac{freq \times \text{clk_mul}}{2^{\text{clkc_odiv}}}\right)$$

clksel = 1

For this implementation the NX_PLL makes use of the external feedback. The generated clock will be phase aligned with the reference clock but it will also be at the same frequency as the reference clock (the generic **clk_mul** has to be set to 1).

By setting the generic "**noclkfb**" to 1, the feedback will be handled inside the clkgen module. Instead, if "**noclkfb**" is set to 0 the feedback has to be handled externally and the feedback clock has to be provided through the input *cgi.pllref*.

In this case, the clkgen generates two clocks: *clk* and *clk2x*.

The first one, *clk*, has a frequency equal to the reference clock. Instead, *clk2x* doubles the input clock frequency if

$$\text{freq} \leq 100 Mhz$$

Otherwise *clk2x* has also the same frequency as the reference clock.

Both for **clksel** = 1 or 0, the output *cgo.clklock* (active high) signals if the PLL is locked and if it generates stable clocks. Instead, the output *cgo.pcilock* is always high.

Moreover, there are several possibilities regarding the PCI clock :

- If **pcien**= 0 then the input *pciclkin* is not used and the output *pciclk* is set low
- If **pcien**/= 0 then *pciclk* output is fed through a buffer by the *pciclkin* input
- If **pcien** /= 0 and **pcisysclk** is /=0 then *pciclkin* input is also used as the main clock reference and the *clkin* input is not used.

In the last case, the generic **freq** has to be set to the value of the *pciclkin* frequency in KHz.

28.3 Configuration options

Table 246 shows the configuration options of the core (VHDL generics).

*Table 246.*Configuration options

Generic name	Function	Allowed range	Default
tech	Target technology	0 - NTECH	inferred
clk_mul	Clock multiplier, used in clock scaling. Not all techbolo-gies support clock scaling.		1
clk_div	Clock divisor, used in clock scaling. Not all technologies support clock scaling.		1
sdramen	When this generic is set to 1 the core will generate a clock on the SDCLK. Not supported by all technologies. See technology specific description.		0
noclkfb	When this generic is set to 0 the core will use the CGI.PLLREF input as feedback clock for some technol-ogies. See technology specific description.		1
pcien	When this generic is set to 1 the PCI clock is activated. Otherwise the PCICLKIN input is typically unused. See technology specific descriptions.		0
pcidll	When this generic is set to 1, a DLL will be instantiated for the PCI input clock for some technologies. See the technology specific descriptions.		0
pcisysclk	When this generic is set to 1 the clock generator will use the pciclkin input as the main clock reference. This also requires generic pcien to be set to 1.		0
freq	Clock frequency in kHz		25000
clk2xen	Enables 2x clock output. Not available in all technolgies and may have additional options. See technology spe-cific description.		0
clksel	Enable clock select. Not available in all technologies.		0
clk_odiv	ProASIC3/Fusion output divider for GLA. Only used in ProASIC3/Fusion technology.	1 - 32	1
clkb_odiv	ProASIC3/Fusion output divider for GLB. Only used in ProASIC3/Fusion technology. Set this value to 0 to dis-able generation of GLB.	0 - 32	0
clkc_odiv	ProASIC3/Fusion output divider for GLC. Only used in ProASIC3/Fusion technology. Set this value to 0 to dis-able generation of GLC.	0 - 32	0

28.4 Signal descriptions

Table 247 shows the interface signals of the core (VHDL ports).

*Table 247.*Signal descriptions

Signal name	Field	Type	Function	Active
CLKIN	N/A	Input	Reference clock input	-
PCICLKIN	N/A	Input	PCI clock input	
CLK	N/A	Output	Main clock	-
CLKN	N/A	Output	Inverted main clock	-
CLK2X	N/A	Output	2x clock	-
SDCLK	N/A	Output	SDRAM clock	-
PCICLK	N/A	Output	PCI clock	-
CGI	PLLREF	Input	Optional reference for PLL	-
	PLLRST	Input	Optional reset for PLL	
	PLLCTRL	Input	Optional control for PLL	
	CLKSEL	Input	Optional clock select	
CGO	CLKLOCK	Output	Lock signal for main clock	
	PCILOCK	Output	Lock signal for PCI clock	
CLK4X	N/A	Output	4x clock	
CLK1XU	N/A	Output	Unscaled 1x clock	
CLK2XU	N/A	Output	Unscaled 2x clock	
CLKB	N/A	Output	GLB output from ProASIC3/Fusion PLL	-
CLKC	N/A	Output	GLC output from ProASIC3/Fusion PLL	

28.5 Signal definitions and reset values

The signals and their reset values are described in table 248.

*Table 248.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
clk	Input	System clock	Rising edge	-

28.6 Timing

The timing waveforms and timing parameters are shown in figure 64 and are defined in table 249.

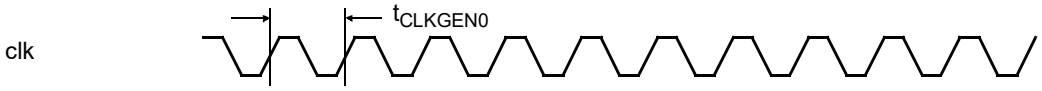


Figure 64. Timing waveforms

*Table 249.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{CLKGEN0}$	clock period	-	TBD	-	ns

28.7 Library dependencies

Table 250 shows the libraries used when instantiating the core (VHDL libraries).

*Table 250.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Component, signals	Core signal definitions
TECHMAP	ALLCLKGEN	Component	Technology specific CLKGEN components

28.8 Instantiation

This example shows how the core can be instantiated together with the GRLIB reset generator.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;
library gaisler;
use gaisler.misc.all;

entity clkgen_ex is
  port (
    resetn  : in  std_ulogic;
    clk  : in  std_ulogic; -- 50 MHz main clock
    pllref   : in  std_ulogic
  );
end;

architecture example of clkgen_ex is

signal lclk, clkm, rstn, rstraw, sdclkl, clk50: std_ulogic;
signal cgi   : clkgen_in_type;
signal cgo   : clkgen_out_type;

begin
  cgi.pllctrl <= "00"; cgi.pllrst <= rstraw;

  pllref_pad : clkpad generic map (tech => padtech) port map (pllref, cgi.pllref);

  clk_pad : clkpad generic map (tech => padtech) port map (clk, lclk);

  clkgen0 : clkgen  -- clock generator
    generic map (clktech, CFG_CLKMUL, CFG_CLKDIV, CFG_MCTRL_SDEN,
                 CFG_CLK_NOFB, 0, 0, 0, BOARD_FREQ)
    port map (lclk, lclk, clkm, open, open, sdclkl, open, cgi, cgo, open, clk50);

  sdclk_pad : outpad generic map (tech => padtech, slew => 1, strength => 24)
    port map (sdclk, sdclkl);

  resetn_pad : inpad generic map (tech => padtech) port map (resetn, rst);

  rst0 : rstgen  -- reset generator
    port map (rst, clkm, cgo.clklock, rstn, rstraw);

end;
```

29 DDRSPA - 16-, 32- and 64-bit DDR266 Controller

29.1 Overview

DDRSPA is a DDR266 SDRAM controller with AMBA AHB back-end. The controller can interface two 16-, 32- or 64-bit DDR266 memory banks to a 32-bit AHB bus. The controller acts as a slave on the AHB bus where it occupies a configurable amount of address space for DDR SDRAM access. The DDR controller is programmed by writing to a configuration register mapped located in AHB I/O address space. Internally, DDRSPA consists of a ABH/DDR controller and a technology specific DDR PHY. For currently supported technologies for the PHY see section 29.7.2. The modular design of DDRSPA allows to add support for other target technologies in a simple manner.

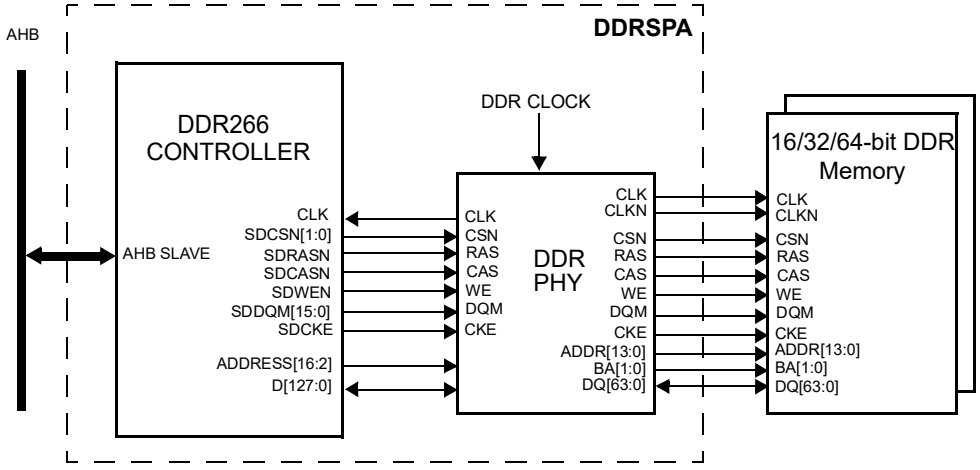


Figure 65. DDRSPA Memory controller conected to AMBA bus and DDR SDRAM

29.2 Operation

29.2.1 General

Double data-rate SDRAM (DDR RAM) access is supported to two banks of 16-, 32- or 64-bit DDR266 compatible memory devices. The controller supports 64M, 128M, 256M, 512M and 1G devices with 9- 12 column-address bits, up to 14 row-address bits, and 4 internal banks. The size of each of each chip select can be programmed in binary steps between 8 Mbyte and 1024 Mbyte. The DDR data width is set by the *ddrbits* VHDL generic, and will affect the width of DM, DQS and DQ signals. The DDR data width does not change the behavior of the AHB interface, except for data latency. When the VHDL generic *mobile* is set to a value not equal to 0, the controller supports mobile DDR SDRAM (LPDDR).

29.2.2 Read cycles

An AHB read access to the controller will cause a corresponding access to the external DDR RAM. The read cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a READ command. CAS latency of 2 (CL=2) or 3 (CL=3) can be used. Byte, half-word (16-bit) and word (32-bit) AHB accesses are supported. Incremental AHB burst access are supported for 32-bit words only. The read cycle(s) are always terminated with a PRE-CHARGE command, no banks are left open between two accesses. DDR read cycles are always performed in (aligned) 8-word bursts, which are stored in a FIFO. After an initial latency, the data is then read out on the AHB bus with zero waitstates.

29.2.3 Write cycles

Write cycles are performed similarly to read cycles, with the difference that WRITE commands are issued after activation. An AHB write burst will store up to 8 words in a FIFO, before writing the data to the DDR memory. As in the read case, only word bursts are supported

29.2.4 Initialization

If the *pwron* VHDL generic is 1, then the DDR controller will automatically perform the DDR initialization sequence as described in the JEDEC DDR266 standard: PRE-CHARGE, LOAD-EXTMODE-REG, LOAD-MODE-REG, PRE-CHARGE, 2xREFRESH and LOAD-MODE-REG; or as described in the JEDEC LPDDR standard when mobile DDR is enabled: PRE-CHARGE, 2xREFRESH, LOAD-MODE-REG and LOAD-EXTMODE-REG. The VHDL generics *col* and *Mbyte* can be used to also set the correct address decoding after reset. In this case, no further software initialization is needed. The DDR initialization can be performed at a later stage by setting bit 15 in the DDR control register.

29.2.5 Configurable DDR SDRAM timing parameters

To provide optimum access cycles for different DDR devices (and at different frequencies), three timing parameters can be programmed through the memory configuration register (SDCFG): TRCD, TRP and TRFCD. The value of these field affects the SDRAM timing as described in table 251.

*Table 251.*DDR SDRAM programmable minimum timing parameters

SDRAM timing parameter	**Minimum timing (clocks)**
Precharge to activate (t_{RP})	TRP + 2
Auto-refresh command period (t_{RFC})	TRFC + 3
Activate to read/write (t_{RCD})	TRCD + 2
Activate to Activate (t_{RC})	TRCD + 8
Activate to Precharge (t_{RAS})	TRCD + 6

If the TCD, TRP and TRFC are programmed such that the DDR200/266 specifications are fulfilled, the remaining SDRAM timing parameters will also be met. The table below shows typical settings for 100 and 133 MHz operation and the resulting SDRAM timing (in ns):

*Table 252.*DDR SDRAM example programming

DDR SDRAM settings	**t_{RCD}**	**t_{RC}**	**t_{RP}**	**t_{RFC}**	**t_{RAS}**
100 MHz: CL=2, TRP=0, TRFC=4, TRCD=0	20	80	20	70	60
133 MHz: CL=2, TRP=1, TRFC=6, TRCD=1	22.5	75	22.5	67.5	52.5

When the DDRSPA controller uses CAS latency (CL) of two cycles a DDR SDRAM speed grade of -75Z or better is needed to meet 133 MHz timing.

When mobile DDR support is enabled, two additional timing parameters can be programmed though the Power-Saving configuration register.

*Table 253.*Mobile DDR SDRAM programmable minimum timing parameters

SDRAM timing parameter	**Minimum timing (clocks)**
Exit Power-down mode to first valid command (t_{XP})	TXP + 1
Exit Self Refresh mode to first valid command (t_{XSR})	TXSR + 1
CKE minimum pulse width (t_{CKE})	TCKE + 1

29.2.6 Extended timing fields

The DDRSPA controller can be configured with extended timing fields to provide support for DDR333 and DDR400. These fields can be detected by checking the XTF bit in the SDCFG register.

When the extended timing fields are enabled, extra upper bits are added to increase the range of the TRP, TRFC, TXSR and TXP fields. A new TWR field allow increasing the write recovery time. A new TRAS field to directly control the Active to Precharge period has been added.

*Table 254.*DDR SDRAM extended timing parameters

SDRAM timing parameter	Minimum timing (clocks)
Activate to Activate (t_{RC})	TRAS+TRCD + 2
Activate to Precharge (t_{RAS})	TRAS + 6
Write recovery time (t_{WR})	TWR+2

*Table 255.*DDR SDRAM extended timing example programming

DDR SDRAM settings	t_{RCD}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}	t_{WR}
166 MHz: CL=2, TRP=1, TRFC=9, TRCD=1, TRAS=1, TWR=1	18	60	18	72	42	18
200 MHz: CL=3, TRP=1, TRFC=11, TRCD=1, TRAS=2, TWR=1	15	55	15	70	40	15

29.2.7 Refresh

The DDRSPA controller contains a refresh function that periodically issues an AUTO-REFRESH command to both SDRAM banks. The period between the commands (in clock periods) is programmed in the refresh counter reload field in the SDCFG register. Depending on SDRAM type, the required period is typically 7.8 us (corresponding to 780 at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by bit 31 in SDCTRL register.

29.2.8 Self Refresh

The self refresh mode can be used to retain data in the SDRAM even when the rest of the system is powered down. When in the self refresh mode, the SDRAM retains data without external clocking and refresh are handled internally. The memory array that is refreshed during the self refresh operation is defined in the extended mode register. These settings can be changed by setting the PASR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the PASR bits are changed. The supported “Partial Array Self Refresh” modes are: Full, Half, Quarter, Eighth, and Sixteenth array. “Partial Array Self Refresh” is only supported when mobile DDR functionality is enabled. To enable the self refresh mode, set the PMODE bits in the Power-Saving configuration register to “010” (Self Refresh). The controller will enter self refresh mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits are cleared. When exiting this mode and mobile DDR is disabled, the controller introduce a delay of 200 clock cycles and a AUTO REFRESH command before any other memory access is allowed. When mobile DDR is enabled the delay before the AUTO REFRESH command is defined by tXSR in the Power-Saving configuration register. The minimum duration of this mode is defined by tRFC. This mode is only available when the VHDL generic *mobile* is >= 1.

29.2.9 Clock Stop

In the clock stop mode, the external clock to the SDRAM is stop at a low level (DDR_CLK is low and DDR_CLKB is high). This reduce the power consumption of the SDRAM while retaining the data. To enable the clock stop mode, set the PMODE bits in the Power-Saving configuration register to “100” (Clock Stop). The controller will enter clock stop mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits are cleared. The REFRESH command

will still be issued by the controller in this mode. This mode is only available when the VHDL generic *mobile* is >= 1 and mobile DDR functionality is enabled.

29.2.10 Power-Down

When entering the power-down mode all input and output buffers, including DDR_CLK and DDR_-CLKB and excluding DDR_CKE, are deactivated. This is a more efficient power saving mode then clock stop mode, with a grater reduction of the SDRAM's power consumption. All data in the SDRAM is retained during this operation. To enable the power-down mode, set the PMODE bits in the Power-Saving configuration register to "001" (Power-Down). The controller will enter power-down mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits is cleared. The REFRESH command will still be issued by the controller in this mode. When exiting this mode a delay of one or two (when tXP in the Power-Saving configuration register is '1') clock cycles are added before issue any command to the memory. This mode is only available when the VHDL generic *mobile* is >= 1.

29.2.11 Deep Power-Down

The deep power-down operating mode is used to achieve maximum power reduction by eliminating the power of the memory array. Data will not be retained after the device enters deep power-down mode. To enable the deep power-down mode, set the PMODE bits in the Power-Saving configuration register to "101" (Deep Power-Down). To exit the deep power-down mode the PMODE bits in the Power-Saving configuration register must be cleared followed by the mobile SDRAM initialization sequence. The mobile SDRAM initialization sequence can be performed by setting bit 15 in the DDR control register. This mode is only available when the VHDL generic *mobile* is >= 1 and mobile DDR functionality is enabled.

29.2.12 Status Read Register

The status read register (SRR) is used to read the manufacturer ID, revision ID, refresh multiplier, width type, and density of the SDRAM. To Read the SSR a LOAD MODE REGISTER command with BA0 = 1 and BA1 = 0 must be issued followed by a READ command with the address set to 0. This command sequence is executed then the Status Read Register is read. This register is only available when the VHDL generic *mobile* is >= 1 and mobile DDR functionality is enabled. Only DDR_CSB[0] is enabled during this operation.

29.2.13 Temperature-Compensated Self Refresh

The settings for the temperature-compensation of the Self Refresh rate can be controlled by setting the TCSR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the TCSR bits are changed. Note that some vendors implements a Internal Temperature-Compensated Self Refresh feature, which makes the memory to ignore the TCSR bits. This functionality is only available when the VHDL generic *mobile* >= 1 and mobile DDR functionality is enabled.

29.2.14 Drive Strength

The drive strength of the output buffers can be controlled by setting the DS bits in the Power-Saving configuration register. The extended mode register is automatically updated when the DS bits are changed. The available options are: full, three-quarter, one-half, and one-quarter drive strengths. This functionality is only available when the VHDL generic *mobile* is >= 1 and mobile DDR functionality is enabled.

29.2.15 SDRAM commands

The controller can issue four SDRAM commands by writing to the SDRAM command field in SDCFG: PRE-CHARGE, LOAD-EXTMODE-REG, LOAD-MODE-REG and REFRESH. If the LEMR command is issued, the PLL Reset bit as programmed in SDCFG will be used, when mobile DDR support is enabled the DS, TCSR and PASR as programmed in Power-Saving configuration register will be used. If the LMR command is issued, the CAS latency as programmed in the Power-Saving configuration register will be used and remaining fields are fixed: 8 word sequential burst. The command field will be cleared after a command has been executed.

29.2.16 Clocking

The DDR controller is designed to operate with two clock domains, one for the DDR memory clock and one for the AHB clock. The two clock domains do not have to be the same or be phase-aligned. The DDR input clock (CLK_DDR) can be multiplied and divided by the DDR PHY to form the final DDR clock frequency. The final DDR clock is driven on one output (CLKDDRO), which should always be connected to the CLKDDRI input. If the AHB clock and DDR clock area generated from the same clock source, a timing-ignore constraint should be placed between the CLK_AHB and CLKDDRI to avoid optimization of false-paths during synthesis and place&route.

The Xilinx version of the PHY generates the internal DDR read clock using an external clock feed-back. The feed-back should have the same delay as DDR signals to and from the DDR memories. The feed-back should be driven by DDR_CLK_FB_OUT, and returned on DDR_CLK_FB. Most Xilinx FPGA boards with DDR provides clock feed-backs of this sort. The supported frequencies for the Xilinx PHY depends on the clock-to-output delay of the DDR output registers, and the internal delay from the DDR input registers to the read data FIFO. Virtex2 and Virtex4 can typically run at 120 MHz, while Spartan3e can run at 100 MHz.

The read data clock in the Xilinx version of the PHY is generated using a DCM to offset internal delay of the DDR clock feed back. If the automatic DCM phase adjustment does not work due to unsuitable pin selection, extra delay can be added through the RSKEW VHDL generic. The VHDL generic can be between -255 and 255, and is passed directly to the PHASE_SHIFT generic of the DCM.

The Altera version of the PHY use the DQS signals and an internal PLL to generate the DDR read clock. No external clock feed-back is needed and the DDR_CLK_FB_OUT/DDR_CLK_FB signals are not used. The supported frequencies for the Altera PHY are 100, 110, 120 and 130 MHz. For Altera CycloneIII, the read data clock is generated by the PLL. The phase shift of the read data clock is set be the VHDL generic RSKEW in ps (e.g. a value of 2500 equals 90’ phase for a 100MHz system).

29.2.17 Pads

The DDRSPA core has technology-specific pads inside the core. The external DDR signals should therefore be connected directly the top-level ports, without any logic in between.

29.2.18 Endianness

The core is designed for big-endian systems.

29.3 Registers

The DDRSPA core implements two control registers. The registers are mapped into AHB I/O address space defined by the AHB BAR1 of the core. Only 32-bit single-accesses to the registers are supported.

*Table 256.*DDR controller registers

Address offset - AHB I/O - BAR1	Register
0x00	SDRAM control register
0x04	SDRAM configuration register (read-only)
0x08	SDRAM Power-Saving configuration register
0x0C	Reserved
0x10	Status Read Register (Only available when mobile DDR support is enabled)
0x14	PHY configuration register 0 (Only available when VHDL generic confapi = 1, TCI RTL_PHY)
0x18	PHY configuration register 1 (Only available when VHDL generic confapi = 1, TCI TRL_PHY)

29.3.1 Control Register

Table 257. SDRAM control register (SDCTRL)

31	30	29 27	26	25 23	22 21	20 18	17	16	15	14 0
Refresh	tRP	tRFC	tRCD	SDRAM bank size	SDRAM col. size	SDRAM command	PR	IN	CE	SDRAM refresh load value

31 SDRAM refresh. If set, the SDRAM refresh will be enabled. This register bit is read only when Power-Saving mode is other then none.

30 SDRAM tRP timing. tRP will be equal to 2 or 3 system clocks (0/1). When mobile DDR support is enabled, this bit also represent the MSB in the tRFC timing.

29: 27 SDRAM tRFC timing. tRFC will be equal to 3 + field-value system clocks. When mobile DDR support is enabled, this field is extended with the bit 30.

26 SDRAM tRCD delay. Sets tRCD to 2 + field value clocks.

25: 23 SDRAM banks size. Defines the decoded memory size for each SDRAM chip select: “000”= 8 Mbyte, “001”= 16 Mbyte, “010”= 32 Mbyte .... “111”= 1024 Mbyte.

22: 21 SDRAM column size. “00”=512, “01”=1024, “10”=2048, “11”=4096

20: 18 SDRAM command. Writing a non-zero value will generate an SDRAM command: “010”=PRECHARGE, “100”=AUTO-REFRESH, “110”=LOAD-COMMAND-REGISTER, “111”=LOAD-EXTENDED-COMMAND-REGISTER. The field is reset after command has been executed.

17 PLL Reset. This bit is used to set the PLL RESET bit during LOAD-CONFIG-REG commands.

16 Initialize (IN). Set to ‘1’ to perform power-on DDR RAM initialisation. Is automatically cleared when initialisation is completed. This register bit is read only when Power-Saving mode is other then none.

15 Clock enable (CE). This value is driven on the CKE inputs of the DDR RAM. Should be set to ‘1’ for correct operation. This register bit is read only when Power-Saving mode is other then none.

14: 0 The period between each AUTO-REFRESH command - Calculated as follows: tREFRESH = ((reload value) + 1) / DDRCLOCK

29.3.2 Configuration Register

Table 258. SDRAM configuration register (SDCFG)

31 21	20	19 16	15	14 12	11 0
Reserved	XTF	CONFAPI	MD	Data width	DDR Clock frequency

31: 21 Reserved

20 Extended timing fields for DDR400 available

19: 16 Register API configuration.
0 = Standard register API.
1 = TCI TSMC90 PHY register API.

15 Mobile DDR support enabled. ‘1’ = Enabled, ‘0’ = Disabled (read-only)

14: 12 DDR data width: “001” = 16 bits, “010” = 32 bits, “011” = 64 bits (read-only)

11: 0 Frequency of the (external) DDR clock (read-only)

29.3.3 Power-Saving Configuration Register

*Table 259.*SDRAM Power-Saving configuration register

31	30	29 28	27 26	25	24	23 20	19	18 16	15 12	11	10	9	8 7	5	4	3 2	0
ME	CL	TRAS	xXS*	xXP	tC	tXSR	tXP	PMODE	Reserved	TWR	xTRP	xTRFC	DS		TCSR		PASR

Frontgrade Gaisler AB
Kungsgatan | SE-411 19 | Goteborg | Sweden
+46 31 7758650 | frontgrade.com/gaisler

Table 259. SDRAM Power-Saving configuration register

31	Mobile DDR functionality enabled. '1' = Enabled (support for Mobile DDR SDRAM), '0' = disabled (support for standard DDR SDRAM)
30	CAS latency; '0' => CL = 2, '1' => CL = 3
29: 28	SDRAM extended tRAS timing, tRAS will be equal to field-value + 6 system clocks. (Reserved when extended timing fields are disabled)
27: 26	SDRAM extended tXSR field, extend tXSR with field-value * 16 clocks (Reserved when extended timing fields are disabled)
25	SDRAM extended tXP field, extend tXP with 2*field-value clocks (Reserved when extended timing fields are disabled)
24	SDRAM tCKE timing, tCKE will be equal to 1 or 2 clocks (0/1). (Read only when Mobile DDR support is disabled).
23: 20	SDRAM tXSR timing. tXSR will be equal to field-value system clocks. (Read only when Mobile DDR support is disabled).
19	SDRAM tXP timing. tXP will be equal to 2 or 3 system clocks (0/1). (Read only when Mobile DDR support is disabled).
18: 16	Power-Saving mode (Read only when Mobile DDR support is disabled). "000": none "001": Power-Down (PD) "010": Self-Refresh (SR) "100": Clock-Stop (CKS) "101": Deep Power-Down (DPD)
15: 12	Reserved
11	SDRAM extended tWR timing, tWR will be equal to field-value + 2 clocks (Reserved when extended timing fields are disabled)
10	SDRAM extended tRP timing, extend tRP with field-value * 2 clocks
9: 8	SDRAM extended tRFC timing, extend tRFC with field-value * 8 clocks
7: 5	Selectable output drive strength (Read only when Mobile DDR support is disabled). "000": Full "001": One-half "010": One-quarter "011": Three-quarter
4: 3	Reserved for Temperature-Compensated Self Refresh (Read only when Mobile DDR support is disabled). "00": 70ªC "01": 45ªC "10": 15ªC "11": 85ªC
2: 0	Partial Array Self Refresh (Read only when Mobile DDR support is disabled). "000": Full array (Banks 0, 1, 2 and 3) "001": Half array (Banks 0 and 1) "010": Quarter array (Bank 0) "101": One-eighth array (Bank 0 with row MSB = 0) "110": One-sixteenth array (Bank 0 with row MSB = 00)

29.3.4 Status Read Register

Table 260. Status Read Register

31 16	15 0
SRR_16	SRR

31: 16 Status Read Register when 16-bit DDR memory is used (read only)

15: 0 Status Read Register when 32/64-bit DDR memory is used (read only)

29.3.5 PHY Configuration Register 0

Table 261. PHY configuration register 0 (TCI RTL_PHY only)

31	30	29	28	27 22	21 16	15 8	7 0
R1	R0	P1	P0	TSTCTRL1	TSTCTRL0	MDAJ_DLL1	MDAJ_DLL0

31 Reset DLL 1 (active high)

30 Reset DLL 1 (active high)

29 Power Down DLL 1 (active high)

28 Power Down DLL 1 (active high)

27: 22 Test control DLL 1
tstclkin(1) is connected to SIGI_1 on DDL 1 when bit 26:25 is NOT equal to “00“.
tstclkin(0) is connected to SIGI_0 on DDL 1 when bit 23:22 is NOT equal to “00“.

21: 16 Test control DLL 0

15: 8 Master delay adjustment input DLL 1

7: 0 Master delay adjustment input DLL 0

29.3.6 PHY Configuration Register 1

Table 262. PHY configuration register 1 (TCI RTL_PHY only)

31 24	23 16	15 8	7 0
ADJ_RSYNC	ADJ_90	ADJ_DQS1	ADJ_DQS0

31: 24 Slave delay adjustment input for resync clock (Slave 1 DLL 1)

23: 16 Slave delay adjustment input for 90’ clock (Slave 0 DLL 1)

15: 8 Slave delay adjustment input for DQS 1 (Slave 1 DLL 0)

7: 0 Slave delay adjustment input for DQS 0 (Slave 0 DLL 0)

29.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x025. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

29.5 Implementation

29.5.1 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User’s Manual). The core is designed for big-endian systems.

29.6 Configuration options

Table 263 shows the configuration options of the core (VHDL generics).

*Table 263.*Configuration options

Generic	Function	Allowed range	Default
fabtech	PHY technology selection	virtex2, virtex4, spartan3e, altera	virtex2
memtech	Technology selection for DDR FIFOs	infered, virtex2, virtex4, spartan3e, altera	infered
hindex	AHB slave index	0 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR0 defining SDRAM area. Default is 0xF0000000 - 0xFFFFFFFF.	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0 defining SDRAM area.	0 - 16#FFF#	16#F00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space where DDR control register is mapped.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR1 defining I/O address space	0 - 16#FFF#	16#FFF#
ddrbits	Data bus width of external DDR memory	16, 32, 64	16
MHz	DDR clock input frequency in MHz.	10 - 200	100
clkmul, clkdiv	The DDR input clock is multiplied with the clkmul generic and divided with clkdiv to create the final DDR clock	2 - 32	2
rstdel	Clock reset delay in micro-seconds.	1 - 1023	200
col	Default number of column address bits	9 - 12	9
Mbyte	Default memory chip select bank size in Mbyte	8 - 1024	16
pwron	Enable SDRAM at power-on initialization	0 - 1	0
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
ahbfreq	Frequency in MHz of the AHB clock domain	1 - 1023	50
rskew	Additional read data clock skew Read data clock phase for Altera CycloneIII	-255 - 255. 0 - 9999	0
mobile	Enable Mobile DDR support 0: Mobile DDR support disabled 1: Mobile DDR support enabled but not default 2: Mobile DDR support enabled by default 3: Mobile DDR support only (no regular DDR support)	0 - 3	0
confapi	Set the PHY configuration register API: 0 = standard register API (conf0 and conf1 disabled). 1 = TCI RTL_PHY register API.		
conf0	Reset value for PHY register 0, conf[31:0]	0 - 16#FFFFFFFF#	0
conf1	Reset value for PHY register1, conf[63:32]	0 - 16#FFFFFFFF#	0
regoutput	Enables registers on signal going from controller to PHY	0 - 1	0
ddr400	Enables extended timing fields for DDR400 support	0 - 1	1
scantest	Enable scan test support	0 - 1	0
phyiconf	PHY implementation configuration. This generic sets technology specific implementation options for the DDR PHY. Meaning of values depend on the setting of VHDL generic *fabtech*. For fabtech:s virtex4, virtex5, virtex6: phyiconf selects type of pads used for DDR clock pairs. 0 instantiates a differential pad and 1 instantiates two outpads.	0 - 16#FFFFFFFF#	0

29.7 Implementation

29.7.1 Technology mapping

The core has two technology mapping VHDL generics: *memtech* and *fabtech*. The VHDL generic *memtech* controls the technology used for memory cell implementation. The VHDL generic *fabtech* controls the technology used in the PHY implementation. See the GRLIB Users's Manual for available settings.

29.7.2 FPGA support

Complete PHY:s for a number FPGA technologies are included in the distribution, see table below. Unless otherwise noted these have been only functionally tested on evaluation board in lab environment and detailed timing analysis has not been performed. Note also that some of the FPGA phy:s use simplified sampling approaches which may require the memory timing to be better than the JEDEC standard specifies.

Scripts for post-layout static timing analysis are not included. Because these PHY:s are based on dedicated hard macros with fixed placement in the FPGA:s pad structure, just a minimal set of constraints are normally neccessary for synthesis purposes.

*Table 264.*FPGA DDR PHYs included in GRLIB

Technology	fabtech	Read clock method	Built-in pads
Virtex4,5,6	virtex4, virtex5, virtex6	Clock feedback loop + static shift	Yes
Virtex2, Spartan3	virtex2, spartan3	Clock feedback loop + static shift	Yes
Spartan3E,6	spartan3	Clock feedback loop + static shift	Yes
Stratix II	stratix2	Tech intrinsics (DQS based)	Yes
Cyclone 3	cyclone3	Static shift	Yes

29.7.3 RAM usage

The FIFOs in the core are implemented with the *syncram_2p* (with separate clock for each port) component found in the technology mapping library (TECHMAP). The number of RAMs used for the FIFO implementation depends on the DDR data width, set by the *ddrbits* VHDL generic.

*Table 265.*RAM usage

RAM dimension (depth x width)	Number of RAMs (DDR data width 64)	Number of RAMs (DDR data width 32)	Number of RAMs (DDR data width 16)
4 x 128	1		
4 x 32	4		
5 x 64		1	
5 x 32		2	
6 x 32			2

29.8 Signal descriptions

Table 266 shows the interface signals of the core (VHDL ports).

*Table 266.*Signal descriptions

Signal name	Type	Function	Active
RST_DDR	Input	Reset input for DDR clock domain	Low
RST_AHB	Input	Reset input for AHB clock domain	Low
CLK_DDR	Input	DDR input Clock	-
CLK_AHB	Input	AHB clock	-
LOCK	Output	DDR clock generator locked	High
CLKDDRO		Internal DDR clock output after clock multiplication	
CLKDDRI		Clock input for the internal DDR clock domain. Must be connected to CLKDDRO.	
AHBSI	Input	AHB slave input signals	-
AHBSO	Output	AHB slave output signals	-
DDR_CLK[2:0]	Output	DDR memory clocks (positive)	High
DDR_CLKB[2:0]	Output	DDR memory clocks (negative)	Low
DDR_CLK_FB_OUT	Output	Same a DDR_CLK, but used to drive an external clock feedback.	-
DDR_CLK_FB	Input	Clock input for the DDR clock feed-back	-
DDR_CKE[1:0]	Output	DDR memory clock enable	High
DDR_CSB[1:0]	Output	DDR memory chip select	Low
DDR_WEB	Output	DDR memory write enable	Low
DDR_RASB	Output	DDR memory row address strobe	Low
DDR_CASB	Output	DDR memory column address strobe	Low
DDR_DM[DDRBITS/8-1:0]	Output	DDR memory data mask	Low
DDR_DQS[DDRBITS/8-1:0]	Bidir	DDR memory data strobe	Low
DDR_AD[13:0]	Output	DDR memory address bus	Low
DDR_BA[1:0]	Output	DDR memory bank address	Low
DDR_DQ[DDRBITS-1:0]	BiDir	DDR memory data bus	-

1) see GRLIB IP Library User's Manual 2) Polarity selected with the oepol generic

29.9 Library dependencies

Table 267 shows libraries used when instantiating the core (VHDL libraries).

*Table 267.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

29.10 Component declaration

```
component ddrspa
  generic (
    fabtech : integer := 0;
    memtech : integer := 0;
    hindex  : integer := 0;
    haddr   : integer := 0;
    hmask   : integer := 16#f00#;
    ioaddr  : integer := 16#000#;
    iomask  : integer := 16#fff#;
    MHz     : integer := 100;
    clkmul  : integer := 2;
    clkdiv  : integer := 2;
    col     : integer := 9;
    Mbyte   : integer := 16;
    rstdel  : integer := 200;
    pwron   : integer := 0;
    oepol   : integer := 0;
    ddrbits : integer := 16;
    ahbfreq : integer := 50
  );
  port (
    rst_ddr : in  std_ulogic;
    rst_ahb : in  std_ulogic;
    clk_ddr : in  std_ulogic;
    clk_ahb : in  std_ulogic;
    lock    : out std_ulogic;-- DCM locked
    clkddro : out std_ulogic;-- DCM locked
    clkddri : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;
    ahbso   : out ahb_slv_out_type;
    ddr_clk : out std_logic_vector(2 downto 0);
    ddr_clkb: out std_logic_vector(2 downto 0);
    ddr_clk_fb_out  : out std_logic;
    ddr_clk_fb  : in std_logic;
    ddr_cke  : out std_logic_vector(1 downto 0);
    ddr_csb  : out std_logic_vector(1 downto 0);
    ddr_web  : out std_ulogic;                        -- ddr write enable
    ddr_rasb  : out std_ulogic;                        -- ddr ras
    ddr_casb  : out std_ulogic;                        -- ddr cas
    ddr_dm   : out std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dm
    ddr_dqs  : inout std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dqs
    ddr_ad     : out std_logic_vector (13 downto 0);   -- ddr address
    ddr_ba     : out std_logic_vector (1 downto 0);    -- ddr bank address
    ddr_dq    : inout  std_logic_vector (ddrbits-1 downto 0) -- ddr data

  );
  end component;
```

29.11 Instantiation

This examples shows how the core can be instantiated.

The DDR SDRAM controller decodes SDRAM area at 0x40000000 - 0x7FFFFFFF. The SDRAM registers are mapped into AHB I/O space on address (AHB I/O base address + 0x100).

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;

entity ddr_Interface is
  port ( ddr_clk  : out std_logic_vector(2 downto 0);
    ddr_clkb  : out std_logic_vector(2 downto 0);
    ddr_clk_fb  : in std_logic;
    ddr_clk_fb_out  : out std_logic;
    ddr_cke  : out std_logic_vector(1 downto 0);
    ddr_csb  : out std_logic_vector(1 downto 0);
    ddr_web  : out std_ulogic;                        -- ddr write enable
    ddr_rasb  : out std_ulogic;                        -- ddr ras
    ddr_casb  : out std_ulogic;                        -- ddr cas
    ddr_dm   : out std_logic_vector (7 downto 0);    -- ddr dm
    ddr_dqs  : inout std_logic_vector (7 downto 0);    -- ddr dqs
    ddr_ad      : out std_logic_vector (13 downto 0);   -- ddr address
    ddr_ba      : out std_logic_vector (1 downto 0);    -- ddr bank address
    ddr_dq  : inout std_logic_vector (63 downto 0); -- ddr data

);
end;

architecture rtl of mctrl_ex is

  -- AMBA bus
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
signal clkml, lock : std_ulogic;


begin

-- DDR controller


ddrc : ddrspa generic map ( fabtech => virtex4, ddrbits => 64, memtech => memtech,
hindex => 4, haddr => 16#400#, hmask => 16#F00#, ioaddr => 1,
pwron => 1, MHz => 100, col => 9, Mbyte => 32, ahbfreq => 50, ddrbits => 64)
port map (
rstneg, rstn, lclk, clkm, lock, clkml, clkml,  ahbsi, ahbso(4),
ddr_clk, ddr_clkb, ddr_clk_fb_out, ddr_clk_fb,
ddr_cke, ddr_csb, ddr_web, ddr_rasb, ddr_casb,
ddr_dm, ddr_dqs, ddr_adl, ddr_ba, ddr_dq);
```

30 DDR2SPA - 16-, 32- and 64-bit Single-Port Asynchronous DDR2 Controller

30.1 Overview

DDR2SPA is a DDR2 SDRAM controller with AMBA AHB back-end. The controller can interface 16-, 32- or 64-bit wide DDR2 memory with one or two chip selects. The controller acts as a slave on the AHB bus where it occupies a configurable amount of address space for DDR2 SDRAM access. The DDR2 controller is programmed by writing to configuration registers mapped located in AHB I/O address space.

Internally, DDR2SPA consists of a ABH/DDR2 controller and a technology specific DDR2 PHY. For currently supported technologies for the PHY, see section 30.8.2. The modular design of DDR2SPA allows to add support for other target technologies in a simple manner.

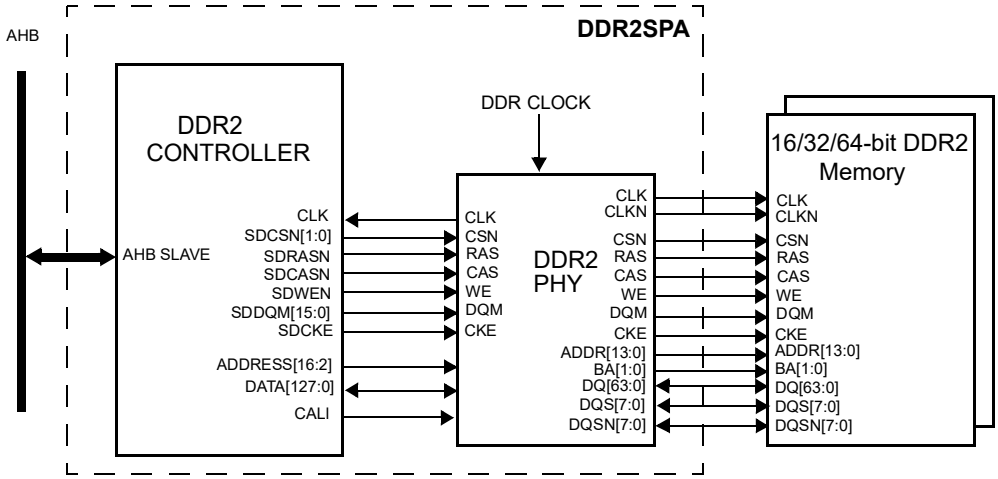


Figure 66. DDR2SPA Memory controller connected to AMBA bus and DDR2 SDRAM

30.2 Operation

30.2.1 General

Single DDR2 SDRAM chips are typically 4,8 or 16 data bits wide. By putting multiple identical chips side by side, wider SDRAM memory banks can be built. Since the command signals are common for all chips, the memories behave as one single wide memory chip.

This memory controller supports one or two (identical) such 16/32/64-bit wide DDR2 SDRAM memory banks. The size of the memory can be programmed in binary steps between 8 Mbyte and 1024 Mbyte, or between 32 Mbyte and 4096 Mbyte. The DDR data width is set by the DDRBITS generic, and will affect the width of DM, DQS and DQ signals. The DDR data width does not change the behavior of the AHB interface, except for data latency.

30.2.2 Data transfers

An AHB read or write access to the controller will cause a corresponding access cycle to the external DDR2 RAM. The cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a sequence of READ or WRITE commands (the count depending on memory width and burst length setting). After the sequence, a PRECHARGE command is performed to deactivate the SDRAM bank.

All access types are supported, but only incremental bursts of 32 bit width and incremental bursts of maximum width (if wider than 32) are handled efficiently. All other bursts are handled as single-

accesses. For maximum throughput, incremental bursts of full AHB width with both alignment and length corresponding to the burstlen generic should be performed.

The maximum supported access size can be limited by using the ahbbits generic, which is set to the full AHB bus size by default. Accesses larger than this size are not supported.

The memory controller's FIFO has room for two write bursts which improves throughput, since the second write can be written into the FIFO while the first write is being written to the DDR memory.

In systems with high DDR clock frequencies, the controller may have to insert wait states for the minimum activate-to-precharge time (t_{RAS}) to expire before performing the precharge command. If a new AHB access to the same memory row is performed during this time, the controller will perform the access in the same access cycle.

30.2.3 Initialization

If the **pwron** VHDL generic is 1, then the DDR2 controller will automatically on start-up perform the DDR2 initialization sequence as described in the JEDEC DDR2 standard. The VHDL generics **col** and **Mbyte** can be used to also set the correct address decoding after reset. In this case, no further software initialization is needed except for enabling the auto-refresh function. If power-on initialization is not enabled, the DDR2 initialization can be started at a later stage by setting bit 16 in the DDR2 control register DDR2CFG1.

30.2.4 Big memory support

The total memory size for each chip select is set through the 3-bit wide SDRAM banks size field, which can be set in binary steps between 8 Mbyte and 1024 Mbyte. To support setting even larger memory sizes of 2048 and 4096 Mbyte, a fourth bit has been added to this configuration field.

Only 8 different sizes are supported by the controller, either the lower range of 8 MB - 1 GB, or the higher range of 32 MB - 4 GB. Which range is determined by the bigmem generic, and can be read by software through the DDR2CFG2 register.

30.2.5 Configurable DDR2 SDRAM timing parameters

To provide optimum access cycles for different DDR2 devices (and at different frequencies), six timing parameters can be programmed through the memory configuration registers: TRCD, TCL, TRTP, TWR, TRP and TRFC. For faster memories (DDR2-533 and higher), the TRAS setting also needs to be configured to satisfy timing. The value of these fields affects the DDR2RAM timing as described in table 268. Note that if the CAS latency setting is changed after initialization, this change needs also to be programmed into the memory chips by executing the Load Mode Register command.

*Table 268.*DDR2 SDRAM programmable minimum timing parameters

DDR2 SDRAM timing parameter	Minimum timing (clocks)
CAS latency, CL	TCL + 3
Activate to read/write command (t_{RCD})	TRCD + 2
Read to precharge (t_{RTP})	TRTP + 2
Write recovery time (t_{WR})	TWR-2
Precharge to activate (t_{RP})	TRP + 2
Activate to precharge (t_{RAS})	TRAS + 1
Auto-refresh command period (t_{RFC})	TRFC + 3

If TRCD, TCL, TRTP, TWR, TRP, TRFC and TRAS are programmed such that the DDR2 specifications are full filled, the remaining SDRAM timing parameters will also be met. The table below

shows typical settings for 130, 200 and 400 MHz operation and the resulting DDR2 SDRAM timing (in ns):

*Table 269.*DDR2 SDRAM example programming

DDR2 SDRAM settings	CL	t_{RCD}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}
130 MHz: TCL=0,TRCD=0,TRTP=0,TRP=0,TRAS=0,TRFC=7	3	15	76	15	76	61
200 MHz: TCL=0,TRCD=1,TRTP=0,TRP=1,TRAS=1,TRFC=13	3	15	60	15	80	45
400 MHz: TCL=2,TRCD=4,TRTP=1,TRP=4,TRAS=10,TRFC=29	5	15	60	15	80	45

30.2.6 Refresh

The DDR2SPA controller contains a refresh function that periodically issues an AUTO-REFRESH command to both SDRAM banks. The period between the commands (in clock periods) is programmed in the refresh counter reload field in the DDR2CFG1 register. Depending on SDRAM type, the required period is typically 7.8 us (corresponding to 780 at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by bit 31 in DDR2CFG1 register.

30.2.7 DDR2 SDRAM commands

The controller can issue four SDRAM commands by writing to the SDRAM command field in SDCFG1: PRE-CHARGE, LOAD-EXTMODE-REG, LOAD-MODE-REG and REFRESH. If the LMR command is issued, the PLL Reset bit as programmed in DDR2CFG1, CAS Latency setting as programmed in DDR2CFG4 and the WR setting from DDR2CFG3 will be used, remaining fields are fixed: 4 word sequential burst. If the LEMR command is issued, the OCD bits will be used as programmed in the DDR2CFG1 register, and all other bits are set to zero. The command field will be cleared after a command has been executed.

30.2.8 Registered SDRAM

Registered memory modules (RDIMM:s) have one cycle extra latency on the control signals due to the external register. They can be supported with this core by setting the REG bit in the DDR2CFG4 register.

This should not be confused with Fully-Buffered DDR2 memory, which uses a different protocol and is not supported by this controller.

30.2.9 Clocking

The DDR2 controller operates in two separate clock domains, one domain synchronous to the DDR2 memory and one domain synchronous to the AHB bus. The two clock domains do not have to be the same or be phase-aligned.

The clock for the DDR2 memory domain is generated from the controller's ddr_clk input via a technology-specific PLL component. The multiplication and division factor can be selected via the clkmul/clkdiv configuration options. The final DDR2 clock is driven on one output (CLKDDRO), which should always be connected to the CLKDDRI input.

The ddr_rst input asynchronously resets the PHY layer and the built-in PLL. The ahb_rst input should be reset simultaneously and then kept in reset until the PLL has locked (indicated by the lock output).

If the AHB and DDR2 clocks are based on the same source clock and are kept phase-aligned by the PLL, the clock domain transition is synchronous to the least common multiple of the two clock frequencies. In this case, the nosync configuration option can be used to remove the synchronization and handshaking between the two clock domains, which saves a few cycles of memory access latency. If nosync is not set in this case, a timing-ignore constraint should be placed between the CLK_AHB and CLKDDRI to avoid optimization of false-paths during synthesis and place&route.

The supported DDR2 frequencies depends on the clock-to-output delay of the DDR output registers, and the internal delay from the DDR input registers to the read data FIFO. Virtex5 can typically run at 200 MHz.

When reading data, the data bus (DQ) signals should ideally be sampled 1/4 cycle after each data strobe (DQS) edge. How this is achieved is technology-specific as described in the following sections.

30.2.10 Read data clock calibration on Xilinx Virtex

On Xilinx Virtex4/5 the data signal inputs are delayed via the I/O pad IDELAY feature to get the required 1/4 cycle shift. The delay of each byte lane is tuned independently between 0-63 tap delays, each tap giving 78 ps delay, and the initial value on startup is set via the generics ddelayb[7:0].

The delays can be tuned at runtime by using the DDR2CFG3 control register. There are two bits in the control register for each byte. One bit determines if the delay should be increased or decreased and the other bit is set to perform the update. Setting bit 31 in the DDR2CFG3 register resets the delays to the initial value.

To increase the calibration range, the controller can add additional read latency cycles. The number of additional read latency cycles is set by the RD bits in the DDR2CFG3 register.

30.2.11 Read data clock calibration on Altera Stratix

On Altera StratixIII, the technology's delay chain feature is used to delay bytes of input data in a similar fashion as the Virtex case above. The delay of each byte lane is tuned between 0-15 tap delays, each tap giving 50 ps delay, and the initial value on startup is 0.

The delays are tuned at runtime using the DDR2CFG3 register, and extra read cycles can be added using DDR2CFG3, the same way as described for Virtex.

The data sampling clock can also be skewed on Stratix to increase the calibration range. This is done writing the PLL_SKEW bits in the DDR2CFG3 register.

30.2.12 Read data clock calibration on Xilinx Spartan-3

On Spartan3, a clock loop is utilized for sampling of incoming data. The DDR_CLK_FB_OUT port should therefore be connected to a signal path of equal length as the DDR_CLK + DDR_DQS signal path. The other end of the signal path is to be connected to the DDR_CLK_FB port. The fed back clock can then be skewed for alignment with incoming data using the rskew generic. The rskew generic can be set between +/-255 resulting in a linear +/-360 degree change of the clock skew. Bits 29 and 30 in the DDR2CFG3 register can be used for altering the skew at runtime.

30.2.13 Pads

The DDR2SPA core has technology-specific pads inside the core. The external DDR2 signals should therefore be connected directly the top-level ports, without any logic in between.

30.2.14 Endianness

The core is designed for big-endian systems.

30.3 Fault-tolerant operation (preliminary)

30.3.1 Overview

The memory controller can be configured to support bit-error tolerant operation by setting the ft generic (not supported in all versions of GRLIB). In this mode, the DDR data bus is widened and the extra bits are used to store 16 or 32 checkbits corresponding to each 64 bit data word. The variant to

be used can be configured at run-time depending on the connected DDR2 data width and the desired level of fault tolerance.

When writing, the controller generates the check bits and stores them along with the data. When reading, the controller will transparently correct any correctable bit errors and provide the corrected data on the AHB bus. However, the corrected bits are not written back to the memory so external scrubbing is necessary to avoid uncorrectable errors accumulating over time.

An extra corrected error output signal is asserted when a correctable read error occurs, at the same cycle as the corrected data is delivered. This can be connected to an interrupt input or to a memory scrubber. In case of uncorrectable error, this is signaled by giving an AHB error response to the master.

30.3.2 Memory setup

In order to support error-correction, the DDR2 data bus needs to be expanded. The different possible physical configurations are tabulated below. For software, there is no noticeable difference between these configurations.

If the hardware is built for the wider code, it is still possible to leave the upper half of the checkbit data bus unconnected and use it for code B.

*Table 270.*Configurations of FT DDR2 memory banks

Data bits (DDRBITS)	Checkbits (FTBITS)	Interleaving modes supported
64	32	A and B
64	16	B only
32	16	A and B
32	8	B only
16	8	A only

30.3.3 Error-correction properties

The memory controller uses an interleaved error correcting code which works on nibble (4-bit) units of data. The codec can be used in two interleaving modes, mode A and mode B.

In mode A, the basic code has 16 data bits, 8 check bits and can correct one nibble error. This code is interleaved by 4 using the pattern in table 271 to create a code with 64 data bits and 32 check bits.

This code can tolerate one nibble error in each of the A,B,C,D groups shown below. This means that we can correct 100% of single errors in two adjacent nibbles, or in any 8/16-bit wide data bus lane, that would correspond to a physical DDR2 chip. The code can also correct 18/23=78% of all possible random two-nibble errors.

This interleaving pattern was designed to also provide good protection in case of reduced (32/16-bit) DDR bus width with the same data-checkbit relation, so software will see the exact same checkbits on diagnostic reads.

In mode B, the basic code has 32 data bits, 8 check bits and can correct one nibble error. This code is then interleaved by a factor of two to create a code with 64 data bits and 16 check bits.

Note that when configured for a 16-bit wide DDR data bus, code A must be used to get protection from multi-column errors since each data bus nibbles holds four code word nibbles.

*Table 271.*Mode Ax4 interleaving pattern (64-bit data width)

63:60	59:56	55:52	51:48	47:44	43:40	39:36	35:32	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
C	D	A	B	A	B	C	D	B	A	D	C	D	C	B	A
								127:120	119:112	111:104	103:96	95:88	87:80	79:72	71:64
								C_{cb}	D_{cb}	A_{cb}	B_{cb}	C_{cb}	D_{cb}	A_{cb}	B_{cb}

*Table 272.*Mode Bx2 interleaving pattern (64-bit data width)

63:60	59:56	55:52	51:48	47:44	43:40	39:36	35:32	31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0
A	B	A	B	A	B	A	B	B	A	B	A	B	A	B	A

95:88	87:80	79:72	71:64
A_{cb}	B_{cb}	A_{cb}	B_{cb}

30.3.4 Data transfers

The read case behaves the same way as the non-FT counterpart, except a few cycles extra are needed for error detection and correction. There is no extra time penalty in the case data is corrected compared to the error-free case.

Only writes of 64 bit width or higher will translate directly into write cycles to the DDR memory. Other types of write accesses will generate a read-modify-write cycle in order to correctly update the check-bits. In the special case where an uncorrectable error is detected while performing the RMW cycle, the write is aborted and the incorrect checkbits are left unchanged so they will be detected upon the next read.

Only bursts of maximum AHB width is supported, other bursts will be treated as single accesses.

The write FIFO only has room for one write (single or burst).

30.3.5 DDR2 behavior

The behavior over the DDR2 interface is largely unchanged, the same timing parameters and setup applies as for the non-FT case. The checkbit data and data-mask signals follow the same timing as the corresponding signals for regular data.

30.3.6 Configuration

Whether the memory controller is the FT or the non-FT version can be detected by looking at the FTV bit in the DDR2CFG2 register.

Checkbits are always written out to memory when writing even if EDACEN is disabled. Which type of code, A or B, that is used for both read and write is controlled by the CODE field in the DDR2FTCFG register.

Code checking on read is disabled on reset and is enabled by setting the EDACEN bit in the DDR2FTCFG register. Before enabling this, the code to be used should be set in the CODE field and the memory contents should be (re-)initialized.

30.3.7 Diagnostic checkbit access

The checkbits and data can be accessed directly for testing and fault injection. This is done by writing the address of into the DDR2FTDA register. The check-bits and data can then be read and written via the DDR2FTDC and DDR2FTDD register. Note that for checkbits the DDR2FTDA address is 64-bit aligned, while for data it is 32-bit aligned.

After the diagnostic data register has been read, the FT control register bits 31:19 can be read out to see if there were any correctable or uncorrectable errors detected, and where the correctable errors were located. For the 64 databit wide version, there is one bit per byte lane describing whether a correctable error occurred.

30.3.8 Code boundary

The code boundary feature allows you to gradually switch the memory from one interleaving mode to the other and regenerate the checkbits without stopping normal operation. This can be used when recovering from memory faults, as explained further below.

If the boundary address enable (BAEN) control bit is set, the core will look at the address of each access, and use the interleaving mode selected in the CODE field for memory accesses above or equal to the boundary address, and the opposite code for memory accesses below to the boundary address.

If the boundary address update (BAUPD) control bit is also set, the core will shift the boundary upwards whenever the the address directly above the boundary is written to. Since the written data is now below the boundary, it will be written using the opposite code. The write can be done with any size supported by the controller.

30.3.9 Data muxing

When code B is used instead of code A, the upper half of the checkbits are unused. The controller supports switching in this part of the data bus to replace another faulty part of the bus. To do this, one sets the DATAMUX field to a value between 1-4 to replace a quarter of the data bus, or to 5 to replace the active checkbit half.

30.3.10 Memory fault recovery

The above features are designed to, when combined and integrated correctly, make the system capable of dealing with a permanent fault in an external memory chip.

A basic sequence of events is as follows:

1. The system is running correctly with EDAC enabled and the larger code A is used.

2. A memory chip gets a fault and delivers incorrect data. The DDR2 controller keeps delivering error-free data but reports a correctable error on every read access.

3. A logging device (such as the memory scrubber core) registers the high frequency of correctable errors and signals an interrupt.

4. The CPU performs a probe using the DDR2 FT diagnostic registers to confirm that the error is permanent and on which physical lane the error is.

5. After determining that a permanent fault has occurred, the CPU reconfigures the FTDDR2 controller as follows (all configuration register fields changed with a single register write):

   The data muxing control field is set so the top checkbit half replaces the failed part of the data bus.

   The code boundary register is set to the lowest memory address.

   The boundary address enable and boundary address update enable bits are set.

   The mask correctable error bit is set

6. The memory data and checkbits are now regenerated using locked read-write cycles to use the smaller code and replace the broken data with the upper half of the checkbit bus. This can be done in hardware using an IP core, such as the AHB memory scrubber, or by some other means depending on system design.

7. After the whole memory has been regenerated, the CPU disables the code boundary, changes the code selection field to code B, and unsets the mask correctable error bit.

After this sequence, the system is now again fully operational, but running with the smaller code and replacement chip and can again recover from any single-nibble error. Note that during this sequence, it is possible for the system to operate and other masters can both read and write to memory while the regeneration is ongoing.

30.4 Registers

The DDR2SPA core implements between 5 and 12 control registers, depending on the FT generic and target technology. The registers are mapped into AHB I/O address space defined by the AHB BAR1 of the core. Only 32-bit single-accesses to the registers are supported.

Older revisions of the core only have registers DDRCFG1-4, which are aliased on the following addresses. For that reason, check the REG5 bit in DDR2CFG2 before using these bits for backward compatibility.

For backward compatibility, some of the bits in DDR2CFG5 are mirrored in other registers. Writing to these bits will affect the contents of DDR2CFG5 and vice versa.

Table 273. DDR2 controller registers

Address offset - AHB I/O - BAR1	Register
0x00	DDR2 SDRAM control register (DDR2CFG1)
0x04	DDR2 SDRAM configuration register (DDR2CFG2)
0x08	DDR2 SDRAM control register (DDR2CFG3)
0x0C	DDR2 SDRAM control register (DDR2CFG4)
0x10*	DDR2 SDRAM control register (DDR2CFG5)
0x14*	Reserved
0x18	DDR2 Technology specific register (DDR2TSR1)
0x1C*	DDR2 Technology specific register (DDR2TSR2)
0x20	DDR2 FT Configuration Register (FT only) (DDR2FTCFG)
0x24	DDR2 FT Diagnostic Address register (FT only) (DDR2FTDA)
0x28	DDR2 FT Diagnostic Checkbit register (FT only) (DDR2FTDC)
0x2C	DDR2 FT Diagnostic Data register (FT only) (DDR2FTDD)
0x30	DDR2 FT Code Boundary Register (FT only) (DDR2FTBND)

* Older DDR2SPA versions contain aliases of DDR2CFG1-4 at these addresses. Therefore, check bit 15 of DDR2CFG2 before using these registers.

30.4.1 DDR2 SDRAM Configuration Register 1

Table 274. 0x00 - DDR2CFG1 - DDR2 SRAM control register 1

31	30	29 28	27	26	25 23	22 21	20 18	17	16	15	14 0
Refresh	OCD	EMR	bank size 3	(TRCD)	SDRAM bank size2:0	SDRAM col. size	SDRAM command	PR	IN	CE	SDRAM refresh load value
0	0	0	0	0	0	0	0	0	0	0	0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31 SDRAM refresh. If set, the SDRAM refresh will be enabled.

30 OCD operation

29: 28 Selects Extended mode register (1,2,3)

27 SDRAM banks size bit 3. By enabling this bit the memory size can be set to “1000” = 2048 Mbyte and “1001” = 4096 Mbyte. See the section on big-memory support.

26 Lowest bit of TRCD field in DDR2CFG, for backward compatibility

25: 23 SDRAM banks size. Defines the decoded memory size for each SDRAM chip select: “000”= 8 Mbyte, “001”= 16 Mbyte, “010”= 32 Mbyte.... “111”= 1024 Mbyte.

22: 21 SDRAM column size. “00”=512, “01”=1024, “10”=2048, “11”=4096

20: 18 SDRAM command. Writing a non-zero value will generate an SDRAM command: “010”=PRE-CHARGE, “100”=AUTO-REFRESH, “110”=LOAD-COMMAND-REGISTER, “111”=LOAD-EXTENDED-COMMAND-REGISTER. The field is reset after command has been executed.

17 PLL Reset. This bit is used to set the PLL RESET bit during LOAD-CONFIG-REG commands.

16 Initialize (IN). Set to ‘1’ to perform power-on DDR RAM initialisation. Is automatically cleared when initialisation is completed.

15 Clock enable (CE). This value is driven on the CKE inputs of the DDR RAM. Should be set to ‘1’ for correct operation.

14: 0 The period between each AUTO-REFRESH command - Calculated as follows: tREFRESH = ((reload value) + 1) / DDRCLOCK

30.4.2 DDR2 SDRAM Configuration Register 2

Table 275. 0x04 - DDR2CFG2 - DDR2 SDRAM configuration register 2

31 26	25 18	17	16	15	14 12	11 0
RESERVED	PHY Tech	BIG	FTV	REG5	Data width	DDR Clock frequency
0	0	0	0	0	0	0
r	r	r	r	r	r	r

31: 26 Reserved

25: 18 PHY technology identifier, value 0 is for generic/unknown

17 Big memory support, if ‘1’ then memory can be set between 32 Mbyte and 4 Gbyte, if ‘0’ then memory size can be set between 8 Mbyte and 1 Gbyte.

16 Reads ‘1’ if the controller is fault-tolerant version and EDAC registers exist.

15 Reads ‘1’ if DDR2CFG5 register exists.

14: 12 SDRAM data width: “001” = 16 bits, “010” = 32 bits, “011” = 64 bits.

11: 0 Frequency of the (external) DDR clock.

30.4.3 DDR2 SDRAM Configuration Register 3

*Table 276.*0x08 - DDR2CFG3 - DDR2 SDRAM configuration register 3

31	30 29	28	27 23	22 18	17 16	15 8	7 0
	PLL	(TRP)	tWR	(TRFC)	RD	inc/dec delay	Update delay
		0	0	*	*	0	0
		rw	rw	rw	rw	rw	rw

31	Reset byte delay
30: 29	PLL_SKEW Bit 29: Update clock phase Bit 30: 1 = Inc / 0 = Dec clock phase
28	Lowest bit of DDR2CFG4 TRP field for backward compatibility
27: 23	SDRAM write recovery time. tWR will be equal to field value - 2DDR clock cycles
22: 18	Lower 5 bits of DDR2CFG4 TRFC field for backward compatibility.
17: 16	Number of added read delay cycles, default = 1
15: 8	Set to ‘1’ to increment byte delay, set to ‘0’ to decrement delay
7: 0	Set to ‘1’ to update byte delay

30.4.4 DDR2 SDRAM Configuration Register 4

*Table 277.*0x0C - DDR2CFG4 - DDR2 SDRAM configuration register 4

31 28	27 24	23 22	21	20 14	13	12 11	10 9	8	7 0
inc/dec CB delay	Update CB delay	RDH	REG	RESERVED	TRTP	RES	TCL	B8	DQS gating offset
0	0	0	0	0	*	0	*	*	0
rw	rw	rw	rw	r	rw	r	rw	rw	rw

31: 28	Set to ‘1’ to increment checkbits byte delay, set to ‘0’ to decrement delay
27: 24	Set to ‘1’ to update checkbits byte delay
23: 22	Read delay high bits, setting this field to N adds 4 x N read delay cycles
21	Registered memory (1 cycle extra latency on control signals)
20: 14	Reserved
13	SDRAM read-to-precharge timing, tRTP will be equal to field value + 2 DDR-clock cycles.
12: 11	Reserved
10: 9	SDRAM CAS latency timing. CL will be equal to field value + 3 DDR-clock cycles. Note: You must reprogram the memory’s MR register after changing this value
8	Enables address generation for DDR2 chips with eight banks 1=addressess generation for eight banks 0=address generation for four banks
7: 0	Number of half clock cycles for which the DQS input signal will be active after a read command is given. After this time the DQS signal will be gated off to prevent latching of faulty data. Only valid if the dqsgating generic is enabled.

30.4.5 DDR2 SDRAM Configuration Register 5

Table 278. 0x10 - DDR2CFG5 - DDR2 SDRAM configuration register 5

31	30 28	27 26	25 18	17 16	15	14 11	10 8	7 5	4 0
R	TRP	RES	TRFC	ODT	DS	RESERVED	TRCD	RESERVED	TRAS
0	*	0	0	0	0	0	*	0	0
r	rw	r	rw	rw	rw	r	rw	r	rw

31	Reserved
30: 28	SDRAM tRP timing. tRP will be equal to 2 + field value DDR-clock cycles
27: 26	Reserved
25: 18	SDRAM tRFC timing. tRFC will be equal to 3 + field-value DDR-clock cycles.
17: 16	SDRAM-side on-die termination setting (0=disabled, 1-3=75/150/50 ohm) Note: You must reprogram the EMR1 register after changing this value.
15	SDRAM-side output drive strength control (0=full strength, 1=half strength) Note: You must reprogram the EMR1 register after changing this value
14: 11	Reserved
10: 8	SDRAM RAS-to-CAS delay (TRCD). tRCD will be equal to field value + 2 DDR-clock cycles
7: 5	Reserved
4: 0	SDRAM RAS to precharge timing. TRAS will be equal to 2+ field value DDR-clock cycles

30.4.6 DDR2 FT Configuration Register

Table 279. 0x20 - DDR2FTCFG - DDR2 FT configuration register

31 20	19	18 16	15 8	7 5	4	3	2	1	0
Diag data read error location	DDERR	DM	RESERVED	DATAMUX	CEM	BAUPD	BAEN	CODE	EDEN
0	0	0	0	0	0	0	0	0	0
r	r	rw	r	rw	rw	rw	rw	rw	rw

31: 20	Bit field describing location of corrected errors for last diagnostic data read (read-only) One bit per byte lane in 64+32-bit configuration
19	Set high if last diagnostic data read contained an uncorrectable error (read-only)
18: 16	Data width, read-only field. 001=16+8, 010=32+16, 011=64+32 bits
15: 8	Reserved
7: 5	Data mux control, setting this nonzero switches in the upper checkbit half with another data lane. For 64-bit interface 000 = no switching 001 = Data bits 15:0, 010 = Data bits 31:16, 011: Data bits 47:32, 100: Data bits 63:48, 101 = Checkbits 79:64, 110,111 = Undefined
4	If set high, the correctable error signal is masked out.
3	Enable automatic boundary shifting on write
2	Enable the code boundary
1	Code selection, 0=Code A (64+32/32+16/16+8), 1=Code B (64+16/32+8)
0	EDAC Enable

30.4.7 DDR2 FT Diagnostic Address

*Table 280.*0x24 - DDR2FTDA - DDR2 FT Diagnostic Address

31	2	1	0
MEMORY ADDRESS		RESERVED	
0		0	
rw		r	

31: 3 Address to memory location for checkbit read/write, 64/32-bit aligned for checkbits/data

1: 0 Reserved (address bits always 0 due to alignment)

30.4.8 DDR2 FT Diagnostic Checkbits

Table 281. 0x28 - DDR2FTDC - DDR2 FT Diagnostic Checkbits

31	24	23	16	15	8	7	0
CHECKBITS D		CHECKBITS C		CHECKBITS B		CHECKBITS A	
*		*		*		0	
rw		rw		rw		rw	

31: 24 Checkbits for part D of 64-bit data word (undefined for code B)

23: 16 Checkbits for part C of 64-bit data word (undefined for code B)

15: 8 Checkbits for part B of 64-bit data word

7: 0 Checkbits for part A of 64-it data word.

30.4.9 DDR2 FT Diagnostic Data

Table 282. 0x2C - DDR2FTDD - DDR2 FT Diagnostic Data

31	0
DATA BITS	
*	
r	

31: 0 Uncorrected data bits for 32-bit address set in DDR2FTDA

30.4.10 DDR2 FT Boundary Address Register

Table 283. 0x30 - DDR2FTBND - DDR2 FT Boundary Address Register

31	3	2	0
CHECKBIT CODE BOUNDARY ADDRESS		R	
0		0	
rw		r	

31: 3 Code boundary address, 64-bit aligned

2: 0 Zero due to alignment

30.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x02E. The revision described in this document is revision 1. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

30.6 Implementation

30.6.1 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

30.7 Configuration options

Table 284 shows the configuration options of the core (VHDL generics).

*Table 284.*Configuration options

Generic	Function	Allowed range	Default
fabtech	PHY technology selection	virtex4, virtex5, stratix3	virtex4
memtech	Technology selection for DDR FIFOs	inferred, virtex2, virtex4, spartan3e, altera	inferred
hindex	AHB slave index	0 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR0 defining SDRAM area. Default is 0xF0000000 - 0xFFFFFFFF.	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0 defining SDRAM area.	0 - 16#FFF#	16#F00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space where DDR control register is mapped.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR1 defining I/O address space	0 - 16#FFF#	16#FFF#
ddrbits	Data bus width of external DDR memory	16, 32, 64	16
MHz	DDR clock input frequency in MHz.	10 - 200	100
clkmul, clkdiv	The DDR input clock is multiplied with the clkmul generic and divided with clkdiv to create the final DDR clock	2 - 32	2
rstdel	Clock reset delay in micro-seconds.	1 - 1023	200
col	Default number of column address bits	9 - 12	9
Mbyte	Default memory chip select bank size in Mbyte	8 - 1024	16
pwron	Enable SDRAM at power-on initialization	0 - 1	0
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
ahbfreq	Frequency in MHz of the AHB clock domain	1 - 1023	50
readdly	Additional read latency cycles (used to increase calibration range)	0-3	1
TRFC	Reset value for the tRFC timing parameter in ns.	75-155	130
ddelayb0*	Input data delay for bit[7:0]	0-63	0
ddelayb1*	Input data delay for bit[15:8]	0-63	0
ddelayb2*	Input data delay for bit[23:16]	0-63	0
ddelayb3*	Input data delay for bit[31:24]	0-63	0
ddelayb4*	Input data delay for bit[39:32]	0-63	0
ddelayb5*	Input data delay for bit[47:40]	0-63	0
ddelayb6*	Input data delay for bit[55:48]	0-63	0
ddelayb7*	Input data delay for bit[63:56]	0-63	0
cbdelayb0*	Input data delay for checkbit[7:0]	0-63	0
cbdelayb1*	Input data delay for checkbit[15:8]	0-63	0
cbdelayb2*	Input data delay for checkbit[23:16]	0-63	0
cbdelayb3*	Input data delay for checkbit[31:24]	0-63	0

*Table 284.*Configuration options

Generic	Function	Allowed range	Default
numidelctrl*	Number of IDELAYCTRL the core will instantiate	-	4
norefclk*	Set to 1 if no 200 MHz reference clock is connected to clkref200 input.	0-1	0
odten	Enable odt: 0 = Disabled, 1 = 75Ohm, 2 =150Ohm, 3 = 50Ohm	0-3	0
rskew**	Set the phase relationship between the DDR controller clock and the input data sampling clock. Sets the phase in ps.	0 - 9999	0
octen**	Enable on chip termination: 1 = enabled, 0 = disabled	0 - 1	0
dqsgating***	Enable gating of DQS signals when doing reads. 1 = enable, 0 = disable	0 - 1	0
nosync	Disable insertion of synchronization registers between AHB clock domain and DDR clock domain. This can be done if the AHB clock's rising edges always are in phase with a rising edge on the DDR clock. If this generic is set to 1 the clkmul and clkdiv generics should be equal. Otherwise the DDR controller may scale the incoming clock and loose the clocks' edge alignment in the process.	0 - 1	0
eightbanks	Enables address generation for DDR2 chips with eight banks. The DDR_BA is extended to 3 bits if set to 1.	0 - 1	0
dqsse	Single-ended DQS. The value of this generic is written to bit 10 in the memory's Extended Mode register. If this bit is 1 DQS is used in a single-ended mode. Currently this bit should only, and must be, set to 1 when the Stratix2 DDR2 PHY is used. This is the only PHY that supports single ended DQS without modification.	0 - 1	0
burstlen	DDR access burst length in 32-bit words	8,16,32,..,256	8
ahbbits	AHB bus width	32,64,128,256	AHBDW
ft	Enable fault-tolerant version	0 - 1	0
ftbits	Extra DDR data bits used for checkbits	0,8,16,32	0
bigmem	Big memory support, changes the range of supported total memory bank sizes from 8MB-1GB to 32MB-4GB	0 - 1	0
raspipe	Enables an extra pipeline stage in the address decoding to improve timing at the cost of one DDR-cycle latency	0 - 1	0
* only available in Virtex4/5 implementation. ** only available in Altera and Spartan3 implementations. *** only available on Nextreme/eASIC implementations			

30.8 Implementation

30.8.1 Technology mapping

The core has two technology mapping VHDL generics: *memtech* and *fabtech*. The VHDL generic *memtech* controls the technology used for memory cell implementation. The VHDL generic *fabtech* controls the technology used in the PHY implementation. See the GRLIB Users's Manual for available settings.

30.8.2 FPGA support

Complete PHY:s for a number of FPGA technologies are included in the distribution, see table below. Unless otherwise noted these have been only functionally tested on evaluation board in lab environment and detailed timing analysis has not been performed. Note also that some of the FPGA phy:s use

simplified sampling approaches which may require the memory timing to be better than the JEDEC standard specifies.

Scripts for post-layout static timing analysis are not included. Because these PHY:s are based on dedicated hard macros with fixed placement in the FPGA:s pad structure, just a minimal set of constraints are normally neccessary for synthesis purposes.

Some PHY:s support instantiation without built-in pads, to separate the pads from the PHY the internal ddr2spax entity and the phy must be instantiated manually.

*Table 285.*FPGA DDR2 PHYs included in GRLIB

Technology	fabtech	Read clock method	Built-in pads
Stratix 2	stratix2	Tech intrinsics (DQS based)	Yes
Stratix 3	stratix3	Tunable static shift	Yes
Spartan 3	spartan3	Clock feedback loop + static shift	Yes
Virtex4,5,6	virtex4, virtex5, virtex6	Fixed clock, DQ shifted using IDELAY	Yes or No
Spartan6	spartan6	Fixed clock, DQ shifted using IDELAY	Yes or No

30.8.3 RAM usage

The FIFOs in the core are implemented with the *syncram_2p* (with separate clock for each port) component found in the technology mapping library (TECHMAP). The number of RAMs used for the FIFO implementation depends om the DDR data width, set by the *ddrbits* VHDL generic, and the AHB bus width in the system.

The RAM block usage is tabulated below for the default burst length of 8 words. If the burst length is doubled, the depths for all the RAMs double as well but the count and width remain the same.

*Table 286.*Block-RAM usage for default burst length

DDR width	AHB width	Write FIFO block-RAM usage			Read-FIFO block-RAM usage			Total RAM count
		Count	Depth	Width	Count	Depth	Width	
16	32	1	16	32	1	8	32	2
16	64	2	8	32	2	4	32	4
16	128	4	4	32	4	2	32	8
16	256	8	2	32	8	1	32	16
32	32	2	8	32	1	4	64	3
32	64	2	8	32	1	4	64	3
32	128	4	4	32	2	2	64	6
32	256	8	2	32	4	1	64	12
64	32	4	4	32	1	2	128	5
64	64	4	4	32	1	2	128	5
64	128	4	4	32	1	2	128	5
64	256	8	2	32	2	1	128	10

30.8.4 Xilinx Virtex-specific issues

The Xilinx tools require one IDELAYCTRL macro to be instantiated in every region where the IDELAY feature is used. Since the DDR2 PHY uses the IDELAY on every data (DQ) pin, this affects the DDR2 core. For this purpose, the core has a numidelctrl generic, controlling how many IDELAYCTRL's get instantiated in the PHY.

The tools allow for two ways to do this instantiation:

- Instantiate the same number of IDELAYCTRL as the number of clock regions containing DQ pins and place the instances manually using UCF LOC constraints.
- Instantiate just one IDELAYCTRL, which the ISE tools will then replicate over all regions.

The second solution is the simplest, since you just need to set the numidelctrl to 1 and no extra constraints are needed. However, this approach will not work if IDELAY is used anywhere else in the FPGA design.

For more information on IDELAYCTRL, see Xilinx Virtex4/5 User's Guide.

30.8.5 Design tools

To run the design in Altera Quartus 7.2 you have to uncomment the lines in the .qsf file that assigns the MEMORY_INTERFACE_DATA_PIN_GROUP for the DDR2 interface. These group assignments result in error when Altera Quartus 8.0 is used.

30.9 Signal descriptions

Table 287 shows the interface signals of the core (VHDL ports).

*Table 287.*Signal descriptions

Signal name	Type	Function	Active
RST_DDR	Input	Reset input for the DDR PHY	Low
RST_AHB	Input	Reset input for AHB clock domain	Low
CLK_DDR	Input	DDR input Clock	-
CLK_AHB	Input	AHB clock	-
CLKREF200	Input	200 MHz reference clock	-
LOCK	Output	DDR clock generator locked	High
CLKDDRO		Internal DDR clock output after clock multiplication	
CLKDDRI		Clock input for the internal DDR clock domain. Must be connected to CLKDDRO.	
AHBSI	Input	AHB slave input signals	-
AHBSO	Output	AHB slave output signals	-
DDR_CLK[2:0]	Output	DDR memory clocks (positive)	High
DDR_CLKB[2:0]	Output	DDR memory clocks (negative)	Low
DDR_CLK_FB_OUT	Output	DDR data synchronization clock, connect this to a signal path with equal length of the DDR_CLK trace + DDR_DQS trace	-
DDR_CLK_FB	Input	DDR data synchronization clock, connect this to the other end of the signal path connected to DDR_-CLK_FB_OUT	-
DDR_CKE[1:0]	Output	DDR memory clock enable	High
DDR_CSB[1:0]	Output	DDR memory chip select	Low
DDR_WEB	Output	DDR memory write enable	Low
DDR_RASB	Output	DDR memory row address strobe	Low
DDR_CASB	Output	DDR memory column address strobe	Low
DDR_DM[(DDRBITS+FTBITS)/8-1:0]	Output	DDR memory data mask	Low
DDR_DQS[(DDRBITS+FTBITS)/8-1:0]	Bidir	DDR memory data strobe	Low
DDR_DQSN[(DDRBITS+FTBITS)/8-1:0]	Bidir	DDR memory data strobe (inverted)	High
DDR_AD[13:0]	Output	DDR memory address bus	Low
DDR_BA[2 or 1:0] 3)	Output	DDR memory bank address	Low
DDR_DQ[DDRBITS+FTBITS-1:0]	BiDir	DDR memory data bus	-
DDR_ODT[1:0]	Output	DDR memory odt	Low

1) see GRLIB IP Library User's Manual
2) Polarity selected with the oepol generic
3) DDR_BA[2:0] if the eightbanks generic is set to 1 else DDR_BA[1:0]
4) Only used on Virtex4/5
5) Only used on Spartan3

30.10 Library dependencies

Table 288 shows libraries used when instantiating the core (VHDL libraries).

*Table 288.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

30.11 Component declaration

```
component ddr2spa
  generic (
    fabtech : integer := 0;
    memtech : integer := 0;
    hindex  : integer := 0;
    haddr   : integer := 0;
    hmask   : integer := 16#f00#;
    ioaddr  : integer := 16#000#;
    iomask  : integer := 16#fff#;
    MHz     : integer := 100;
    clkmul  : integer := 2;
    clkdiv  : integer := 2;
    col     : integer := 9;
    Mbyte   : integer := 16;
    rstdel  : integer := 200;
    pwron   : integer := 0;
    oepol   : integer := 0;
    ddrbits : integer := 16;
    ahbfreq : integer := 50;
    readdly : integer := 1;
    ddelayb0: integer := 0;
    ddelayb1: integer := 0;
    ddelayb2: integer := 0;
    ddelayb3: integer := 0;
    ddelayb4: integer := 0;
    ddelayb5: integer := 0;
    ddelayb6: integer := 0;
    ddelayb7: integer := 0
  );
  port (
    rst_ddr    : in  std_ulogic;
    rst_ahb    : in  std_ulogic;
    clk_ddr    : in  std_ulogic;
    clk_ahb    : in  std_ulogic;
    clkref200 : in std_ulogic;
    lock       : out std_ulogic;-- DCM locked
    clkddro    : out std_ulogic;-- DCM locked
    clkddri    : in  std_ulogic;
    ahbsi      : in  ahb_slv_in_type;
    ahbso      : out ahb_slv_out_type;
    ddr_clk    : out std_logic_vector(2 downto 0);
    ddr_clkb   : out std_logic_vector(2 downto 0);
    ddr_cke    : out std_logic_vector(1 downto 0);
    ddr_csb    : out std_logic_vector(1 downto 0);
    ddr_web    : out std_ulogic;                       -- ddr write enable
    ddr_rasb   : out std_ulogic;                       -- ddr ras
    ddr_casb   : out std_ulogic;                       -- ddr cas
    ddr_dm     : out std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dm
    ddr_dqs    : inout std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dqs
    ddr_dqsn   : inout std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dqs
    ddr_ad     : out std_logic_vector (13 downto 0);   -- ddr address
    ddr_ba     : out std_logic_vector (1 downto 0);    -- ddr bank address
    ddr_dq     : inout  std_logic_vector (ddrbits-1 downto 0); -- ddr data
    ddr_odt    : out std_logic_vector(1 downto 0) -- odt
  );
  end component;
```

30.12 Instantiation

This example shows how the core can be instantiated.

The DDR SDRAM controller decodes SDRAM area at 0x40000000 - 0x7FFFFFFF. The DDR2 SDRAM registers are mapped into AHB I/O space on address (AHB I/O base address + 0x100).

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;

entity ddr_Interface is
  port (
    ddr_clk  : out std_logic_vector(2 downto 0);
    ddr_clkb : out std_logic_vector(2 downto 0);
    ddr_cke  : out std_logic_vector(1 downto 0);
    ddr_csb  : out std_logic_vector(1 downto 0);
    ddr_web  : out std_ulogic;                        -- ddr write enable
    ddr_rasb : out std_ulogic;                        -- ddr ras
    ddr_casb : out std_ulogic;                        -- ddr cas
    ddr_dm   : out std_logic_vector (7 downto 0);     -- ddr dm
    ddr_dqs  : inout std_logic_vector (7 downto 0);     -- ddr dqs
    ddr_dqsn : inout std_logic_vector (7 downto 0);     -- ddr dqsn
    ddr_ad   : out std_logic_vector (13 downto 0);    -- ddr address
    ddr_ba   : out std_logic_vector (1 downto 0);     -- ddr bank address
    ddr_dq   : inout std_logic_vector (63 downto 0); -- ddr data
    ddr_odt  : out std_logic_vector (1 downto 0) -- ddr odt
);
end;

architecture rtl of mctrl_ex is

  -- AMBA bus
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
signal clkml, lock, clk_200,
signal clk_200 : std_ulogic; -- 200 MHz reference clock
signal ddrclkin, ahbclk : std_ulogic; -- DDR input clock and AMBA sys clock
signal rstn  : std_ulogic; -- Synchronous reset signal
signal reset : std_ulogic; -- Asynchronous reset signal


begin

-- DDR controller


ddrc : ddr2spa generic map ( fabtech => virtex4, ddrbits => 64, memtech => memtech,
hindex => 4, haddr => 16#400#, hmask => 16#F00#, ioaddr => 1,
pwron => 1, MHz => 100, col => 9, Mbyte => 32, ahbfreq => 50, ddrbits => 64,
readdly => 1, ddelayb0 => 0, ddelayb1 => 0, ddelayb2 => 0, ddelayb3 => 0,
ddelayb4 => 0, ddelayb5 => 0, ddelayb6 => 0, ddelayb7 => 0)
port map (
reset, rstn, ddrclkin, ahbclk, clk_200, lock, clkml, clkml,  ahbsi, ahbso(4),
ddr_clk, ddr_clkb,
ddr_cke, ddr_csb, ddr_web, ddr_rasb, ddr_casb,
ddr_dm, ddr_dqs, ddr_adl, ddr_ba, ddr_dq, ddr_odt);
```

31 DIV32 - Signed/unsigned 64/32 divider module

31.1 Overview

The divider module performs signed/unsigned 64-bit by 32-bit division. It implements the radix-2 non-restoring iterative division algorithm. The division operation takes 36 clock cycles. The divider leaves no remainder. The result is rounded towards zero. Negative result, zero result and overflow (according to the overflow detection method B of SPARC V8 Architecture manual) are detected.

31.2 Operation

The division is started when '1' is samples on DIVI.START on positive clock edge. Operands are latched externally and provided on inputs DIVI.Y, DIVI.OP1 and DIVI.OP2 during the whole operation. The result appears on the outputs during the clock cycle following the clock cycle after the DIVO.READY was asserted. Asserting the HOLD input at any time will freeze the operation, until HOLDN is de-asserted.

31.3 Implementation

31.3.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers if the GRLIB config package setting *grlib_async_reset_enable* is set.

31.4 Configurations options

Core has only one VHDL generic, scantest, that should be set to 1 if GRLIB has been configured to use asynchronous reset.

31.5 Signal descriptions

Table 289 shows the interface signals of the core (VHDL ports).

*Table 289.*Signal declarations

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
HOLDN	N/A	Input	Hold	Low
DIVI	Y[32:0]	Input	Dividend - MSB part Y[32] - Sign bit Y[31:0] - Dividend MSB part in 2's complement format	High
	OP1[32:0]		Dividend - LSB part OP1[32] - Sign bit OP1[31:0] - Dividend LSB part in 2's complement format	High
	FLUSH		Flush current operation	High
	SIGNED		Signed division	High
	START		Start division	High
DIVO	READY	Output	The result is available one clock after the ready signal is asserted.	High
	NREADY		The result is available three clock cycles, assuming hold=HIGH, after the nready signal is asserted.	High
	ICC[3:0]		Condition codes ICC[3] - Negative result ICC[2] - Zero result ICC[1] - Overflow ICC[0] - Not used. Always '0'.	High
	RESULT[31:0]		Result	High
TESTEN	N/A	Input	Test enable (only used together with async. reset)	High
TESTRST	N/A	Input	Test reset (only used together with async. reset)	Low

31.6 Library dependencies

Table 290 shows libraries used when instantiating the core (VHDL libraries).

*Table 290.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	ARITH	Signals, component	Divider module signals, component declaration

31.7 Component declaration

The core has the following component declaration.

```
component div32
port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    holdn   : in  std_ulogic;
```

```
    divi    : in  div32_in_type;
    divo    : out div32_out_type
);
end component;
```

31.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use gaisler.arith.all;

.
.
.

signal divi  : div32_in_type;
signal divo  : div32_out_type;

begin

div0 : div32 port map (rst, clk, holdn, divi, divo);

end;
```

32 DSU3 - LEON3 Hardware Debug Support Unit

32.1 Overview

To simplify debugging on target hardware, the LEON3 processor implements a debug mode during which the pipeline is idle and the processor is controlled through a special debug interface. The LEON3 Debug Support Unit (DSU) is used to control the processor during debug mode. The DSU acts as an AHB slave and can be accessed by any AHB master. An external debug host can therefore access the DSU through several different interfaces. Such an interface can be a serial UART (RS232), JTAG, PCI, USB or Ethernet. The DSU supports multi-processor systems and can handle up to 16 processors.

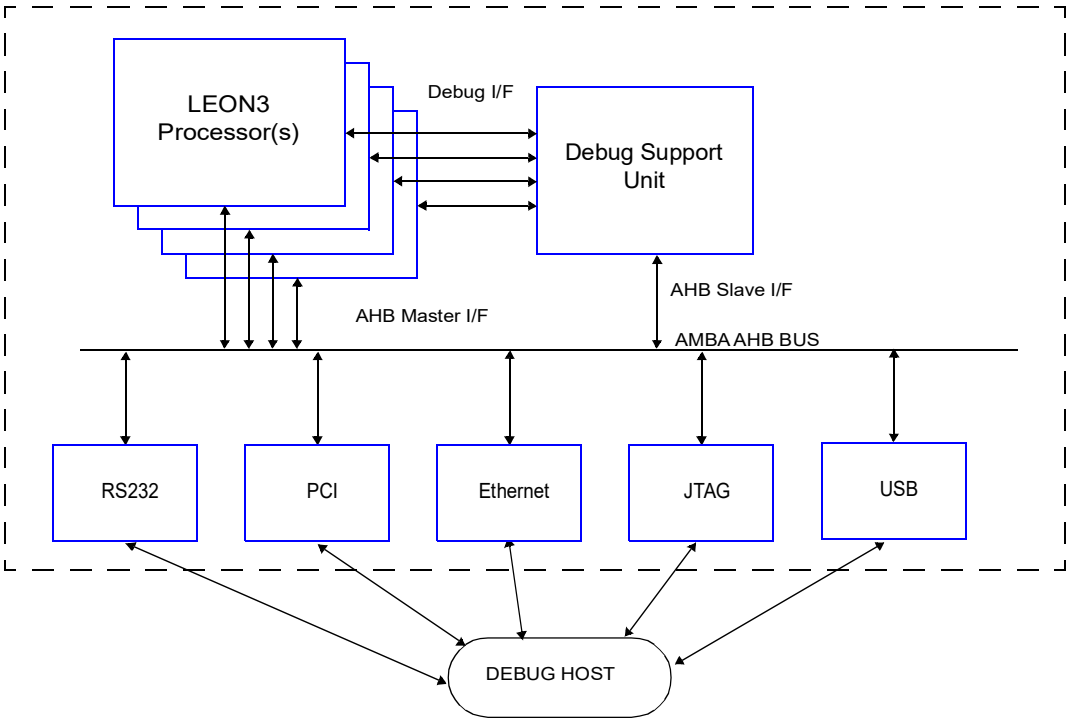


Figure 67. LEON3/DSU Connection

32.2 Operation

Through the DSU AHB slave interface, any AHB master can access the processor registers and the contents of the instruction trace buffer. The DSU control registers can be accessed at any time, while the processor registers and caches can only be accessed when the processor has entered debug mode. In debug mode, the processor pipeline is held and the processor state can be accessed by the DSU. Entering the debug mode can occur on the following events:

- executing a breakpoint instruction (ta 1)
- integer unit hardware breakpoint/watchpoint hit (trap 0xb)
- rising edge of the external break signal (DSUBRE)
- setting the break-now (BN) bit in the DSU control register
- a trap that would cause the processor to enter error mode
- occurrence of any, or a selection of traps as defined in the DSU control register
- after a single-step operation
- one of the processors in a multiprocessor system has entered the debug mode
- DSU AHB breakpoint or watchpoint hit

The debug mode can only be entered when the debug support unit is enabled through an external signal (DSUEN). For DSU break (DSUBRE), and the break-now BN bit, to have effect the Break-on-IU-watchpoint (BW) bit must be set in the DSU control register. This bit is set when DSUBRE is active after reset and should also be set by debug monitor software when initializing the DSU. When the debug mode is entered, the following actions are taken:

- PC and nPC are saved in temporary registers (accessible by the debug unit)
- an output signal (DSUACT) is asserted to indicate the debug state
- the timer unit is (optionally) stopped to freeze the LEON timers and watchdog

The instruction that caused the processor to enter debug mode is not executed, and the processor state is kept unmodified. Execution is resumed by clearing the BN bit in the DSU control register or by de-asserting DSUEN. The timer unit will be re-enabled and execution will continue from the saved PC and nPC. Debug mode can also be entered after the processor has entered error mode, for instance when an application has terminated and halted the processor. The error mode can be reset and the processor restarted at any address.

When a processor is in the debug mode, an access to ASI diagnostic area is forwarded to the IU which performs access with ASI equal to value in the DSU ASI register and address consisting of 20 LSB bits of the original address.

32.3 AHB trace buffer

The AHB trace buffer consists of a circular buffer that stores AHB data transfers, the monitored AHB bus is either the same bus as the DSU AHB slave interface is connected to, or a completely separate bus. The address, data and various control signals of the AHB bus are stored and can be read out for later analysis. The trace buffer is 128, 160 or 224 bits wide, depending on the AHB bus width. Note that for systems with a 256 bit wide AHB bus, the DSU3 can be used but will only save the lower 128 bits of the AHB data bus. The way information stored is indicated in the table below:

*Table 291.*AHB Trace buffer data allocation

Bits	Name	Definition
223:160	Load/Store data	AHB HRDATA/HWDATA(127:64)
159:129	Load/Store data	AHB HRDATA/HWDATA(63:32)
127	AHB breakpoint hit	Set to ‘1’ if a DSU AHB breakpoint hit occurred.
126	-	Not used
125:96	Time tag	DSU time tag counter
95:80	-	Not used
79	Hwrite	AHB HWRITE
78:77	Htrans	AHB HTRANS
76:74	Hsize	AHB HSIZE
73:71	Hburst	AHB HBURST
70:67	Hmaster	AHB HMASTER
66	Hmastlock	AHB HMASTLOCK
65:64	Hresp	AHB HRESP
63:32	Load/Store data	AHB HRDATA/HWDATA(31:0)
31:0	Load/Store address	AHB HADDR

In addition to the AHB signals, the DSU time tag counter is also stored in the trace.

The trace buffer is enabled by setting the enable bit (EN) in the trace control register. Each AHB transfer is then stored in the buffer in a circular manner. The address to which the next transfer is writ-

ten is held in the trace buffer index register, and is automatically incremented after each transfer. Tracing is stopped when the EN bit is reset, or when a AHB breakpoint is hit. Tracing is temporarily suspended when the processor enters debug mode, unless the trace force bit (TF) in the trace control register is set. If the trace force bit is set, the trace buffer is activated as long as the enable bit is set. The force bit is reset if an AHB breakpoint is hit and can also be cleared by software. Note that neither the trace buffer memory nor the breakpoint registers (see below) can be read/written by software when the trace buffer is enabled.

The DSU has an internal time tag counter and this counter is frozen when the processor enters debug mode. When AHB tracing is performed in debug mode (using the trace force bit) it may be desirable to also enable the time tag counter. This can be done using the timer enable bit (TE). Note that the time tag is also used for the instruction trace buffer and the timer enable bit should only be set when using the DSU as an AHB trace buffer only, and not when performing profiling or software debugging. The timer enable bit is reset on the same events as the trace force bit.

32.3.1 AHB trace buffer filters

The DSU can be implemented with filters that can be applied to the AHB trace buffer, breakpoints and watchpoints. If implemented, these filters are controlled via the AHB trace buffer filter control and AHB trace buffer filter mask registers. The fields in these registers allows masking access characteristics such as master, slave, read, write and address range so that accesses that correspond to the specified mask are not written into the trace buffer. Address range masking is done using the second AHB breakpoint register set. The values of the LD and ST fields of this register has no effect on filtering.

32.3.2 AHB statistics

The DSU can be implemented to generate statistics from the traced AHB bus. When statistics collection is enabled the DSU will assert outputs that are suitable to connect to a LEON3 statistics unit (L3STAT). The statistical outputs can be filtered by the AHB trace buffer filters, this is controlled by the Performance counter Filter bit (PF) in the AHB trace buffer filter control register. The DSU can collect data for the events listed in table 292 below.

*Table 292.*AHB events

Event	Description	Note
idle	HTRANS=IDLE	Active when HTRANS IDLE is driven on the AHB slave inputs and slave has asserted HREADY.
busy	HTRANS=BUSY	Active when HTRANS BUSY is driven on the AHB slave inputs and slave has asserted HREADY.
nseq	HTRANS=NONSEQ	Active when HTRANS NONSEQ is driven on the AHB slave inputs and slave has asserted HREADY.
seq	HTRANS=SEQ	Active when HTRANS SEQUENTIAL is driven on the AHB slave inputs and slave has asserted HREADY.
read	Read access	Active when HTRANS is SEQUENTIAL or NON-SEQUENTIAL, slave has asserted HREADY and the HWRITE input is low.
write	Write access	Active when HTRANS is SEQUENTIAL or NON-SEQUENTIAL, slave has asserted HREADY and the HWRITE input is high.
hsize[5:0]	Transfer size	Active when HTRANS is SEQUENTIAL or NON-SEQUENTIAL, slave has asserted HREADY and HSIZE is BYTE (hsize[0]), HWORD (HSIZE[1]), WORD (hsize[2]), DWORD (hsize[3]), 4WORD hsize[4], or 8WORD (hsize[5]).
ws	Wait state	Active when HREADY input to AHB slaves is low and AMBA response is OKAY.
retry	RETRY response	Active when master receives RETRY response
split	SPLIT response	Active when master receives SPLIT response

*Table 292.*AHB events

Event	Description	Note
spdel	SPLIT delay	Active during the time a master waits to be granted access to the bus after reception of a SPLIT response. The core will only keep track of one master at a time. This means that when a SPLIT response is detected, the core will save the master index. This event will then be active until the same master is re-allowed into bus arbitration and is granted access to the bus. This also means that the delay measured will include the time for re-arbitration, delays from other ongoing transfers and delays resulting from other masters being granted access to the bus before the SPLIT:ed master is granted again after receiving SPLIT complete. If another master receives a SPLIT response while this event is active, the SPLIT delay for the second master will not be measured.
locked	Locked access	Active while the HMASTLOCK signal is asserted on the AHB slave inputs.

32.4 Instruction trace buffer

The instruction trace buffer consists of a circular buffer that stores executed instructions. The instruction trace buffer is located in the processor, and read out via the DSU. The trace buffer is 128 bits wide, the information stored is indicated in the table below:

*Table 293.*Instruction trace buffer data allocation

Bits	Name	Definition
127	-	Unused
126	Multi-cycle instruction	Set to '1' on the second and third instance of a multi-cycle instruction (LDD, ST or FPOP)
125:96	Time tag	The value of the DSU time tag counter
95:64	Load/Store parameters	Instruction result, Store address or Store data
63:34	Program counter	Program counter (2 lsb bits removed since they are always zero)
33	Instruction trap	Set to '1' if traced instruction trapped
32	Processor error mode	Set to '1' if the traced instruction caused processor error mode
31:0	Opcode	Instruction opcode

During tracing, one instruction is stored per line in the trace buffer with the exception of multi-cycle instructions. Multi-cycle instructions are entered two or three times in the trace buffer. For store instructions, bits [95:64] correspond to the store address on the first entry and to the stored data on the second entry (and third in case of STD). Bit 126 is set on the second and third entry to indicate this. A double load (LDD) is entered twice in the trace buffer, with bits [95:64] containing the loaded data. Bit 126 is set for the second entry.

When the processor enters debug mode, tracing is suspended. The trace buffer and the trace buffer control register can be read and written while the processor is in the debug mode. During the instruction tracing (processor in normal mode) the trace buffer and trace buffer control register 0 can not be written. If the two-port trace buffer is enabled (refer to the *tbuf* generic in section 102.16), then the trace buffer can be read contextually to the instruction tracing (processor in normal mode). The traced instructions can optionally be filtered on instruction types. Which instructions are traced is defined in the instruction trace register [31:28], as defined in the table below:

*Table 294.*Trace filter operation

Trace filter	Instructions traced
0x0	All instructions
0x1	SPARC Format 2 instructions
0x2	Control-flow changes. All Call, branch and trap instructions including branch targets
0x4	SPARC Format 1 instructions (CALL)
0x8	SPARC Format 3 instructions except LOAD or STORE
0xC	SPARC Format 3 LOAD or STORE instructions
0xD	SPARC Format 3 LOAD or STORE instructions to alternate space
0xE	SPARC Format 3 LOAD or STORE instructions to alternate space 0x80 - 0xFF with ASI last digit base filtering

It is also possible to filter traced instructions based on the program counter value. This option is combined with the filtering option if an additional filtering mechanism is activated from Table 294. Refer to section 32.6.13 for detailed information.

32.5 DSU memory map

The DSU memory map can be seen in table 295 below. In a multiprocessor systems, the register map is duplicated and address bits 27 - 24 are used to index the processor.

Note: The DSU memory interface is intended to be accessed by a debug monitor. Software running on the LEON processors should not access the DSU interface. Registers, such as ASR registers, may not have all fields available via the DSU interface

*Table 295.*DSU memory map

Address offset	Register
0x000000	DSU control register
0x000008	Time tag counter
0x000020	Break and Single Step register
0x000024	Debug Mode Mask register
0x000040	AHB trace buffer control register
0x000044	AHB trace buffer index register
0x000048	AHB trace buffer filter control register
0x00004c	AHB trace buffer filter mask register
0x000050	AHB breakpoint address 1
0x000054	AHB mask register 1
0x000058	AHB breakpoint address 2
0x00005c	AHB mask register 2
0x100000 - 0x10FFFF	Instruction trace buffer (..0: Trace bits 127 - 96, ..4: Trace bits 95 - 64, ..8: Trace bits 63 - 32, ..C : Trace bits 31 - 0)
0x110000	Instruction Trace buffer control register 0
0x110004	Instruction Trace buffer control register 1
0x200000 - 0x210000	AHB trace buffer (..0: Trace bits 127 - 96, ..4: Trace bits 95 - 64, ..8: Trace bits 63 - 32, ..C : Trace bits 31 - 0)
0x300000 - 0x3007FC	IU register file, port1 (%asr16.dpsel = 0) IU register file, port 2 (%asr16.dpsel = 1)
0x300800 - 0x300FFC	IU register file check bits (LEON3FT only)

*Table 295.*DSU memory map

Address offset	Register
0x301000 - 0x30107C	FPU register file
0x301800 - 0x30187C	FPU register file check bits (LEON3FT only)
0x400000 - 0x4FFFFC	IU special purpose registers
0x400000	Y register
0x400004	PSR register
0x400008	WIM register
0x40000C	TBR register
0x400010	PC register
0x400014	NPC register
0x400018	FSR register
0x40001C	CPSR register
0x400020	DSU trap register
0x400024	DSU ASI register
0x400040 - 0x40007C	ASR16 - ASR31 (when implemented)
0x700000 - 0x7FFFFC	ASI diagnostic access (ASI = value in DSU ASI register, address = address[19:0]) ASI = 0x9 : Local instruction RAM, ASI = 0xB : Local data RAM ASI = 0xC : Instruction cache tags, ASI = 0xD : Instruction cache data ASI = 0xE : Data cache tags, ASI = 0xF : Data cache data ASI = 0x1E : Separate snoop tags

The addresses of the IU registers depends on how many register windows has been implemented:

- %o*n* : 0x300000 + (((psr.cwp * 64) + 32 + *n**4) mod (NWINDOWS*64))
- %l*n* : 0x300000 + (((psr.cwp * 64) + 64 + *n**4) mod (NWINDOWS*64))
- %i*n* : 0x300000 + (((psr.cwp * 64) + 96 + *n**4) mod (NWINDOWS*64))
- %g*n* : 0x300000 + (NWINDOWS*64) + n*4
- %f*n* : 0x301000 + *n**4

32.6 DSU registers

32.6.1 DSU control register

The DSU is controlled by the DSU control register:

*Table 296.*0x000000 - CTRL - DSU control register

31 … 12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED	PW	HL	PE	EB	EE	DM	BZ	BX	BS	BW	BE	TE
0	0	0	0	*	*		*	*	*	*	*	*
r	r	rw	rw	r	r	r	rw	rw	rw	rw	rw	rw

31: 12 Reserved

11 Power down (PW) - Returns ‘1’ when processor is in power-down mode.

10 Processor halt (HL) - Returns ‘1’ on read when processor is halted. If the processor is in debug mode, setting this bit will put the processor in halt mode.

9 Processor error mode (PE) - returns ‘1’ on read when processor is in error mode, else ‘0’. If written with ‘1’, it will clear the error and halt mode.

8 External Break (EB) - Value of the external DSUBRE signal (read-only)

7 External Enable (EE) - Value of the external DSUEN signal (read-only)

6 Debug mode (DM) - Indicates when the processor has entered debug mode (read-only).

5 Break on error traps (BZ) - if set, will force the processor into debug mode on all *except* the following traps: priviledged_instruction, fpu_disabled, window_overflow, window_underflow, asynchronous_interrupt, ticc_trap.

Table 296. 0x000000 - CTRL - DSU control register

4	Break on trap (BX) - if set, will force the processor into debug mode when any trap occurs.
3	Break on S/W breakpoint (BS) - if set, debug mode will be forced when an breakpoint instruction (ta 1) is executed.
2	Break on IU watchpoint (BW) - if set, debug mode will be forced on a IU watchpoint (trap 0xb).
1	Break on error (BE) - if set, will force the processor to debug mode when the processor would have entered error condition (trap in trap).
0	Trace enable (TE) - Enables instruction tracing. If set the instructions will be stored in the trace buffer. Remains set when then processor enters debug or error mode

32.6.2 DSU Break and Single Step register

This register is used to break or single step the processor(s). This register controls all processors in a multi-processor system, and is only accessible in the DSU memory map of processor 0.

Table 297. 0x000020 - BRSS - BRSS - DSU Break and Single Step register

31 16	15 0
SS[15:0]	BN[15:0]

31: 16	Single step (SSx) - if set, the processor x will execute one instruction and return to debug mode. The bit remains set after the processor goes into the debug mode. As an exception, if the instruction is a branch with the annul bit set, and if the delay instruction is effectively annulled, the processor will execute the branch, the annulled delay instruction and the instruction thereafter before returning to debug mode.
15: 0	Break now (BNx) -Force processor x into debug mode if the Break on watchpoint (BW) bit in the processors DSU control register is set. If cleared, the processor x will resume execution.

32.6.3 DSU Debug Mode Mask Register

When one of the processors in a multiprocessor LEON3 system enters the debug mode the value of the DSU Debug Mode Mask register determines if the other processors are forced in the debug mode. This register controls all processors in a multi-processor system, and is only accessible in the DSU memory map of processor 0.

Table 298. 0x000024 - DBGM - DSU Debug Mode Mask register

31 16	15 0
DM[15:0]	ED[15:0]

31: 16	Debug mode mask (DMx) - If set, the corresponding processor will not be able to force running processors into debug mode even if it enters debug mode.
15: 0	Enter debug mode (EDx) - Force processor x into debug mode if any of processors in a multiprocessor system enters the debug mode. If 0, the processor x will not enter the debug mode.

32.6.4 DSU trap register

The DSU trap register is a read-only register that indicates which SPARC trap type that caused the processor to enter debug mode. When debug mode is force by setting the BN bit in the DSU control register, the trap type will be 0xb (hardware watchpoint trap).

Table 299. 0x400020 - DTR - DSU Trap register

31 13	12	11 4	3 0
RESERVED	EM	TRAPTYPE	R

31: 13 RESERVED

12 Error mode (EM) - Set if the trap would have cause the processor to enter error mode.

11: 4 Trap type (TRAPTYPE) - 8-bit SPARC trap type

3: 0 Read as 0x0

32.6.5 DSU time tag counter

The trace buffer time tag counter is incremented each clock as long as the processor is running. The counter is stopped when the processor enters debug mode and when the DSU is disabled (unless the timer enable bit in the AHB trace buffer control register is set), and restarted when execution is resumed.

Table 300. 0x000008 - DTTC - DSU time tag counter

31 0
TIMETAG
0
rw

31: 0 DSU Time Tag Value (TIMETAG)

The value is used as time tag in the instruction and AHB trace buffer.

The width of the timer is configurable at implementation time.

32.6.6 DSU ASI register

The DSU can perform diagnostic accesses to different ASI areas. The value in the ASI diagnostic access register is used as ASI while the address is supplied from the DSU.

Table 301. 0x400024 - DASI - ASI diagnostic access register

31 8	7 0
RESERVED	ASI
0	NR
r	rw

31: 8 RESERVED

7: 0 ASI (ASI) - ASI to be used on diagnostic ASI access

32.6.7 AHB Trace buffer control register

The AHB trace buffer is controlled by the AHB trace buffer control register:

Table 302. 0x000040 - ATBC - AHB trace buffer control register

31 16	15 8	7	6	5	4 3	2	1	0
DCNT	RESERVED	DF	SF	TE	TF	BW	BR	DM	EN

*Table 302.*0x000040 - ATBC - AHB trace buffer control register

0	0	0	0	0	0	0	0	0	0
rw	r	rw	rw	rw	rw	r	rw	rw	rw

31: 16 Trace buffer delay counter (DCNT) - Note that the number of bits actually implemented depends on the size of the trace buffer.

15: 9 RESERVED

8 Enable Debug Mode Timer Freeze (DF) - The time tag counter keeps counting in debug mode when at least one of the processors has the internal timer enabled. If this bit is set to '1' then the time tag counter is frozen when the processors have entered debug mode.

7 Sample Force (SF) - If this bit is written to '1' it will have the same effect on the AHB trace buffer as if HREADY was asserted on the bus at the same time as a sequential or non-sequential transfer is made. This means that setting this bit to '1' will cause the values in the trace buffer's sample registers to be written into the trace buffer, and new values will be sampled into the registers. This bit will automatically be cleared after one clock cycle.

Writing to the trace buffer still requires that the trace buffer is enabled (EN bit set to '1') and that the CPU is not in debug mode or that tracing is forced (TF bit set to '1'). This functionality is primarily of interest when the trace buffer is tracing a separate bus and the traced bus appears to have frozen.

6 Timer enable (TE) - Activates time tag counter also in debug mode.

5 Trace force (TF) - Activates trace buffer also in debug mode. Note that the trace buffer must be disabled when reading out trace buffer data via the core's register interface.

4: 3 Bus width (BW) - This value corresponds to log2(Supported bus width / 32)

2 Break (BR) - If set, the processor will be put in debug mode when AHB trace buffer stops due to AHB breakpoint hit.

1 Delay counter mode (DM) - Indicates that the trace buffer is in delay counter mode.

0 Trace enable (EN) - Enables the trace buffer.

32.6.8 AHB trace buffer index register

The AHB trace buffer index register contains the address of the next trace line to be written.

*Table 303.*0x000044 - ATBI - AHB trace buffer index register

31 4	3 0
INDEX	R
NR	0
rw	r

31: 4 Trace buffer index counter (INDEX) - Note that the number of bits actually implemented depends on the size of the trace buffer.

3: 0 Read as 0x0

32.6.9 AHB trace buffer filter control register

The trace buffer filter control register is only available if the core has been implemented with support for AHB trace buffer filtering.

*Table 304.*0x000048 - ATBFC - AHB trace buffer filter control register

31 14	13 12	11 10	9 8	7 4	3	2	1	0
RESERVED	WPF	R	BPF	RESERVED	PF	AF	FR	FW
0	0	0	0	0	0	0	0	0
r	rw	r	rw	r	rw	rw	rw	rw

31: 14 RESERVED

*Table 304.*0x000048 - ATBFC - AHB trace buffer filter control register

13: 12	AHB watchpoint filtering (WPF) - Bit 13 of this field applies to AHB watchpoint 2 and bit 12 applies to AHB watchpoint 1. If the WPF bit for a watchpoint is set to '1' then the watchpoint will not trigger unless the access also passes through the filter. This functionality can be used to, for instance, set a AHB watchpoint that only triggers if a specified master performs an access to a specified slave.
11: 10	RESERVED
9: 8	AHB breakpoint filtering (BPF) - Bit 9 of this field applies to AHB breakpoint 2 and bit 8 applies to AHB breakpoint 1. If the BPF bit for a breakpoint is set to '1' then the breakpoint will not trigger unless the access also passes through the filter. This functionality can be used to, for instance, set a AHB breakpoint that only triggers if a specified master performs an access to a specified slave.
7: 4	RESERVED
3	Performance counter Filter (PF) - If this bit is set to '1', the cores performance counter (statistical) outputs will be filtered using the same filter settings as used for the trace buffer. If a filter inhibits a write to the trace buffer, setting this bit to '1' will cause the same filter setting to inhibit the pulse on the statistical output.
2	Address Filter (AF) - If this bit is set to '1', only the address range defined by AHB trace buffer breakpoint 2's address and mask will be included in the trace buffer.
1	Filter Reads (FR) - If this bit is set to '1', read accesses will not be included in the trace buffer.
0	Filter Writes (FW) - If this bit is set to '1', write accesses will not be included in the trace buffer.

32.6.10 AHB trace buffer filter mask register

The trace buffer filter mask register is only available if the core has been implemented with support for AHB trace buffer filtering.

*Table 305.*0x00004C - ATBFM - AHB trace buffer filter mask register

31 16	15 0
SMASK[15:0]	MMASK[15:0]
0	0
rw	rw

31: 16	Slave Mask (SMASK) - If SMASK[n] is set to '1', the trace buffer will not save accesses performed to slave n.
15: 0	Master Mask (MMASK) - If MMASK[n] is set to '1', the trace buffer will not save accesses performed by master n.

32.6.11 AHB trace buffer breakpoint registers

The DSU contains two breakpoint registers for matching AHB addresses. A breakpoint hit is used to freeze the trace buffer by automatically clearing the enable bit. Freezing can be delayed by programming the DCNT field in the trace buffer control register to a non-zero value. In this case, the DCNT value will be decremented for each additional trace until it reaches zero, after which the trace buffer is frozen. A mask register is associated with each breakpoint, allowing breaking on a block of addresses. Only address bits with the corresponding mask bit set to '1' are compared during breakpoint detection. To break on AHB load or store accesses, the LD and/or ST bits should be set.

*Table 306.*0x000050, 0x000058 - ATBBA - AHB trace buffer break address register

31 2	1	0
BADDR[31:2]		R
NR		0
rw		r

31: 2	Break point address (BADDR) - Bits 31:2 of breakpoint address
1: 0	Read as 0b00

*Table 307.*0x000054, 0x00005C - ATBBM - AHB trace buffer break mask register

31 2	1	0
BMASK[31:2]	LD	ST
NR	0	0
rw	rw	rw

31: 2	Breakpoint mask (BMASK) - (see text)
1	Load (LD) - Break on data load address
0	Store (ST) - Break on data store address

32.6.12 Instruction trace control register 0

The instruction trace control register 0 contains a pointer that indicates the next line of the instruction trace buffer to be written.

*Table 308.*0x110000 - ITBCO - Instruction trace control register 0

31 29	28 16	15 0
	RESERVED	ITPOINTER
	0	NR
	r	rw

31: 28	Trace filter configuration
27: 16	RESERVED
15: 0	Instruction trace pointer (ITPOINTER) - Note that the number of bits actually implemented depends on the size of the trace buffer

32.6.13 Instruction trace control register 1

The instruction trace control register 1 contains settings used for trace buffer overflow detection, in addition it includes settings used for some of the instruction trace buffer filtering options. This register can be written while the processor is running.

Bits [31:28] is used to enable or disable Instruction Trace Buffer Address based Filtering (ITBAF). ITBAF is intended to allow the available hardware watch-point (HWP) registers to be used as instruction trace buffer filters when they are not used for breakpoint operation. If a bit is set to ‘1’ in ITBAF, the corresponding address and mask information in the HWP register will be used to filter instruction trace entries based on the program counter (PC) value. Bits[31:28] corresponds to HWP[3:0] respectively. ITBAF can only be used if the corresponding HWP register exist in the hardware. Instruction Trace Buffer Address based Filtering Option (ITBAFO, Bits[19:16]) determines the type of filtering for the corresponding ITBAF entry. If an ITBAFO entry is set to ‘0’ only the PC value(s) that match the address and mask option in the corresponding HWP register will be logged in the instruction trace buffer (ITB). If a bit is set to ‘1’ only the PC value(s) that does not match the address and mask option in the corresponding HWP register will be logged in the ITB. Bits[19:16] corresponds to the option for ITBAF[3:0] respectively. If there is more than one address filtering operation is enabled, the corresponding filtering operations will be combined together.

Bits[15:0] corresponds to ASI last digit based filtering mask (ASIFMASK). ASIFMASK is in effect when the trace filter configuration is set to 0xE (SPARC Format 3 LOAD or STORE instructions to alternate space 0x80 - 0xFF with ASI last digit base filtering). Bits[15:0] corresponds to digits [0xF:0x0] respectively. If a bit is set to ‘0’ in the ASIFMASK, the load and store instructions which have an ASI between the range of 0x80-0xFF and have the corresponding last digit are logged in the instruction trace buffer. For example if only the bit0 and bit2 of the ASIFMASK are ‘0’ then only the load and store instructions with ASIs 0x80, 0x82, 0x90, 0x92, 0xA0, 0xA2, 0xB0, 0xB2, 0xC0, 0xC2, 0xD0, 0xD2, 0xE0, 0xE2, 0xF0, 0xF2 are tracked in the ITB. After the reset of processor all the bits

in the ASIFMASK is set to 0x0000 which means by default all the ASIs in the range of 0x80-0xFF are tracked.

*Table 309.*0x110004 - ITBCI - Instruction trace control register 1

31 28	27	26 24	23	22 20	19 16	15 0
ITBAF	WO	TLIM	OV	RESERVED	ITBAFO	ASIFMASK
0	0	0	0	0	0	0
rw	rw	rw	rw	r	rw	rw

31: 28 Instruction Trace Buffer Address based Filtering (ITBAF) (see text)

27 Watchpoint on overflow (WO) - If this bit is set, and Break on iu watchpoint (BW) is enabled in the DSU control register, then a watchpoint will be inserted when a trace overflow is detected (TOV field in this register gets set).

26: 24 Trace Limit (TLIM) - TLIM is compared with the top bits of ITPOINTER in Instruction trace control register 0 to generate the value in the TOV field below.

23 Trace Overflow (TOV) - Gets set to '1' when the DSU detects that TLIM equals the top three bits of ITPOINTER.

22: 20 RESERVED

19: 16 Instruction Trace Buffer Address based Filtering Option (ITBAFO) (see text)

15: 0 ASI last digit based filtering mask (ASIFMASK) (see text)

32.7 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x017. For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

32.8 Implementation

32.8.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

32.8.2 Technology mapping

DSU3 has one technology mapping generic, *tech*. This generic controls the implementation of which technology that will be used to implement the trace buffer memories. The AHB trace buffer will use two identical SYNCRAM64 blocks to implement the buffer memory (SYNCRAM64 may then result in two 32-bit wide memories on the target technology, depending on the technology map), with one additional 32-bit wide SYNCRAM if the system's AMBA data bus width is 64-bits, and also one additional 64-bit wide SYNCRAM if the system's AMBA data bus width exceeds 64 bits.

The depth of the RAMs depends on the KBYTES generic. If KBYTES = 1 (1 Kbyte), then the depth will be 64. If KBYTES = 2, then the RAM depth will be 128 and so on.

32.9 Configuration options

Table 310 shows the configuration options of the core (VHDL generics).

*Table 310.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	0 - NAHBSLV-1	0
haddr	AHB slave address (AHB[31:20])	0 - 16#FFF#	16#900#
hmask	AHB slave address mask	0 - 16#FFF#	16#F00#
ncpu	Number of attached processors	1 - 16	1
tbits	Number of bits in the time tag counter	2 - 63	30
tech	Memory technology for trace buffer RAM	0 - NTECH-1	0 (inferred)
kbytes	Size of trace buffer memory in Kbytes. A value of 0 will disable the trace buffer function.	0 - 64	0 (disabled)
clk2x	Support for LEON3 double-clocking (this generic is only available on dsu3x entity), see next section.	0 - 1	0 (disabled)
testen	Scan test support enable	0 - 1	0
bwidth	Traced AHB bus width	32, 64, 128	32
ahbpf	AHB performance counters and filtering. If *ahbpf* is non-zero the core will support AHB trace buffer filtering. If *ahbpf* is larger than 1 then the core's statistical outputs will be enabled.	0 - 2	0

32.10 Signal descriptions

Table 311 shows the interface signals of the core (VHDL ports). There are several top-level entities available for the DSU3. The dsu3x entity contains all signals and settings. The other entities are wrappers around dsu3x. The available entities are:

- dsu3 - Entity without support for double clocking. AHB trace of same bus as DSU AHB slave interface is connected to.
- dsu3_2x - Entity with support for LEON3 double-clocking. AHB trace of same bus as DSU AHB slave interface is connected to.
- dsu3_mb - Entity with support for AHB tracing of separate bus
- dsu3x - Entity with support for all features (double-clocking and tracing of separate bus)

32.11 Signal definitions and reset values

*Table 311.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	CPU and bus clock, on dsu3 and dsu3_mb entites	-
HCLK	N/A	Input	Bus clock, on dsu3_2x and dsu3x entities. Only used when double-clocking is enabled.	-
CPUCLK	N/A	Input	CPU clock, on dsu3_2x and dsu3x entities	-
AHBMI	*	Input	AHB master input signals, used for AHB tracing	-
AHBSI	*	Input	AHB slave input signals, used for AHB tracing when using dsu3 and dsu3_2x entities	-
AHBSO	*	Output	AHB slave output signals	-
TAHBSI	*	Input	AHB slave input signals, used for AHB tracing when using dsu3_mb and dsu3x entities	-
DBGI	-	Input	Debug signals from LEON3	-
DBGO	-	Output	Debug signals to LEON3	-
DSUI	ENABLE	Input	DSU enable	High
	BREAK	Input	DSU break	High
DSUO	ACTIVE	Output	Debug mode	High
	PWD[n-1 : 0]	Output	Clock gating enable for processor [n]	High
	ASTAT (record)	Output	AHB statistic/performance counter events	-
HCLKEN	N/A	Input	Double-clocking qualifier signal. Only used with double-clocking on dsu4_2x and dsu4x entities	High

* see GRLIB IP Library User's Manual

The signals and their reset values are described in table 312.

*Table 312.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
dsuen	Input	DSU enable	High	-
dsubre	Input	DSU break	High	-
dsuact	Output	Debug mode	High	Logical 0

32.12 Timing

The timing waveforms and timing parameters are shown in figure 68 and are defined in table 313.

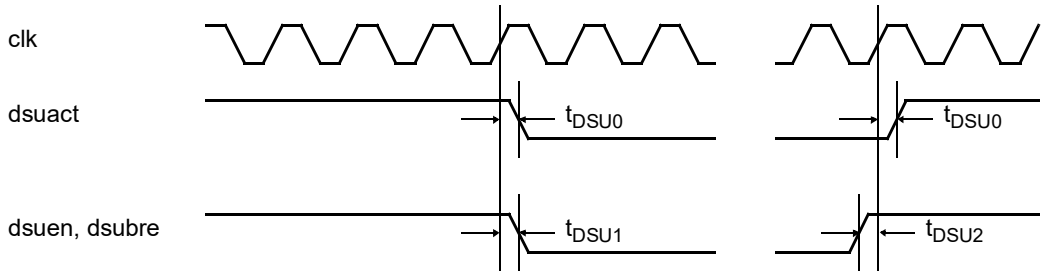


Figure 68. Timing waveforms

*Table 313.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{DSU0}	clock to output delay	rising *clk* edge	TBD	TBD	ns
t_{DSU1}	input to clock hold	rising *clk* edge	-	-	ns
t_{DSU2}	input to clock setup	rising *clk* edge	-	-	ns

- Note: The *dsubre* and *dsuen* are re-synchronized internally. These signals do not have to meet any setup or hold requirements.

32.13 Library dependencies

Table 314 shows libraries used when instantiating the core (VHDL libraries).

*Table 314.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	LEON3	Component, signals	Component declaration, signals declaration

32.14 Component declaration

The core has the following component declaration.

```
component dsu3
  generic (
    hindex : integer := 0;
    haddr : integer := 16#900#;
    hmask : integer := 16#f00#;
    ncpu    : integer := 1;
    tbits   : integer := 30;
    tech    : integer := 0;
    irq     : integer := 0;
    kbytes  : integer := 0
  );
  port (
    rst    : in  std_ulogic;
    clk    : in  std_ulogic;
    ahbmi  : in  ahb_mst_in_type;
    ahbsi  : in  ahb_slv_in_type;
    ahbso  : out ahb_slv_out_type;
    dbgi   : in l3_debug_out_vector(0 to NCPU-1);
    dbgo   : out l3_debug_in_vector(0 to NCPU-1);
    dsui   : in dsu_in_type;
    dsuo   : out dsu_out_type
```

```
  );
  end component;
```

32.15 Instantiation

This example shows how the core can be instantiated.

The DSU is always instantiated with at least one LEON3 processor. It is suitable to use a generate loop for the instantiation of the processors and DSU and showed below.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.leon3.all;

constant NCPU : integer := 1; -- select number of processors

signal leon3i : l3_in_vector(0 to NCPU-1);
signal leon3o : l3_out_vector(0 to NCPU-1);
signal irqi   : irq_in_vector(0 to NCPU-1);
signal irqo   : irq_out_vector(0 to NCPU-1);

signal dbgi : l3_debug_in_vector(0 to NCPU-1);
signal dbgo : l3_debug_out_vector(0 to NCPU-1);

signal dsui   : dsu_in_type;
signal dsuo   : dsu_out_type;

.
begin

cpu : for i in 0 to NCPU-1 generate
    u0 : leon3s-- LEON3 processor
    generic map (ahbndx => i, fabtech => FABTECH, memtech => MEMTECH)
    port map (clkm, rstn, ahbmi, ahbmo(i), ahbsi, ahbsi, ahbso,
 irqi(i), irqo(i), dbgi(i), dbgo(i));
    irqi(i) <= leon3o(i).irq; leon3i(i).irq <= irqo(i);
end generate;

dsu0 : dsu3-- LEON3 Debug Support Unit
    generic map (ahbndx => 2, ncpu => NCPU, tech => memtech, kbytes => 2)
    port map (rstn, clkm, ahbmi, ahbsi, ahbso(2), dbgo, dbgi, dsui, dsuo);
dsui.enable <= dsuen; dsui.break <= dsubre; dsuact <= dsuo.active;
```

33 DSU4 - LEON4 Hardware Debug Support Unit

33.1 Overview

To simplify debugging on target hardware, the LEON4 processor implements a debug mode during which the pipeline is idle and the processor is controlled through a special debug interface. The LEON4 Debug Support Unit (DSU) is used to control the processor during debug mode. The DSU acts as an AHB slave and can be accessed by any AHB master. An external debug host can therefore access the DSU through several different interfaces. Such an interface can be a serial UART (RS232), JTAG, PCI, USB or Ethernet. The DSU supports multi-processor systems and can handle up to 16 processors.

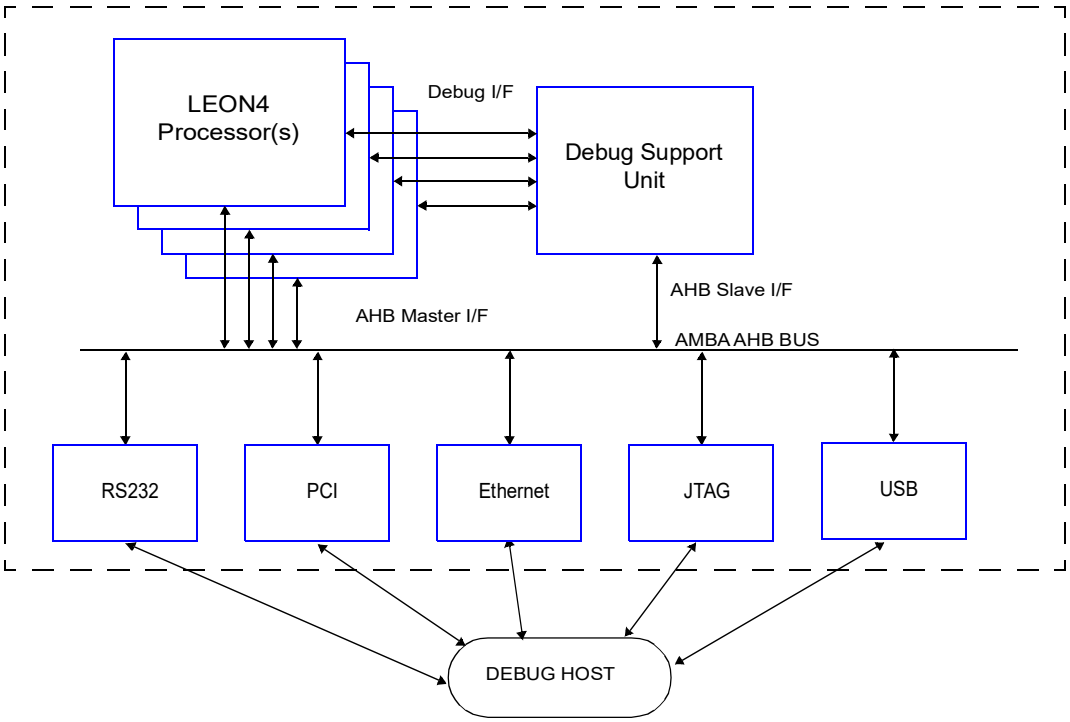


Figure 69. LEON4/DSU Connection

33.2 Operation

Through the DSU AHB slave interface, any AHB master can access the processor registers and the contents of the instruction trace buffer. The DSU control registers can be accessed at any time, while the processor registers and caches can only be accessed when the processor has entered debug mode. In debug mode, the processor pipeline is held and the processor state can be accessed by the DSU. Entering the debug mode can occur on the following events:

- executing a breakpoint instruction (ta 1)
- integer unit hardware breakpoint/watchpoint hit (trap 0xb)
- rising edge of the external break signal (DSUBRE)
- setting the break-now (BN) bit in the DSU control register
- a trap that would cause the processor to enter error mode
- occurrence of any, or a selection of traps as defined in the DSU control register
- after a single-step operation
- one of the processors in a multiprocessor system has entered the debug mode
- DSU AHB breakpoint or watchpoint hit

The debug mode can only be entered when the debug support unit is enabled through an external signal (DSUEN). For DSU break, and the break-now BN bit, to have effect the Break-on-IU-watchpoint (BW) bit must be set in the DSU control register. This bit is set when DSUBRE is active after reset and should also be set by debug monitor software when initializing the DSU. When the debug mode is entered, the following actions are taken:

- PC and nPC are saved in temporary registers (accessible by the debug unit)
- an output signal (DSUACT) is asserted to indicate the debug state
- the timer unit is (optionally) stopped to freeze the LEON timers and watchdog

The instruction that caused the processor to enter debug mode is not executed, and the processor state is kept unmodified. Execution is resumed by clearing the BN bit in the DSU control register or by de-asserting DSUEN. The timer unit will be re-enabled and execution will continue from the saved PC and nPC. Debug mode can also be entered after the processor has entered error mode, for instance when an application has terminated and halted the processor. The error mode can be reset and the processor restarted at any address.

When a processor is in the debug mode, an access to ASI diagnostic area is forwarded to the IU which performs access with ASI equal to value in the DSU ASI register and address consisting of 20 LSB bits of the original address.

33.3 AHB trace buffer

The AHB trace buffer consists of a circular buffer that stores AHB data transfers, the monitored AHB bus is either the same bus as the DSU AHB slave interface is connected to, or a completely separate bus. The address, data and various control signals of the AHB bus are stored and can be read out for later analysis. The trace buffer is 128, 160 or 224 bits wide, depending on the AHB bus width. Note that for systems with a 256 bit wide AHB bus, the DSU3 can be used but will only save the lower 128 bits of the AHB data bus. The way information stored is indicated in the table below:

*Table 315.*AHB Trace buffer data allocation

Bits	Name	Definition
223:160	Load/Store data	AHB HRDATA/HWDATA(127:64)
159:129	Load/Store data	AHB HRDATA/HWDATA(63:32)
127	AHB breakpoint hit	Set to ‘1’ if a DSU AHB breakpoint hit occurred.
126	-	Not used
125:96	Time tag	DSU time tag counter
95:80	-	Not used
79	Hwrite	AHB HWRITE
78:77	Htrans	AHB HTRANS
76:74	Hsize	AHB HSIZE
73:71	Hburst	AHB HBURST
70:67	Hmaster	AHB HMASTER
66	Hmastlock	AHB HMASTLOCK
65:64	Hresp	AHB HRESP
63:32	Load/Store data	AHB HRDATA/HWDATA(31:0)
31:0	Load/Store address	AHB HADDR

In addition to the AHB signals, the DSU time tag counter is also stored in the trace.

The trace buffer is enabled by setting the enable bit (EN) in the trace control register. Each AHB transfer is then stored in the buffer in a circular manner. The address to which the next transfer is writ-

ten is held in the trace buffer index register, and is automatically incremented after each transfer. Tracing is stopped when the EN bit is reset, or when a AHB breakpoint is hit. Tracing is temporarily suspended when the processor enters debug mode, unless the trace force bit (TF) in the trace control register is set. If the trace force bit is set, the trace buffer is activated as long as the enable bit is set. The force bit is reset if an AHB breakpoint is hit and can also be cleared by software. Note that neither the trace buffer memory nor the breakpoint registers (see below) can be read/written by software when the trace buffer is enabled.

The DSU has an internal time tag counter and this counter is frozen when the processor enters debug mode. When AHB tracing is performed in debug mode (using the trace force bit) it may be desirable to also enable the time tag counter. This can be done using the timer enable bit (TE). Note that the time tag is also used for the instruction trace buffer and the timer enable bit should only be set when using the DSU as an AHB trace buffer only, and not when performing profiling or software debugging. The timer enable bit is reset on the same events as the trace force bit.

33.3.1 AHB trace buffer filters

The DSU can be implemented with filters that can be applied to the AHB trace buffer, breakpoints and watchpoints. If implemented, these filters are controlled via the AHB trace buffer filter control and AHB trace buffer filter mask registers. The fields in these registers allows masking access characteristics such as master, slave, read, write and address range so that accesses that correspond to the specified mask are not written into the trace buffer. Address range masking is done using the second AHB breakpoint register set. The values of the LD and ST fields of this register has no effect on filtering.

33.3.2 AHB statistics

The DSU can be implemented to generate statistics from the traced AHB bus. When statistics collection is enabled the DSU will assert outputs that are suitable to connect to a LEON4 statistics unit (L4STAT). The statistical outputs can be filtered by the AHB trace buffer filters, this is controlled by the Performance counter Filter bit (PF) in the AHB trace buffer filter control register. The DSU can collect data for the events listed in table 316 below.

Table 316. AHB events

Event	Description	Note
idle	HTRANS=IDLE	Active when HTRANS IDLE is driven on the AHB slave inputs and slave has asserted HREADY.
busy	HTRANS=BUSY	Active when HTRANS BUSY is driven on the AHB slave inputs and slave has asserted HREADY.
nseq	HTRANS=NONSEQ	Active when HTRANS NONSEQ is driven on the AHB slave inputs and slave has asserted HREADY.
seq	HTRANS=SEQ	Active when HTRANS SEQUENTIAL is driven on the AHB slave inputs and slave has asserted HREADY.
read	Read access	Active when HTRANS is SEQUENTIAL or NON-SEQUENTIAL, slave has asserted HREADY and the HWRITE input is low.
write	Write access	Active when HTRANS is SEQUENTIAL or NON-SEQUENTIAL, slave has asserted HREADY and the HWRITE input is high.
hsize[5:0]	Transfer size	Active when HTRANS is SEQUENTIAL or NON-SEQUENTIAL, slave has asserted HREADY and HSIZE is BYTE (hsize[0]), HWORD (HSIZE[1]), WORD (hsize[2]), DWORD (hsize[3]), 4WORD hsize[4], or 8WORD (hsize[5]).
ws	Wait state	Active when HREADY input to AHB slaves is low and AMBA response is OKAY.
retry	RETRY response	Active when master receives RETRY response
split	SPLIT response	Active when master receives SPLIT response

*Table 316.*AHB events

Event	Description	Note
spdel	SPLIT delay	Active during the time a master waits to be granted access to the bus after reception of a SPLIT response. The core will only keep track of one master at a time. This means that when a SPLIT response is detected, the core will save the master index. This event will then be active until the same master is re-allowed into bus arbitration and is granted access to the bus. This also means that the delay measured will include the time for re-arbitration, delays from other ongoing transfers and delays resulting from other masters being granted access to the bus before the SPLIT:ed master is granted again after receiving SPLIT complete. If another master receives a SPLIT response while this event is active, the SPLIT delay for the second master will not be measured.
locked	Locked access	Active while the HMASTLOCK signal is asserted on the AHB slave inputs.

33.4 Instruction trace buffer

The instruction trace buffer consists of a circular buffer that stores executed instructions. The instruction trace buffer is located in the processor, and read out via the DSU. The trace buffer is 128 bits wide, the information stored is indicated in the table below:

*Table 317.*Instruction trace buffer data allocation

Bits	Name	Definition
126	Multi-cycle instruction	Set to '1' on the second instance of a multi-cycle instruction
125:96	Time tag	The value of the DSU time tag counter
95:64	Result or Store address/data	Instruction result, Store address or Store data
63:34	Program counter	Program counter (2 lsb bits removed since they are always zero)
33	Instruction trap	Set to '1' if traced instruction trapped
32	Processor error mode	Set to '1' if the traced instruction caused processor error mode
31:0	Opcode	Instruction opcode

During tracing, one instruction is stored per line in the trace buffer with the exception of atomic load/store instructions, which are entered twice (one for the load and one for the store operation). Bits [95:64] in the buffer correspond to the store address and the loaded data for load instructions. Bit 126 is set for the second entry.

When the processor enters debug mode, tracing is suspended. The trace buffer and the trace buffer control register can be read and written while the processor is in the debug mode. During the instruction tracing (processor in normal mode) the trace buffer and trace buffer control register 0 can not be written. If the two-port trace buffer is enabled (refer to the *tbuf* generic in section 103.15), then the trace buffer can be read contextually to the instruction tracing (processor in normal mode). The traced instructions can optionally be filtered on instruction types. Which instructions are traced is defined in the instruction trace register [31:28], as defined in the table below:

*Table 318.*Trace filter operation

Trace filter	Instructions traced
0x0	All instructions
0x1	SPARC Format 2 instructions
0x2	Control-flow changes. All Call, branch and trap instructions including branch targets
0x4	SPARC Format 1 instructions (CALL)
0x8	SPARC Format 3 instructions except LOAD or STORE
0xC	SPARC Format 3 LOAD or STORE instructions
0xD	SPARC Format 3 LOAD or STORE instructions to alternate space
0xE	SPARC Format 3 LOAD or STORE instructions to alternate space 0x80 - 0xFF with ASI last digit base filtering

It is also possible to filter traced instructions based on the program counter value. This option is combined with the filtering option if an additional filtering mechanism is activated from Table 318. Refer to section 33.6.13 for detailed information.

33.5 DSU memory map

The DSU memory map can be seen in table 319 below. In a multiprocessor systems, the register map is duplicated and address bits 27 - 24 are used to index the processor.

Note: The DSU memory interface is intended to be accessed by a debug monitor. Software running on the LEON processors should not access the DSU interface. Registers, such as ASR registers, may not have all fields available via the DSU interface.

*Table 319.*DSU memory map

Address offset	Register
0x000000	DSU control register
0x000008	Time tag counter
0x000020	Break and Single Step register
0x000024	Debug Mode Mask register
0x000040	AHB trace buffer control register
0x000044	AHB trace buffer index register
0x000048	AHB trace buffer filter control register
0x00004c	AHB trace buffer filter mask register
0x000050	AHB breakpoint address 1
0x000054	AHB mask register 1
0x000058	AHB breakpoint address 2
0x00005c	AHB mask register 2
0x000070	Instruction count register
0x000080	AHB watchpoint control register
0x000090 - 0x00009C	AHB watchpoint 1 data registers
0x0000A0 - 0x0000AC	AHB watchpoint 1 mask registers
0x0000B0 - 0x0000BC	AHB watchpoint 2 data registers
0x0000C0 - 0x0000CC	AHB watchpoint 2 mask registers
0x100000 - 0x10FFFF	Instruction trace buffer (..0: Trace bits 127 - 96, ..4: Trace bits 95 - 64, ..8: Trace bits 63 - 32, ..C : Trace bits 31 - 0)

*Table 319.*DSU memory map

Address offset	Register
0x110000	Instruction Trace buffer control register 0
0x110004	Instruction Trace buffer control register 1
0x200000 - 0x210000	AHB trace buffer (..0: Trace bits 127 - 96, ..4: Trace bits 95 - 64, ..8: Trace bits 63 - 32, ..C : Trace bits 31 - 0)
0x300000 - 0x3007FC	IU register file. The addresses of the IU registers depends on how many register windows has been implemented: %o*n*: 0x300000 + (((psr.cwp * 64) + 32 + *n**4) mod (NWINDOWS*64)) %l*n*: 0x300000 + (((psr.cwp * 64) + 64 + *n**4) mod (NWINDOWS*64)) %i*n*: 0x300000 + (((psr.cwp * 64) + 96 + *n**4) mod (NWINDOWS*64)) %g*n*: 0x300000 + (NWINDOWS*64) + n*4 %f*n*: 0x301000 + *n**4
0x300800 - 0x300FFC	IU register file check bits (LEON4FT only)
0x301000 - 0x30107C	FPU register file
0x400000	Y register
0x400004	PSR register
0x400008	WIM register
0x40000C	TBR register
0x400010	PC register
0x400014	NPC register
0x400018	FSR register
0x40001C	CPSR register
0x400020	DSU trap register
0x400024	DSU ASI register
0x400040 - 0x40007C	ASR16 - ASR31 (when implemented)
0x700000 - 0x7FFFFC	ASI diagnostic access (ASI = value in DSU ASI register, address = address[19:0]) ASI = 0x9 : Local instruction RAM ASI = 0xB : Local data RAM ASI = 0xC : Instruction cache tags ASI = 0xD : Instruction cache data ASI = 0xE : Data cache tags ASI = 0xF : Data cache data ASI = 0x1E : Separate snoop tags

33.6 DSU registers

33.6.1 DSU control register

The DSU is controlled by the DSU control register:

*Table 320.*0x000000 - CTRL - DSU control register

31 12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED	PW	HL	PE	EB	EE	DM	BZ	BX	BS	BW	BE	TE
0	0	0	0	*	*		*	*	0	*	*	*
r	r	rw	uc	r	r	r	rw	rw	rw	rw	rw	rw

31: 12 Reserved

11 Power down (PW) - Returns ‘1’ when processor is in power-down mode.

10 Processor halt (HL) - Returns ‘1’ on read when processor is halted. If the processor is in debug mode, setting this bit will put the processor in halt mode.

*Table 320.*0x000000 - CTRL - DSU control register

9	Processor error mode (PE) - returns '1' on read when processor is in error mode, else '0'. If written with '1', it will clear the error and halt mode.
8	External Break (EB) - Value of the external DSUBRE signal (read-only)
7	External Enable (EE) - Value of the external DSUEN signal (read-only)
6	Debug mode (DM) - Indicates when the processor has entered debug mode (read-only).
5	Break on error traps (BZ) - if set, will force the processor into debug mode on all *except* the following traps: priviledged_instruction, fpu_disabled, window_overflow, window_underflow, asynchronous_interrupt, ticc_trap.
4	Break on trap (BX) - if set, will force the processor into debug mode when any trap occurs.
3	Break on S/W breakpoint (BS) - if set, debug mode will be forced when an breakpoint instruction (ta 1) is executed.
2	Break on IU watchpoint (BW) - if set, debug mode will be forced on a IU watchpoint (trap 0xb).
1	Break on error (BE) - if set, will force the processor to debug mode when the processor would have entered error condition (trap in trap).
0	Trace enable (TE) - Enables instruction tracing. If set the instructions will be stored in the trace buffer. Remains set when then processor enters debug or error mode

33.6.2 DSU Break and Single Step register

This register is used to break or single step the processor(s). This register controls all processors in a multi-processor system, and is only accessible in the DSU memory map of processor 0.

*Table 321.*0x000020 - BRSS - DSU Break and Single Step register

31 16	15 0
SS[15:0]	BN[15:0]
0	0
rw	rw

31: 16	Single step (SSx) - if set, the processor x will execute one instruction and return to debug mode. The bit remains set after the processor goes into the debug mode. As an exception, if the instruction is a branch with the annul bit set, and if the delay instruction is effectively annulled, the processor will execute the branch, the annulled delay instruction and the instruction thereafter before returning to debug mode.
15: 0	Break now (BNx) -Force processor x into debug mode if the Break on watchpoint (BW) bit in the processors DSU control register is set. If cleared, the processor x will resume execution.

33.6.3 DSU Debug Mode Mask Register

When one of the processors in a multiprocessor LEON4 system enters the debug mode the value of the DSU Debug Mode Mask register determines if the other processors are forced in the debug mode. This register controls all processors in a multi-processor system, and is only accessible in the DSU memory map of processor 0.

*Table 322.*0x000024 - DBGM - DSU Debug Mode Mask register

31 16	15 0
DM[15:0]	ED[15:0]
0	0
rw	rw

31: 16	Debug mode mask (DMx) - If set, the corresponding processor will not be able to force running processors into debug mode even if it enters debug mode.
15: 0	Enter debug mode (EDx) - Force processor x into debug mode if any of processors in a multiprocessor system enters the debug mode. If 0, the processor x will not enter the debug mode.

33.6.4 DSU trap register

The DSU trap register is a read-only register that indicates which SPARC trap type that caused the processor to enter debug mode. When debug mode is force by setting the BN bit in the DSU control register, the trap type will be 0xb (hardware watchpoint trap).

*Table 323.*0x400020 - DTR - DSU Trap register

31 13	12	11 4	3 0
RESERVED	EM	TRAPTYPE	R
0	NR	NR	0
r	r	r	r

31: 13 RESERVED

12 Error mode (EM) - Set if the trap would have cause the processor to enter error mode.

11: 4 Trap type (TRAPTYPE) - 8-bit SPARC trap type

3: 0 Read as 0x0

33.6.5 DSU time tag counter

The trace buffer time tag counter is incremented each clock as long as the processor is running. The counter is stopped when the processor enters debug mode and when the DSU is disabled (unless the timer enable bit in the AHB trace buffer control register is set), and restarted when execution is resumed.

*Table 324.*0x000008 - DTTL - DSU time tag counter

31 0
TIMETAG
0
rw

31: 0 DSU Time Tag Value (TIMETAG)

The value is used as time tag in the instruction and AHB trace buffer.

The width of the timer is configurable at implementation time.

33.6.6 DSU ASI register

The DSU can perform diagnostic accesses to different ASI areas. The value in the ASI diagnostic access register is used as ASI while the address is supplied from the DSU.

*Table 325.*0x400024 -DASI - ASI diagnostic access register

31 8	7 0
RESERVED	ASI
0	NR
r	rw

31: 8 RESERVED

7: 0 ASI (ASI) - ASI to be used on diagnostic ASI access

33.6.7 AHB Trace buffer control register

The AHB trace buffer is controlled by the AHB trace buffer control register:

Table 326. 0x000040 - ATBC - AHB trace buffer control register

31 16	15 9	8	7	6	5	4 3	2	1	0
DCNT	RESERVED	DF	SF	TE	TF	BW	BR	DM	EN
0	0	0	0	0	0	*	0	0	*
rw	r	rw	rw	rw	rw	r	rw	rw	rw

31: 16 Trace buffer delay counter (DCNT) - Note that the number of bits actually implemented depends on the size of the trace buffer.

15: 9 RESERVED

8 Enable Debug Mode Timer Freeze (DF) - The time tag counter keeps counting in debug mode when at least one of the processors has the internal timer enabled. If this bit is set to '1' then the time tag counter is frozen when the processors have entered debug mode.

7 Sample Force (SF) - If this bit is written to '1' it will have the same effect on the AHB trace buffer as if HREADY was asserted on the bus at the same time as a sequential or non-sequential transfer is made. This means that setting this bit to '1' will cause the values in the trace buffer's sample registers to be written into the trace buffer, and new values will be sampled into the registers. This bit will automatically be cleared after one clock cycle.

Writing to the trace buffer still requires that the trace buffer is enabled (EN bit set to '1') and that the CPU is not in debug mode or that tracing is forced (TF bit set to '1'). This functionality is primarily of interest when the trace buffer is tracing a separate bus and the traced bus appears to have frozen.

6 Timer enable (TE) - Activates time tag counter also in debug mode.

5 Trace force (TF) - Activates trace buffer also in debug mode. Note that the trace buffer must be disabled when reading out trace buffer data via the core's register interface.

4: 3 Bus width (BW) - This value corresponds to log2(Supported bus width / 32)

2 Break (BR) - If set, the processor will be put in debug mode when AHB trace buffer stops due to AHB breakpoint hit.

1 Delay counter mode (DM) - Indicates that the trace buffer is in delay counter mode.

0 Trace enable (EN) - Enables the trace buffer.

33.6.8 AHB trace buffer index register

The AHB trace buffer index register contains the address of the next trace line to be written.

Table 327. 0x000044 - ATBI - AHB trace buffer index register

31 4	3 0
INDEX	R
NR	0
rw	r

31: 4 Trace buffer index counter (INDEX) - Note that the number of bits actually implemented depends on the size of the trace buffer.

3: 0 Read as 0x0

33.6.9 AHB trace buffer filter control register

The trace buffer filter control register is only available if the core has been implemented with support for AHB trace buffer filtering.

Table 328. 0x000048 - ATBFC - AHB trace buffer filter control register

31 14	13 12	11 10	9 8	7 4	3	2	1	0
RESERVED	WPF	R	BPF	RESERVED	PF	AF	FR	FW
0	0	0	0	0	0	0	0	0
r	rw	r	rw	r	rw	rw	rw	rw

*Table 328.*0x000048 - ATBFC - AHB trace buffer filter control register

31: 14	RESERVED
13: 12	AHB watchpoint filtering (WPF) - Bit 13 of this field applies to AHB watchpoint 2 and bit 12 applies to AHB watchpoint 1. If the WPF bit for a watchpoint is set to '1' then the watchpoint will not trigger unless the access also passes through the filter. This functionality can be used to, for instance, set a AHB watchpoint that only triggers if a specified master performs an access to a specified slave.
11: 10	RESERVED
9: 8	AHB breakpoint filtering (BPF) - Bit 9 of this field applies to AHB breakpoint 2 and bit 8 applies to AHB breakpoint 1. If the BPF bit for a breakpoint is set to '1' then the breakpoint will not trigger unless the access also passes through the filter. This functionality can be used to, for instance, set a AHB breakpoint that only triggers if a specified master performs an access to a specified slave. Note that if a AHB breakpoint is coupled with an AHB watchpoint then the setting of the corresponding bit in this field has no effect.
7: 4	RESERVED
3	Performance counter Filter (PF) - If this bit is set to '1', the cores performance counter (statistical) outputs will be filtered using the same filter settings as used for the trace buffer. If a filter inhibits a write to the trace buffer, setting this bit to '1' will cause the same filter setting to inhibit the pulse on the statistical output.
2	Address Filter (AF) - If this bit is set to '1', only the address range defined by AHB trace buffer breakpoint 2's address and mask will be included in the trace buffer.
1	Filter Reads (FR) - If this bit is set to '1', read accesses will not be included in the trace buffer.
0	Filter Writes (FW) - If this bit is set to '1', write accesses will not be included in the trace buffer.

33.6.10 AHB trace buffer filter mask register

The trace buffer filter mask register is only available if the core has been implemented with support for AHB trace buffer filtering.

*Table 329.*0x00004C - ATBFM - AHB trace buffer filter mask register

31 16	15 0
SMASK[15:0]	MMASK[15:0]
0	0
rw	rw

31: 16	Slave Mask (SMASK) - If SMASK[n] is set to '1', the trace buffer will not save accesses performed to slave n.
15: 0	Master Mask (MMASK) - If MMASK[n] is set to '1', the trace buffer will not save accesses performed by master n.

33.6.11 AHB trace buffer breakpoint registers

The DSU contains two breakpoint registers for matching AHB addresses. A breakpoint hit is used to freeze the trace buffer by automatically clearing the enable bit. Freezing can be delayed by programming the DCNT field in the trace buffer control register to a non-zero value. In this case, the DCNT value will be decremented for each additional trace until it reaches zero, after which the trace buffer is frozen. A mask register is associated with each breakpoint, allowing breaking on a block of addresses. Only address bits with the corresponding mask bit set to '1' are compared during breakpoint detection. To break on AHB load or store accesses, the LD and/or ST bits should be set.

*Table 330.*0x000050, 0x000058 - ATBBA - AHB trace buffer break address register

31 2	1	0
BADDR[31:2]		R
NR		0
rw		r

Table 330. 0x000050, 0x000058 - ATBBA - AHB trace buffer break address register

31: 2	Break point address (BADDR) - Bits 31:2 of breakpoint address
1: 0	Read as 0b00

Table 331. 0x000054, 0x00005C - ATBBM - AHB trace buffer break mask register

31 2	1	0
BMASK[31:2]	LD	ST
NR	0	0
rw	rw	rw

31: 2	Breakpoint mask (BMASK) - (see text)
1	Load (LD) - Break on data load address
0	Store (ST) - Break on data store address

33.6.12 Instruction trace control register 0

The instruction trace control register 0 contains a pointer that indicates the next line of the instruction trace buffer to be written.

Table 332. 0x110000 - ITBCO - Instruction trace control register 0

31 28	27 16	15 0
TFILT	RESERVED	ITPOINTER
0	0	0
rw	r	rw

31: 28	Trace filter configuration
27: 16	RESERVED
15: 0	Instruction trace pointer (ITPOINTER) - Note that the number of bits actually implemented depends on the size of the trace buffer

33.6.13 Instruction trace control register 1

The instruction trace control register 1 contains settings used for trace buffer overflow detection, in addition it includes settings used for some of the instruction trace buffer filtering options. This register can be written while the processor is running.

Bits [31:28] is used to enable or disable Instruction Trace Buffer Address based Filtering (ITBAF). ITBAF is intended to allow the available hardware watch-point (HWP) registers to be used as instruction trace buffer filters when they are not used for breakpoint operation. If a bit is set to '1' in ITBAF, the corresponding address and mask information in the HWP register will be used to filter instruction trace entries based on the program counter (PC) value. Bits[31:28] corresponds to HWP[3:0] respectively. ITBAF can only be used if the corresponding HWP register exist in the hardware. Instruction Trace Buffer Address based Filtering Option (ITBAFO, Bits[19:16]) determines the type of filtering for the corresponding ITBAF entry. If an ITBAFO entry is set to '0' only the PC value(s) that match the address and mask option in the corresponding HWP register will be logged in the instruction trace buffer (ITB). If a bit is set to '1' only the PC value(s) that does not match the address and mask option in the corresponding HWP register will be logged in the ITB. Bits[19:16] corresponds to the option for ITBAF[3:0] respectively. If there is more than one address filtering operation is enabled, the corresponding filtering operations will be combined together.

Bits[15:0] corresponds to ASI last digit based filtering mask (ASIFMASK). ASIFMASK is in effect when the trace filter configuration is set to 0xE (SPARC Format 3 LOAD or STORE instructions to alternate space 0x80 - 0xFF with ASI last digit base filtering). Bits[15:0] corresponds to digits [0xF:0x0] respectively. If a bit is set to '0' in the ASIFMASK, the load and store instructions which have an ASI between the range of 0x80-0xFF and have the corresponding last digit are logged in the instruction trace buffer. For example if only the bit0 and bit2 of the ASIFMASK is set then only the load and store instructions with ASIs 0x80, 0x82, 0x90, 0x92, 0xA0, 0xA2, 0xB0, 0xB2, 0xC0, 0xC2,

0xD0, 0xD2, 0xE0, 0xE2, 0xF0, 0xF2 are tracked. After the reset of processor all the bits in the ASIFMASK is set to 0x0000 which means by default all the ASIs in the range of 0x80-0xFF are tracked.

*Table 333.*0x110004 - ITBCI - Instruction trace control register 1

31 28	27	26 24	23	22 20	19 16	15 0
ITBAF	WO	TLIM	OV	RESERVED	ITBAFO	ASIFMASK
0	0	0	0	0	0	0
rw	rw	rw	rw	r	rw	rw

31: 28 Instruction Trace Buffer Address based Filtering (ITBAF) (see text)

27 Watchpoint on overflow (WO) - If this bit is set, and Break on iu watchpoint (BW) is enabled in the DSU control register, then a watchpoint will be inserted when a trace overflow is detected (TOV field in this register gets set).

26: 24 Trace Limit (TLIM) - TLIM is compared with the top bits of ITPOINTER in Instruction trace control register 0 to generate the value in the TOV field below.

23 Trace Overflow (TOV) - Gets set to '1' when the DSU detects that TLIM equals the top three bits of ITPOINTER.

22: 20 RESERVED

19: 16 Instruction Trace Buffer Address based Filtering Option (ITBAFO) (see text)

15: 0 ASI last digit based filtering mask (ASIFMASK) (see text)

33.6.14 Instruction count register

The DSU contains an instruction count register to allow profiling of application, or generation of debug mode after a certain clocks or instructions. The instruction count register consists of a 29-bit down-counter, which is decremented on either each clock (IC=0) or on each executed instruction (IC=1). In profiling mode (PE=1), the counter will set to all ones after an underflow without generating a processor break. In this mode, the counter can be periodically polled and statistics can be formed on CPI (clocks per instructions). In non-profiling mode (PE=0), the processor will be put in debug mode when the counter underflows. This allows a debug tool such as GRMON to execute a defined number of instructions, or for a defined number of clocks.

*Table 334.*0x000070 - ICNT - Instruction count register

31	30	29	28 0
CE	IC	PE	ICOUNT[28:0]
0	0	0	NR
rw	rw	rw	rw

31 Counter Enable (CE) - Counter enable

30 Instruction Count (IC) - Instruction (1) or clock (0) counting

29 Profiling Enable (PE) - Profiling enable

28: 0 Instruction count (ICOUNT) - Instruction count

33.6.15 AHB watchpoint control register

The DSU has two AHB watchpoints that can be used to freeze the AHB tracebuffer, or put the processor in debug mode, when a specified data pattern occurs on the AMBA bus. These watchpoints can also be coupled with the two AHB breakpoints so that a watchpoint will not trigger unless the AHB breakpoint is triggered. This also means that when a watchpoint is coupled with an AHB breakpoint,

the breakpoint will not cause an AHB tracebuffer freeze, or put the processor(s), in debug mode unless also the watchpoint is triggered.

*Table 335.*0x000080 - AHBWPC - AHB watchpoint control register

31	7	6	5	4	3	2	1	0
RESERVED		IN	CP	EN	R	IN	CP	EN
0		0	0	0	0	0	0	0
r		rw	rw	rw	r	rw	rw	rw

31: 7 RESERVED

6 Invert (IN) - Invert AHB watchpoint 2. If this bit is set the watchpoint will trigger if data on the AHB bus does NOT match the specified data pattern (typically only usable if the watchpoint has been coupled with an address by setting the CP field).

5 Couple (CP) - Couple AHB watchpoint 2 with AHB breakpoint 2

4 Enable (EN) - Enable AHB watchpoint 2

3 RESERVED

2 Invert (IN) - Invert AHB watchpoint 1. If this bit is set the watchpoint will trigger if data on the AHB bus does NOT match the specified data pattern (typically only usable if the watchpoint has been coupled with an address by setting the CP field).

1 Couple (CP) - Couple AHB watchpoint 1 with AHB breakpoint 1

0 Enable (EN) - Enable AHB watchpoint 1

33.6.16 AHB watchpoint data and mask registers

The AHB watchpoint data and mask registers specify the data pattern for an AHB watchpoint. A watchpoint hit is used to freeze the trace buffer by automatically clearing the enable bit. A watchpoint hit can also be used to force the processor(s) to debug mode.

A mask register is associated with each data register. Only data bits with the corresponding mask bit set to '1' are compared during watchpoint detection.

*Table 336.*0x000040 to 0x00004C - 0x0000B0 to 0x0000BC - AHBWPPO-7 - AHB watchpoint data register

31	0
DATA[127-n*32 : 96-n*32]	
NR	
rw	

31: 0 AHB watchpoint data (DATA) - Specifies the data pattern of one word for an AHB watchpoint. The lower part of the register address specifies with part of the bus that the register value will be compared against: Offset 0x0 specifies the data value for AHB bus bits 127:96, 0x4 for bits 95:64, 0x8 for 63:32 and offset 0xC for bits 31:0.

*Table 337.*0x0000A0 - 0x0000AC - 0x0000C0 to 0x0000CC - AHBWPMO-7 - AHB watchpoint mask register

31	0
MASK[127-n*32 : 96-n*32]	
NR	
rw	

31: 0 AHB watchpoint mask (MASK) - Specifies the mask to select bits for comparison out of one word for an AHB watchpoint. The lower part of the register address specifies with part of the bus that the register value will be compared against: Offset 0x0 specifies the data value for AHB bus bits 127:96, 0x4 for bits 95:64, 0x8 for 63:32 and offset 0xC for bits 31:0.

In a system with 64-bit bus width only half of the data and mask registers must be written. For AHB watchpoint 1, a data value with 64-bits would be written to the AHB watchpoint data registers at offsets 0x98 and 0x9C. The corresponding mask bits would be set in mask registers at offsets 0xA8 and 0xAC.

In most GRLIB systems with wide AMBA buses, the data for an access size that is less than the full bus width will be replicated over the full bus. For instance, a 32-bit write access from a LEON processor on a 64-bit bus will place the same data on bus bits 64:32 and 31:0.

33.7 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x017. For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

33.8 Implementation

33.8.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_async_reset_enable* is set.

33.8.2 Technology mapping

DSU4 has one technology mapping generic, *tech*. This generic controls the implementation of which technology that will be used to implement the trace buffer memories. The AHB trace buffer will use two identical SYNCRAM64 blocks to implement the buffer memory (SYNCRAM64 may then result in two 32-bit wide memories on the target technology, depending on the technology map), with one additional 32-bit wide SYNCRAM if the system's AMBA data bus width is 64-bits, and also one additional 64-bit wide SYNCRAM if the system's AMBA data bus width exceeds 64 bits.

The depth of the RAMs depends on the KBYTES generic. If KBYTES = 1 (1 Kbyte), then the depth will be 64. If KBYTES = 2, then the RAM depth will be 128 and so on.

33.9 Configuration options

Table 338 shows the configuration options of the core (VHDL generics).

*Table 338.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	0 - NAHBSLV-1	0
haddr	AHB slave address (AHB[31:20])	0 - 16#FFF#	16#900#
hmask	AHB slave address mask	0 - 16#FFF#	16#F00#
ncpu	Number of attached processors	1 - 16	1
tbits	Number of bits in the time tag counter	2 - 63	30
tech	Memory technology for trace buffer RAM	0 - NTECH-1	0 (inferred)
kbytes	Size of trace buffer memory in KiB. A value of 0 will disable the trace buffer function.	0 - 64	0 (disabled)
clk2x	Enable LEON4 double-clocking (generic is only available on dsu4_2x and dsu4x entities, see next section)	0 - 1	0
bwidth	Traced AHB bus width	32, 64, 128	64
ahbpf	AHB performance counters and filtering. If *ahbpf* is non-zero the core will support AHB trace buffer filtering. If *ahbpf* is larger than 1 then the core's statistical outputs will be enabled.	0 - 2	0
ahbwp	AHB watchpoint enable. If *ahbwp* is non-zero (default) then the core will support AHB watchpoints (also referred to as AHB data breakpoints). Pipeline registers will be added when *ahbwp* is set to 2 (default value), one register for each bit on the AMBA data bus. This setting is recommended in order to improve timing but has a cost in area. The pipeline registers will also lead to the AHB watchpoint being triggered one cycle later. It is recommended to leave this functionality enabled. However, the added logic can create critical timing paths from the AMBA data vectors and so AHB watchpoints can be completely disabled by setting this generic to 0.	0 - 2	2
scantest	Scan test support enable	0 - 1	0
pipedbg	Add pipeline registers on signals from LEON4. If critical timing paths show between, or through, the DSU4 and LEON4 then this value can be set to 1 to add pipeline registers on the dbgi input vector. This adds one additional wait state on some DSU register accesses.	0 - 1	0
pipeahbt	Add pipeline registers on AMBA signals to AHB trace buffer. If there are critical timing paths between the AMBA AHB bus and the DSU AHB trace buffer memory then this value can be set to 1 to add one stage of pipelining between the AHB bus and the trace buffer RAM.	0 - 1	0

33.10 Signal descriptions

Table 339 shows the interface signals of the core (VHDL ports). There are several top-level entities available for the DSU4. The dsu4x entity contains all signals and settings. The other entities are wrappers around dsu4x. The available entities are:

- dsu4 - Entity without support for double clocking. AHB trace of same bus as DSU AHB slave interface is connected to.
- dsu4_2x - Entity with support for LEON4 double-clocking. AHB trace of same bus as DSU AHB slave interface is connected to.
- dsu4_mb - Entity with support for AHB tracing of separate bus
- dsu4x - Entity with support for all features (tracing of separate bus and LEON4 double-clocking).

Table 339. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock, used in dsu4_mb and dsu4 entities	-
HCLK	N/A	Input	Bus clock, used in dsu4_2x and dsu4x entites, only used when double-clocking is enabled	-
CPUCLK	N/A	Input	CPU clock, used in dsu4x and dsu4_2x entities	-
FCPUCLK	N/A	Input	Free running (never gated) CPU clock, only used on dsu4x entity.	-
AHBMI	*	Input	AHB master input signals, used for AHB tracing	-
AHBSI	*	Input	AHB slave input signals, used for AHB tracing when using dsu4 and dsu4_2x entities	-
AHBSO	*	Output	AHB slave output signals	-
TAHBSI	*	Input	AHB slave input signals, used for AHB tracing when using dsu4_mb and dsu4x entities.	-
DBGI	-	Input	Debug signals from LEON4	-
DBGO	-	Output	Debug signals to LEON4	-
DSUI	ENABLE	Input	DSU enable	High
	BREAK	Input	DSU break	High
DSUO	ACTIVE	Output	Debug mode	High
	PWD[n-1 : 0]	Output	Clock gating enable for processor [n]	High
	ASTAT (record)	Output	AHB statistic/performance counter events	-
HCLKEN	N/A	Input	Double-clocking qualifier signal. Only used with double-clocking on dsu4_2x and dsu4x entities	High

* see GRLIB IP Library User's Manual

33.11 Signal definitions and reset values

The signals and their reset values are described in table 340.

Table 340. Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
dsuen	Input	DSU enable	High	-
dsubre	Input	DSU break	High	-
dsuact	Output	Debug mode	High	Logical 0

33.12 Timing

The timing waveforms and timing parameters are shown in figure 70 and are defined in table 341.

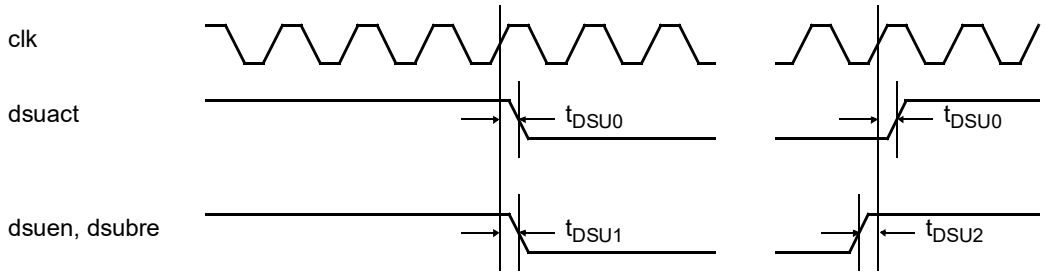


Figure 70. Timing waveforms

*Table 341.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{DSU0}	clock to output delay	rising *clk* edge	TBD	TBD	ns
t_{DSU1}	input to clock hold	rising *clk* edge	-	-	ns
t_{DSU2}	input to clock setup	rising *clk* edge	-	-	ns

Note: The *dsubre* and *dsuen* are re-synchronized internally. These signals do not have to meet any setup or hold requirements.

33.13 Library dependencies

Table 342 shows libraries used when instantiating the core (VHDL libraries).

*Table 342.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	LEON4	Component, signals	Component declaration, signals declaration

33.14 Component declaration

The core has the following component declaration.

```
component dsu4
  generic (
    hindex : integer := 0;
    haddr : integer := 16#900#;
    hmask : integer := 16#f00#;
    ncpu    : integer := 1;
    tbits   : integer := 30;
    tech    : integer := 0;
    irq     : integer := 0;
    kbytes  : integer := 0
  );
  port (
    rst    : in  std_ulogic;
    clk    : in  std_ulogic;
    ahbmi  : in  ahb_mst_in_type;
    ahbsi  : in  ahb_slv_in_type;
    ahbso  : out ahb_slv_out_type;
    dbgi   : in l4_debug_out_vector(0 to NCPU-1);
    dbgo   : out l4_debug_in_vector(0 to NCPU-1);
    dsui   : in dsu4_in_type;
    dsuo   : out dsu4_out_type
```

```
  );
  end component;
```

33.15 Instantiation

This example shows how the core can be instantiated.

The DSU is always instantiated with at least one LEON4 processor. It is suitable to use a generate loop for the instantiation of the processors and DSU and showed below.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.leon4.all;

constant NCPU : integer := 1; -- select number of processors

signal leon4i : l4_in_vector(0 to NCPU-1);
signal leon4o : l4_out_vector(0 to NCPU-1);
signal irqi   : irq_in_vector(0 to NCPU-1);
signal irqo   : irq_out_vector(0 to NCPU-1);

signal dbgi : l4_debug_in_vector(0 to NCPU-1);
signal dbgo : l4_debug_out_vector(0 to NCPU-1);

signal dsui   : dsu4_in_type;
signal dsuo   : dsu4_out_type;

.
begin

cpu : for i in 0 to NCPU-1 generate
    u0 : leon4s                    -- LEON4 processor
    generic map (ahbndx => i, fabtech => FABTECH, memtech => MEMTECH)
    port map (clkm, rstn, ahbmi, ahbmo(i), ahbsi, ahbsi, ahbso,
 irqi(i), irqo(i), dbgi(i), dbgo(i));
    irqi(i) <= leon4o(i).irq; leon4i(i).irq <= irqo(i);
end generate;

dsu0 : dsu4            -- LEON4 Debug Support Unit
    generic map (ahbndx => 2, ncpu => NCPU, tech => memtech, kbytes => 2)
    port map (rstn, clkm, ahbmi, ahbsi, ahbso(2), dbgo, dbgi, dsui, dsuo);
dsui.enable <= dsuen; dsui.break <= dsubre; dsuact <= dsuo.active;
```

34 FTADDR - Autonomous DDR2/DDR3/DDR4 Controller with EDAC

34.1 Overview

FTADDR is a memory controller for DDR2, DDR3 and DDR4 type of SDRAM memory devices. On the memory side, it presents a DFI interface for connection to an on-chip physical layer (PHY) that manages the low-level timing and data recovery and then provides the I/O buffers. Towards the system-on-chip, it presents the memory through an AMBA AHB slave interface.

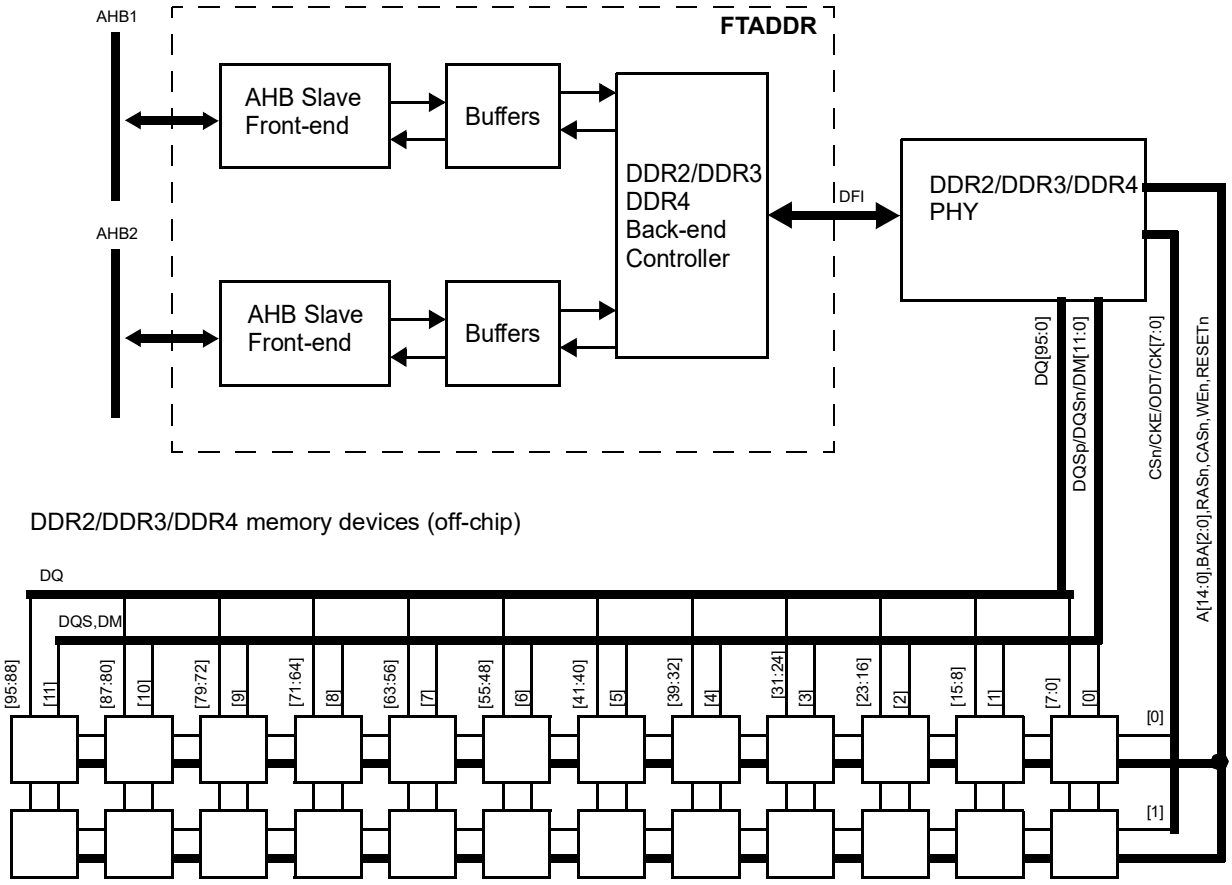


Figure 71. Block diagram of FTADDR Memory controller with two ports and two external banks of 96 bit wide DDR2/DDR3/DDR4 x8 RAM devices.

Internally, the controller is composed of one part clocked by the AMBA clock which manages the user interface, one part clocked by the DFI clock to manage DDR command scheduling and initialization, and FIFOs to communicate between these two parts.

The controller is designed to interface 96-bit wide banks of memory, made out of individual memory devices of width x8 or x4, and then uses a strong error correction code to achieve double-device correction capability. This allows it to deliver correct data despite one full device failure and random SEU-induced errors on the other devices. Up to 8 parallel banks (chip selects) are supported by the controller. Reduced configurations are possible for reduced pin count.

The controller can be used in a multi-ported configuration to support concurrent accesses to different memory banks.

The controller provides configuration registers accessed through a separate address area in the AHB slave. The controller is designed to support processor-less configurations and therefore can operate autonomously with the desired configuration settings supplied as input data at system reset. An interface for direct control of DDR commands and diagnostic reading of data and checkbits is also provided.

34.2 Operation

34.2.1 Memory bank configurations

Single DDR2/DDR3 SDRAM chips are typically 4, 8 or 16 data bits wide. By putting multiple identical chips side by side, wider SDRAM memory banks can be built. Since the command signals are common for all chips, the memories behave as one single wide memory chip, called an external bank or rank. This memory controller supports up to eight identical such memory banks.

Five error detection/correction configurations are supported by the controller: EDAC, SECDED, TMR/5MR, parity and none. The controller can be set into different width settings, full width, half width and narrow mode. The controller can interface banks of x8 or x4 devices. This combined creates 12 possible configurations, listed below.

Devices of width x16 behave exactly as two x8 devices in terms of protocol and is therefore functionally compatible with the controller, however as one device error may affect up to 16 bits, it is not compatible with the EDAC schemes employed by the controller.

Note that the x4 mode is intended for use with PHYs designed with 8 data bits per DQS pair using only the lowest nibble of each byte lane, if the PHY can manage 4 data bits per DQS and present that as a full DFI data bus then the regular x8 mode can be used.

Table 343. List of memory bank configurations supported by the FTADDR Memory controller

Configuration #	Controller setting			External bank layout		# DQ bits	Device-wide error correction	
	enx4	eccen	hwidth	Device type	# of devices	Data+Check	# Corrected	# Detected
1	0 (x8)	11 (80/72)	00	x8	10	64+16	1	1
2			01		9	64+8	0*	0*
3		10 (EDAC)	00		12	64+32	2	2
4			01		6	32+16	1	1
5		01 (Parity)	00		9	64+8	0	1
6			01		5	32+8	0	1
7		00 (None)	00		8	64+0	0	0
8			01		4	32+0	0	0
9	0 (x8)	11 (5MR)	10	x8	N/A	N/A	N/A	N/A
10			11		5	8+32	2	2
11		10 (TMR)	10		12	N/A	N/A	N/A
12			11		3	8+16	1	1
13		01	10		N/A	N/A	N/A	N/A
14			11		N/A	N/A	N/A	N/A
15		00 (None)	10		2	16+0	0	0
16			11		1	8+0	0	0
17	1 (x4)	11	00	x4	10	N/A	N/A	N/A
18			01		9	N/A	N/A	N/A
19		10 (EDAC)	00		12	32+16	2	2
20			01		6	16+8	1	1
21		01 (Parity)	00		9	32+4	0	1
22			01		5	16+4	0	1
23		00 (None)	00		8	32+0	0	0
24			01		4	16+0	0	0

Table 343. List of memory bank configurations supported by the FTADDR Memory controller

25	1 (x4)	11 (5MR)	10	x4	10	N/A	N/A	N/A
26			11		10	8+32	2	2
27		10 (TMR)	10		12	N/A	N/A	N/A
28			11		6	8+16	1	1
29		01 (Parity)	10		9	N/A	N/A	N/A
30			11		5	N/A	N/A	N/A
31		00 (None)	10		4	16+0	0	0
32			11		2	8+0	0	0

34.2.2 Configurable timing parameters

To provide optimum access cycles for different DDR2/3/4 devices (and at different frequencies), different timing parameters can be programmed through the memory configuration registers. The value of these fields affects the memory timing as described in table 344. Note that if the CAS latency setting is changed after initialization, this change needs also to be programmed into the memory chips by executing the Load Mode Register command.

*Table 344.*FTADDR programmable DDR2/DDR3 command timings

DDR2/DDR3 SDRAM timing parameter	Register fields used	Minimum timing (clocks) as function of field values	Formula for field value from timing parameters
CAS latency, CL	DDRT, CASLAT	DDR2: CASLAT DDR3: CASLAT+4 DDR4: CASLAT+9	DDR2: CASLAT=CL DDR3: CASLAT=CL-4 DDR4: CASLAT=CL-9
CAS write latency, CWL (for DDR2, CWL is fixed to (CL-1))	DDRT, CASLAT, WLAT	DDR2: CASLAT-1 DDR3: WLAT+5 DDR4: WLAT+9	DDR2: WLAT=0 (unused) DDR3: WLAT=CWL-5 DDR4: WLAT=CWL-9
Activate to read/write command (t_{RCD})	DRCD	DRCD+2	DRCD=ceil(t_{RCD}/t_{CK})-2
Read to precharge (t_{RTP})[2]	DRTP	DDR2: DRTP+0 DDR3: DRTP+2 DDR4: DRTP+2	DDR2: DRTP=min(ceil(t_{RTP}/t_{CK}), 2) DDR3: DRTP=min(ceil(t_{RTP}/t_{CK})-2, 2) DDR4: DRTP=min(ceil(t_{RTP}/t_{CK})-2, 2)
Write recovery time (t_{WR})	DWR, DDRT, CASLAT, WLAT	DDR2: DWR-CASLAT-1 DDR3: DWR-WLAT-7 DDR4: DWR-WLAT-11	DWR=ceil(t_{WR}/t_{CK})+CWL+2
Precharge to activate (t_{RP})	DRP	DRP+3	DRP=ceil(t_{RP}/t_{CK})-3
Activate to precharge (t_{RAS})	DRAS	DRAS+3	DRAS=ceil(t_{RAS}/t_{CK})-3
Auto-refresh command period (t_{RFC})	DRFC	DRFC+3	DRFC=ceil(t_{RFC}/t_{CK})-3
Activate to activate command period (t_{RRD})	DRRD	DRRD+2	DRRD=ceil(t_{RRD}/t_{CK})-2
Four activate window (t_{FAW})	DFAW	DFAW+DRAS+4	DFAW=max (0, ceil((t_{FAW}-t_{RAS})/t_{CK})-1)
Write to read command delay (t_{WTR})	DWTR, CASLAT, WLAT	DDR2: DWTR-CASLAT-1 DDR3: DWTR-WLAT-7 DDR4: DWTR-WLAT-11	DWTR=ceil(t_{WTR}/t_{CK})+CWL+2
DDR3 mode register set command update delay (t_{MOD})[1]	DMOD	DMOD+3	DMOD=t_{MOD} - 3
CAS to CAS command delay (t_{CCD_L})	DCCD_L	DDR4: DCCDL+4	DCCD=t_{CCD_L} - 4
Long timing parameters (DDR4 Only)	DRRDL DWTRL	These registers only store the differences (deltas) between the long timing and the short timing[3].	DRRDL=DRRD(t_{RRD_L}) - DRRD DWTRL=DWTR(t_{WTR_L}) - DWTR

[1] tMOD also exists with a different definition on DDR2, this does not require the DMOD field to be set

[2] Note that tRTP is defined differently for DDR2 (with BL8) and DDR3, leading to the 2 cycle difference shown here.

[3] The controller adds these deltas to the corresponding short timing values to obtain the effective long timing

If the fields in table 344 are programmed such that the memory specifications are fulfilled, and periodic refresh is correctly setup, then the remaining SDRAM command timing parameters will also be met by design. Table 345 derives example register value for some of the standard speed bins.

*Table 345.*FTADDR timing setup examples based on JEDEC speed bins

Speed bin	DDR2-400B	DDR2-800C	DDR3-800D	DDR3-1600G	DDR4-1600J	
Density / Page size	1 Gb / 2 K	1 Gb / 2 K	2 Gb / 2 K	2 Gb / 2 K	2Gb / 2 K	
Cycle time	5.0	2.5	2.5	1.25	1.25	ns
CL	3	4	5	8	10	nCK
CWL	2	3	5	8	9	nCK
tRCD	15	10	12.5	10	12.50	ns
tRTP	7.5	7.5	10	7.5	7.5	ns
tWR	15	15	15	15	15	ns
tRP	15	12.5	12.5	10	12.50	ns
tRAS	40	45	37.5	35	35	ns
tRFC	127.5	127.5	160	160	160[a]	ns
tRRD	7.5	7.5	10	6	6/7.5[b]	ns
tFAW	50	45	50	40	35	ns
tWTR	7.5	7.5	10	7.5	2.5/7.5[b]	ns
tMOD (DDR3/4)	-	-	12	12	24	nCK
tCCD (DDR4)	4[c]	4[c]	4[c]	4[c]	4/5[b]	nCK
DDRT	0	0	1	1	2	
CASLAT	3	4	1	4	1	
WLAT	0	0	0	3	0	
DRCD	1	2	3	6	8	
DRTP	2	3	2	4	4	
DWR	7	11	13	22	23	
DRP	0	2	2	5	7	
DRAS	5	15	12	25	25	
DRFC	23	48	61	125	125	
DRRD	0	1	2	3	3	
DFAW	1	0	4	3	0	
DWTR	6	8	11	16	13	
DMOD	0	0	9	9	21	
DRRD_L	-	-	-	-	2	
DWTR_L	-	-	-	-	3	
DCCD_L	-	-	-	-	1	
Memory configuration register 1	0x00c0a008	0x0101e081	0x804180a2	0x8133215b	0x004721c3	
Memory configuration register 2	0x0c7210b8	0x10b32180	0x16d23380	0x21646318	0x9B7483e8	
Memory configuration register 4	0x00000000	0x00000000	0x00000009	0x00000009	0x00001695	

[a] timing delay for 1x refresh mode

[b] DDR4 Short (_S) and Long (_L) timing parameters

[c] DDR2/3 always have a t_{CCD} of 4.

34.2.3 Registered SDRAM

Registered memory modules (RDIMM:s) have one cycle extra latency on the control signals due to the external register. They can be supported with this core by setting the REG control register bit. When set this shifts the internal latency used by the controller one cycle relative to the latency programmed to the memory, which compensates for the delay.

This should not be confused with Fully-Buffered DDR2/DDR3 memory, which uses a different protocol and is not supported by this controller.

34.2.4 On-die termination management

At the memory device side, for each chip select the controller can be set up to program different values of on-die termination (Rtt, and for DDR3/4 also Rtt_wr) and output impedance setting into the mode register.

To control the on-die termination in the memory devices (via the ODT control signal), the configuration registers of the controller contain an ODT enable matrix deciding which of the ODT signals are enabled when a specific chip select is being read and written. There is also a default ODT configuration set when neither reading nor writing. By default ODT is always off.

The timing of the ODT signal enabling relative to the read and write command is programmable. The earliest supported switching of the ODT signals are one cycle after the command. For low-latency DDR2 writes this may not be fast enough to switch on the ODT in time, in which case the default ODT configuration should be set to the write configuration instead so that it is set already when the command is given.

A similar configuration is available to enable the controller-side termination on reads, if such termination exist in the PHY.

For DDR3/4 memory, the Rtt setting and ODT enable state desired during write leveling can be selected separately from the setting used during functional mode. Note that the JEDEC standard allows only a subset of the Rtt settings during write leveling.

34.2.5 Setup

The controller is designed to be configurable using two possible means:

- Setting up all configuration register's reset values using external signals, then using the controller directly. The external signals can be either tied to fixed values at higher level in the design, tied to top-level inputs or driven by other logic at the top level of the design
- Configuring parameters over the on-chip bus from processor or other SoC resource

The type of memory (DDR2, DDR3 or DDR4) and the external bank configuration (see section 34.2.1) must be set up before starting up the controller and must remain at the same setting, while other settings can be modified at any time. For correct operation, the timing and size parameters must be set to match the device used, see sections 34.2.2 and 34.2.8.

34.2.6 Memory device initialization and management

Assuming the controller is setup correctly, the controller will manage initialization, refresh and other housekeeping functions. For details on initialization and low-level DDR handling, see section 34.3

34.2.7 Data access

Data accesses to the AHB slave ports are translated to DDR memory accesses by the controller, towards the rest of the system-on-chip the DDR memory will appear as a flat memory space. On the DDR side, all accesses are bursts of length 8, while on the AHB side any length is supported and will be mapped to the necessary number of bursts.

In the multi-ported configuration, each slave port maps to a separate fraction of the available memory. This is done by mapping different ports to different internal banks. Having a fixed mapping all the way from front-end ports minimizes the timing interference between the ports, since accesses on different ports can never map to the same memory row. Accesses can always be done overlapping and therefore the worst case interference becomes only the sum of the maximum data burst length of the other ports rather than one full row open-close cycle for each other port.

For example, with DDR3-800 memory, burst length 8, four ports, the worst case interference becomes 3 ports*(4 cycles burst + 2 cycles turnaround) = 18 DDR cycles = 45 ns, while with a any-to-any mapping the worst case interference would be (all writing to different rows in same internal bank) 3 ports * 25 cycles = 75 cycles = 187.5 ns. The difference gets worse with increasing DDR memory clock rate.

34.2.8 Logical address to memory address mapping

The controller uses a linear address mapping scheme between on-chip bus addresses and external memory addresses. From the least significant bits upward the bits are interpreted as:

- Byte lane on DDR bus. Depending on external bank configuration (see section 34.2.1) the data width is 16, 32 or 64 bits making this field take up the lowest 2-4 bits of the address.
- Column address. Width of this field depends on the number of columns in the external memory devices, between 9-12 bits
- Row address. Width of this field depends on the number of rows in the external memory, between 12-16 bits
- Internal bank. In multi-port configurations the least significant bits of the bank number is implied by which port is accessed and is therefore not included in the address translation. Number of banks can be either 4 or 8 for DDR2, and is always 8 for DDR3. DDR4 has 4 bank groups, and each group contains 4 banks. This leads to between 0-3 address bits.
- External bank/chip select. Up to 8 are supported, leading to 3 address bits for this field.

To achieve a correct address translation without holes or aliases, the AHB address decode register fields need to be set up to match the external memory devices. The algorithm to do this is as follows:

1. Set COLBASE to match the width of the data bus (not including ECC bits).

2. Set ROWBASE so that (9+ROWBASE-COLBASE) matches the number of column address bits on the DDR memories (bits actually used for addressing the column, not counting that A(10) and A(12) are used for auto-precharge and burst-chop indication).

3. Set BANKBASE so that (10+BANKBASE-ROWBASE) matches the number of row address bits on the DDR memories.

4. Set CSBASE so that (2^(CSBASE-BANKBASE)+NPORTS) matches the number of internal banks (4 or 8) or the DDR memories.

5. If the resulting CSBASE value from step 4 became 11 or higher then subtract (CSBASE-10) from both CSBASE and BANKBASE

In DDR4, the address mapping scheme differs slightly from earlier generations. DDR4 devices are organized into 4 bank groups, each containing 4 banks. This requires 2 address bits for the bank group and 2 address bits for the bank. In a 4-port configuration, the most significant bit (MSB) of the bank field is determined by the port, while the least significant bit (LSB) comes from the address. For the bank group field, the MSB is taken from the address, and the LSB is derived from the bank group selection.

As a result, the bank and bank group address mapping is not strictly linear. This non-linear mapping enables bank group interleaving, ensuring that consecutive memory accesses are directed across different bank groups. The primary advantage of this scheme is improved performance, as it mitigates

the impact of longer timing constraints (tCCD_L) associated with repeated accesses to the same bank group.

For x16 DDR4 devices, only two bank groups are available. The corresponding configuration registers in both the backend and the frontend need to be set to support this reduced bank group mapping.

34.3 Back-end functional description

34.3.1 Controller back-end states

From a usage perspective, the DDR side of the controller follows the simplified state diagram shown in figure 72.

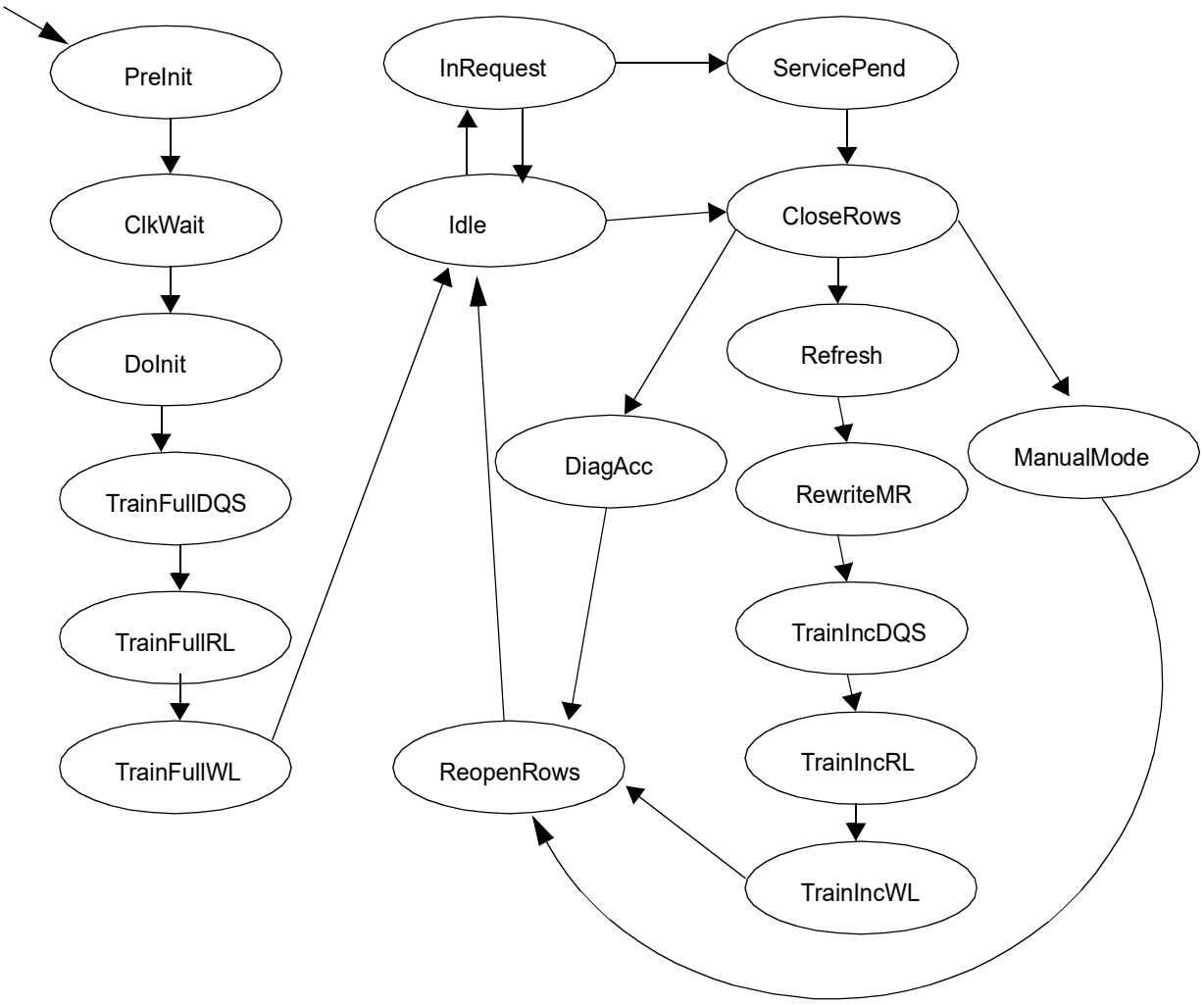


Figure 72. Simplified state diagram of DDR2/DDR3 controller back-end

PreInit - the controller waits for the configuration settings to be set and the pwron bit to be set in the controller. Note that if pwron is set high by the signal interface then this state will be left immediately after reset.

ClkWait - wait until the minimum time has passed from power-on reset before starting the initialization sequence as required by the JEDEC standard. This can be skipped for testing/simulation purposes by writing into the training time counter register while in this state.

DoInit - Initialization sequence according to the selected memory standard.

TrainFullDQS - Training of DQS gate timing

TrainFullRL - Training of read data leveling (DDR3/4 only)

TrainFullWL - Training of write data leveling (DDR3/4 only)

Idle - Waiting for incoming request or service interval timer

InRequest - Serving one or more read/write requests. Can start additional requests to other banks in parallel as decided by internal scheduler and timers.

ServicePend - Serving one or more read/write requests, the service interval timer has expired. New requests are not accepted.

CloseRows - At start of service interval, if any rows are then send a close all rows command to all chip selects. Wait for any tRP delay timers to expire.

Refresh - Send auto-refresh command to all memory devices, if configured to do so at this service interval.

TrainIncDQS - Performing periodic incremental DQS gate delay adjustment, if configured to do so at this service interval.

TrainIncRL - Performing periodic incremental read data leveling (DDR3/4 only), if configured to do so at this service interval.

TrainIncWL - Performing periodic incremental write data leveling (DDR3/4 only), if configured to do so at this service interval.

ReopenRows - Re-open any of the open rows that were closed at the beginning of the service interval. If the controller can see that the next command coming in on a port will result in closing the row (a row close command, or a row open command to a different row), then the row will not be re-opened.

DiagAcc - Perform diagnostic access requested by user.

ManualMode - Performing manual or diagnostic command requested by user (AHB frontend)

34.3.2 Data transfers

The controller contains four bank tracking units to keep track of ongoing data transfers and currently opened rows in the external memory. Each tracking unit has a fixed mapping to a specific set of internal banks in the memory devices, based on the two least significant bits of the internal bank number. At most one opened row at a time can be tracked by each bank tracking unit.

A command scheduler module looks at the state of the bank tracking units, and schedules commands to the DDR command bus as they are ready and can be issued without causing data bus collisions with other ongoing commands. In case more than one bank tracker has a command ready the same cycle, the scheduler will use a round-robin arbiter as a tie-breaker.

The front-end ports are mapped to the bank tracking units in a fixed 1:4, 1:2, or 1:1 mapping depending on how many front-end ports are implemented in the controller, so each front-end port will map to a separate slice of the external memory.

34.3.3 Service intervals

There are several functions in the controller that need to be done periodically, and that require the memory devices to be in idle state so can not be done in parallel with regular data transfers. To handle this efficiently, these are handled in common service intervals. The number of cycles per interval is can be set through configuration registers.

You do not need to do every function at every service interval. For each of the functions (refresh, incremental training, mode register reprogramming), you can configure the frequency as an integer multiple of service intervals. For example, you can set refresh to be done every service interval and incremental training every third interval. Service intervals where nothing is to be done are optimized away so there is no blocking of transfers in that case.

Within each service iteration, all currently opened rows are closed, the pending operations are performed, and the rows are then re-opened to resume operation. To avoid unnecessary re-opening of rows just in order to close them shortly after, the controller peeks at the following command in the command FIFOs, and if that is a row-close or row-open command, the re-opening is skipped for that bank machine. This optimization can be disabled with the FREOP configuration register bit.

It is possible to postpone service iterations using the SHOLD configuration bit. The pending bits and the time counting will still proceed normally. Since there is only a single pending bit for each refresh (or other action), suspending the service iteration for longer than the refresh interval can result in losing a pending refresh command leading to lower refresh rate than expected. Therefore when using the SHOLD function, the user either has to ensure the time is shorter than the refresh interval or the user needs to manually trigger additional refreshes to compensate for the lost refresh commands.

34.3.4 Refresh

The controller contains a refresh function that during a service interval issues an AUTO-REFRESH command to all SDRAM banks. Depending on SDRAM type, the required average period is typically 7.8 us.

34.3.5 ZQ calibration

For DDR3 and DDR4 memories, the controller can be programmed to issue ZQ calibration short (ZQCS) commands periodically at service intervals in order to prevent the input/output drivers on the memory devices from drifting out of specification. ZQ calibration short (ZQCS) and ZQ calibration long (ZQCL) commands can also be manually commanded through the command register. One ZQCL command is always issued during initialization as this is required by the JEDEC standard.

34.3.6 Periodic mode register reprogramming

The controller can be set up to reprogram the memory device's mode registers periodically during service intervals to ensure they are set to correct values.

34.3.7 Delay training

Depending on the PHY design and of the speed of the DDR interface, training of delays may be required. The controller has been designed to support such training.

If the PHY uses MC evaluation mode (DFI mode 01), the training is implemented in a custom sub-block of the controller with an algorithm designed for the specific PHY. For a new PHY that requires this a new phyimpl constant must be defined to the controller which instantiates the new sub-block.

In other training modes (PHY evaluation, PHY independent, and None), the calibration is implemented inside the PHY and the controller only provides some supporting services. This is supported in the generic DFI implementation of the controller without any custom logic.

Incremental training can be done periodically within service intervals to track changes in delays. At each incremental training, a fine adjustment of DQS delays and write leveling is performed. The controller can also be setup to re-do the full training with some interval.

34.3.8 User specified commands

The user can force a refresh or mode register load by writing to command register bits. The controller will schedule the requested command as soon as possible, letting any pending data transfers complete, and ensure the memory timing parameters are respected.

34.3.9 Diagnostic access

The controller has a diagnostic interface where you can command a read or write from a specific location via the register interface. The diagnostic interface can be used safely while the core is running normally.

The procedure for performing a diagnostic access is:

1. Write to the Diagnostic access control register 1, set the DIAGCS,DIAGBANK,DIAGROW fields with the desired location
2. If it's a write, set up the diagnostic checkbit and data registers with the desired data
3. Write to the Diagnostic access control register 2, DIAGCOL field with the desired column, DODG=1, DGWR=1/0 depending on if it's a read or write
4. Poll the diagnostic access status register until the DDONE field is 1.
5. If it was a read, the diagnostic checkbit and data registers hold the read data

The diagnostic access is handled similar to service iterations, current accesses complete, all rows are closed, the diagnostic access is performed, and rows are then re-opened.

The unit of the diagnostic access is always a 64-bit word with the corresponding checkbits. When reading with EDAC enabled, the EDAC decoder error outputs can be seen in the diagnostic status register after the access, however the data that is stored in the diagnostic data/checkbit registers is always uncorrected.

In half-width mode, each diagnostic access will read or write two columns to get a complete EDAC codeword. The column that is addressed should always be even in this mode. Otherwise the diagnostic register interface behaves exactly the same towards the user.

The diagnostic interface ignores the x4 mode setting, and will handle it like the x8 mode. The user has to ignore the top nibbles of the data and checkbits, and mask the corresponding correctable error bits, when using diagnostic accesses in x4 mode.

34.3.10 Manual mode

A raw interface allows to manually perform arbitrary DDR commands such as mode register write, auto refresh, activate, RAS, read/write, and precharge. Note that these manual commands are not managed by the bank tracker and it's the responsibility of the user to not violate the SDRAM protocol. While manual mode is activated, normal accesses will be blocked. Also, no refresh or other service commands will be performed during that time

To activate the manual mode, the user writes 1 to the MANE configuration register field, and then polls the Backend status register until the back-end has entered the manual mode. To return back to normal mode, write 0 to the MANE register field.

In manual mode, the address lines are continuously driven with the value of the DIAGROW field, the bank address lines are driven with the value of the DIAGBANK field. The chip select lines are normally driven high but can be asserted for one cycle by writing into the MANUAL_CSN field.

It is possible to command a read or write command and transfer data with proper DQS strobe generation. This is done by asserting the MRDC or MWDC fields at the same time as giving the CAS command. The data transfer is handled in a similar way as for a diagnostic access, and will use the diagnostic data and checkbit registers (however the diagnostic status register is not modified).

34.3.11 Frequency ratio operation

Frequency ratio operation is only available in the GRLIB integrated version of the controller, not in the stand-alone version.

Besides interfacing PHYs with a frequency ratio of 1:1 (the DFI frequency matching the external memory clock frequency), the controller can support interfacing PHYs with a frequency ratio of 1:2

("half-rate") or 1:4 ("quarter rate"). The functionality is the same when used in half-rate/quarter-rate configuration and all timing register settings in the controller are still set as if the controller was running at 1:1 rate. The selection of frequency ratio between the controller and PHY is intended to be an internal detail that normally does not need to be a concern for the user.

In half rate configuration, the controller will only issue one command per DFI clock cycle, resulting in one command every two memory clock cycles. Depending on the 2T timing setting, either slot 0 or slot 1 will be used for the command (with slot 0 used in 2T mode to setup the address/control lines one cycle ahead of the command). The command selected on any DFI cycle will be arbitrated between the bank machine in the same way as in full-rate operation (see section 34.3.2).

In quarter-rate operation, the controller can issue up to two commands each cycle, one read/write command and one row-open/precharge command. Depending on the 2T setting, either slots 0 and 2 are used for the two commands, or slots 1 and 3 (with slots 0 and 2 used in 2T mode to setup the address/control lines with chip select deasserted one cycle ahead of the command). If only one command is scheduled, the earlier slot (0 or 1) is always used. If two commands are issued the same cycle, the read/write command is scheduled on the earlier slot (0 or 1)and the row open/close command is scheduled on the later slot (2 or 3). Due to the constraints on slots used for commands in frequency ratio operation, some delays will get "rounded up" to multiples of 2 or 4 compared to when operating in full-rate mode.

34.3.12 PHY-specific support

The controller has a phyimpl configuration option specifying if a specific kind of PHY is used. The back-end adapts the default control signal timing (via the timing masks) and may also adjust to other 'quirks' in the specific PHY. Some PHY implementations also provide dedicated configuration or status registers that can be accessed via the PHY indirect address and data registers, those are listed in table 353.

Currently the PHYs in table 346 are implemented.

*Table 346.*FTADDR back-end PHY implementations supported

Value	Name	PHY-specific registers	Description
0	Generic	No	Generic DFI PHY interface
1	Reserved		
2	Altera UniPhy/AFI interface	No	For Altera FPGAs. Can interface a full-rate, half-rate or quarter-rate PHY generated by the Quartus UniPhy MegaWizard
3	Reserved		
4	Xilinx Ultrascale	Yes	For Xilinx Ultrascale FPGAs, interfaces a Xilinx LogiCORE DDR3/DDR4 IP in Phy-only mode.

34.3.13 Memory reboot support

The controller has features to support resetting and possibly also power cycling of individual byte lanes of memory, in order to recover from permanent SEFI errors. While rebooting one byte lane, the other memories are held in self refresh mode to preserve their contents, and the rebooted byte lane's contents can be recovered using the EDAC.

Using the reboot feature requires special support both on the DDR PHY, system design and on the board level. Instead of a separate RESET and CKE signal per rank, there needs to be a separate RESET and CKE signal per byte slice but shared over all ranks. These are output from the controller on the xdfi_bl_resetn and xdfi_bl_cke signals with the same timings as the regular DFI dfi_reset_n and dfi_cke signals. In addition, there needs to be support for either driving both clk_p and clk_n low, or tristating them, this is activated through the additional xdfi_clk_zero signal. An additional output

signal xdfi_bl_pdn signals when the memory should be power cycled, this needs to propagate out to the board level.

If the reboot support is not required, the added signals can be ignored and the standard DFI signals used instead. Using the reboot function will then not have any useful effect, it will just put the memories into self-refresh mode, and then after some time leave self-refresh mode and go back into normal operation.

34.3.14 Internal assertion error flags

There are internal assertion error flags in the back-end status register, that are set when an internally inconsistent state is detected. These are intended mainly for use as assertions in development (for checking after tests) and for debugging malfunctioning systems. The conditions for these flags should never occur under correct operation, and if one of the conditions have occurred then a full system reset is required to get back to a known good state. A brief description of each flag is provided in table 347 below.

Table 347. Internal assertion error flags for back-end

Bit	Description summary
5	Back-end asserted write into status FIFO when FIFO signaled full
4	Back-end asserted write into prefetch data FIFO when FIFO signaled full.
3	Back-end asserted write into read data FIFO when FIFO signaled full
2	Write command received from front-end but expected amount of write data not in write data FIFO.
1	Read data valid received from PHY when no read data has been requested.
0	Currently unused (always zero)

34.4 Front-end functional description

34.4.1 AHB slave ports

Each port implements an AMBA 2.0 AHB compliant slave interface. The AHB interface can have either 32, 64 or 128 bits width on the data bus, selected through the ahbbits VHDL generic.

The GRLIB version of the controller wraps the AHB interface into the GRLIB AMBA record format and also includes the additional sideband signals used for the GRLIB plug-n-play extension to AMBA.

The memory mapped area supports all types of accesses, however some result in suboptimal performance due to read-modify-write cycles or emulating bursts using single cycle accesses or shorter bursts internally. For optimum performance, the following rules should be followed:

- Accesses should be made using the full width of the AHB data bus as HSIZE.
- Wrapping bursts should not be used.

34.4.2 Response patterns

The slave port will respond with wait states using the HREADY signal whenever needed. Retry and split responses are never used.

ERROR responses are used when accessing outside the available memory range, and also in case of EDAC uncorrectable error. The error responses can be disabled using configuration register bits.

34.4.3 Memory read and write

When reading from the memory, the slave port will give wait states until the data is available and then deliver the data streamed out of the internal FIFO as soon as it is ready. The master should be prepared to handle wait states issued at any time within the burst.

When writing to memory, the controller will normally accept the write (single or burst) without wait-states, however if the command FIFO is full then it will issue wait states.

When EDAC is enabled, writes smaller than the codeword size will result in read-modify-write cycles. These are queued internally and will not cause wait states unless the internal FIFOs are full. If the read part of the RMW cycle detects an uncorrectable error, the memory location will be left unmodified and the error will be logged in the core's register and an interrupt will be issued.

34.4.4 Write buffering

Normally any number of writes up to the capacity of the internal buffer can be accepted, however through a configuration register, a maximum limit can be set on the number of queued writes in the buffer, where further accesses will cause wait states. At a minimum, the limit can be set to allow only one buffered write.

The amount of buffering should not be more than needed to mask the latency through the controller and to cover AHB bus timing jitter (unrelated accesses causing gaps in the write stream). Increasing the buffering beyond this will only result in degrading worst-case read latency without improving throughput.

34.4.5 Read pre-fetching

To better support masters streaming long packets of data without requiring very long burst lengths, the AHB slave port supports a read pre-fetching feature. When a read burst is made, the port will queue up one or more additional number of bursts of the same length to the back-end and store these in a prefetch buffer when the data arrives. When the same master comes back and requests the following data, it will then serve this from the prefetch buffer and instead request an additional read burst after the currently prefetched point.

The controller has separate tracking registers for each master in order to detect when prefetching should be activated. The prefetching is activated when the same master makes three read bursts that are back-to-back in terms of address. A prefetch master mask configuration register allows to limit which masters can activate the pre-fetching logic so that random access masters such as caches can be prevented from activating it.

Only one prefetching stream can be ongoing at the same time on each AHB port. The current prefetch is canceled when the currently prefetching master makes an access which does not match the continued stream, or when a prefetch time-out counter expires without the master coming in and requesting data from the next address in the stream. If a write to the same row as is being prefetched is performed on the same AHB port by any master then the prefetch data is discarded in order to prevent serving out-of-date data.

The prefetching continues only to the end of the row and does not cross across row boundaries. It will need to be re-activated on the following row in the same way as described above.

The number of additional read bursts to prefetch is controlled through a configuration register.

34.4.6 Configuration area

Each slave port provides two separate memory areas, one memory mapped area for regular access and one register area for configuration register access.

In the stand-alone version, the ahb_hsel_reg signal selects if memory or registers are accessed. This signal is only considered when the ahb_hsel signal is also asserted on the same cycle. In the GRLIB version, the two AHB areas to two different BARs of the slave similar to other memory controller IP in the library.

The configuration register area must always only be accessed using 32-bit wide accesses, however both single accesses and bursts are permitted.

Any port can access all of the configuration registers, and the register interface is internally arbitrated between the ports.

34.4.7 Multi-ported configuration

In multi-ported configuration, multiple AHB slave ports will be presented. The slave ports are logically separated so they can be either tied to the same bus, or to different AHB buses.

34.5 Error detection and correction features

34.5.1 Overview

The memory controller can be configured to support bit-error tolerant operation by choosing a suitable external memory configuration (see section 34.2.1). In this mode, the DDR data bus is widened and the extra bits are used to store 32 checkbits corresponding to each 64 bit data word.

When writing, the controller generates the checkbits and stores them along with the data on the added part of the data bus. When reading, the controller will transparently correct any correctable bit errors and provide the corrected data on the AHB bus. An extra corrected error output signal is asserted when a correctable read error occurs, at the same cycle as the corrected data is delivered. This can be connected to an interrupt input or to a memory scrubber.

A scrubber function is built into the core that periodically reads from incrementing addresses and writes the data back in case a correctable error is detected. The rate of the scrubbing is programmable and it can also be disabled. The core can also be configured to automatically write back to memory any correctable error that is detected on read.

When performing a write smaller than the codeword size of 64 bits, the controller will automatically perform a read-correct-modify-write cycle to correctly update the checkbits.

34.5.2 Correctable error handling

The intent is for error correction to be transparent to the application, and the controller delivers corrected data with the same timing as uncorrected data. There is no automatic write-back on correctable errors, the controller instead provides a periodic scrubbing function that should be used to prevent error build-up.

A separate ahb_ce output signal is asserted in parallel with the data, this can be used for handling externally to the controller. An internal register holds the address of the correctable error, and an optional interrupt can also be used if desired by the application. All of these features are optional and the controller will work properly even if they are not used.

34.5.3 Uncorrectable error handling

Uncorrectable errors occur when reading a data and checkbit combination that does not match any possible data with any possible combination of errors. To trigger an uncorrectable error, three bytes or more of the data bus must be incorrect (in the default configuration, see table 343 for other configurations). Note that the design intent of the controller with features such as strong EDAC, scrubbing, and SEFI handling is to make the risk of uncorrectable errors occurring as low as possible.

In case of uncorrectable error, this is by default signaled by giving an AHB error response to the master. This behavior can be disabled by configuration and the core will in that case “respond in form” with the corrupted data. A separate sideband signal for uncorrectable errors is also available for custom handling at the SoC level. An uncorrectable error register is included in the core where software can see where the last (if any) uncorrectable error occurred, for diagnosis.

In case of an uncorrectable error detected during a read-modify-write cycle, the modify-write part is canceled and the original contents of the address (containing error) is left untouched, to be found later by the scrubber or a read access.

34.5.4 EDAC details

The FTADDR core supports the following error detection and correction schemes:

EDAC

The implemented EDAC is based on a Reed-Solomon code implemented over the GF2^4 Galois field, with the capability to correct two symbol errors, one symbol corresponding to four binary bits of data. The code has theoretically 11 data symbols and 4 check symbols, however it is shortened to 8 data symbols and 4 check symbols.

The field generating polynomial for the Galois field is a^ 4+a+1, and each nibble of data in and out of the controller is interpreted as powers of alpha, so {b3,b2,b1,b0} = b3*a^3+b2*a^2+b1*a^1+b0. The Reed-Solomon code in this field is defined by the generating polynomial (x+a^6)(x+a^7)(x+a^8)(x+a^9).

To support correction of byte-wide errors, the code is interleaved by a factor of two, resulting in a code with 2*8 data symbols * 4 bit = 64 data bits and 2*4 check symbols * 4 bits/symbol = 32 check bits. The interleaved code can then correct two different byte-wide errors in the data.

Interleaved Reed-Solomon (80-bit)

For the 80-bit mode, two interleaved RS(10,8,2) codewords are used. In this scheme, the data stream is split into even- and odd-indexed symbols, which are separately encoded by two RS encoders, enabling the system to correct up to 2 symbol errors, one in each interleaved codewords, even if the errors occur in adjacent nibbles in the original bitstream. Therefore, this mode can correct and detect 1 device-wide error.

SECDED (72-bit)

The Single-Error Correcting, Double-Error Detecting (SECDED) mode supports only 64-bit data blocks. It appends 8 bits of error correction code (ECC), forming a complete 72-bit codeword capable of correcting a single-bit error and detecting up to two-bit errors. Because of this the SECDED mode does not support device-wide error correction.

TMR/5MR

In TMR (Triple Modular Redundancy), each bit is stored three times and resolved by majority voting, expanding an 8-bit word to 24 bits. In 5MR (Five Modular Redundancy), each bit is stored five times with majority voting, expanding an 8-bit word to 40 bits.

34.5.5 Internal self-checking

The external memory EDAC contains an internal self-check function where the data after decoding is re-encoded and compared with the original read-in data and checkbits. If they differ in some other location than one where an error was corrected, then the EDAC will signal an internal consistency error.

The core can be configured to automatically retry the read transfer when this happens. If the retry is disabled, or if there is another consistency error when the retry is made, this will be treated as an uncorrectable error (section 34.5.3) and in addition a consistency error IRQ is signaled.

34.5.6 Diagnostic interface

The controller supports a diagnostic interface where the checkbits and data bits can be read and written separately with the EDAC bypassed to allow error injection. Accesses to the diagnostic area can be freely mixed with normal memory access which simplifies many test scenarios. Further details are in the back-end description, section 34.3.9.

34.5.7 Initialization

When EDAC is used, the memory must be initialized with correct checkbits. An initialization function that clears the entire memory is implemented in the front-end. This can be set up to initialize with a custom pattern, or to initialize each address with the row/column number of that address.

The controller can be set up to perform a read back immediately after the initialization to verify that the expected data is returned.

34.5.8 Scrubbing

The controller has a built-in scrubber implemented in the front-end side. This reads through all of the memory at a programmable rate, and if any correctable errors are detected, then the corrected data is written back to the same address. The scrubber is also used as the basis for the SEFI handling functionality.

34.5.9 Correctable error counting

There are two separate sets of counters that count statistics on correctable errors, the byte lane error counters, and the address error counters. Both of these will be updated whenever the scrubber finds a correctable error. Note that these are only updated by scrubber accesses and not updated when a regular access triggers a correctable error. The reason for that is to avoid large up-counts if the same error location is read multiple times, which could interfere with the statistics and falsely trigger the SEFI detection logic.

34.5.10 Byte lane error counters

The controller has 12 wrapping counters that track the number of correctable errors that the scrubber finds in each byte of the DDR data bus, plus an additional counter called the watermark that tracks the value of the slowest moving counter. These counters are combined to implement a statistical SEFI detection scheme, which triggers when the byte lane with the most correctable errors has 64 errors more than the byte lane with the least amount of correctable errors.

The watermark is implemented simply by starting at zero after reset like the other counters, and each cycle checking if it is equal to any of the 12 byte lane counters. If the watermark is not equal to any of the counters, then the slowest moving counter must have incremented and the watermark is also incremented to track it.

If one byte lane has a much higher frequency of errors than the others, then that byte lane's error counter will eventually wrap around and count up to the watermark value. That is used as an indication that a SEFI has occurred on that byte lane, which is managed by the SEFI handling state machine.

An equivalent scheme would be to have byte lane counters where you subtract one from each counter once all counters are non-zero and trigger when one counter reaches 64. Each counter minus the watermark (modulo 64) can be viewed as such a counter.

Note that these counters only count errors found by the scrubber, in order to guarantee proportional checking of errors across the whole address space, as otherwise repeated reads of one address with a correctable error could quickly increment a specific counter.

34.5.11 SEFI handling

A special state machine is implemented to attempt SEFI recovery when detected by the byte lane error counters. Note that due to the powerful EDAC and other features implemented in the controller, a single SEFI plus additional SEUs in parallel can be handled implicitly by the controller, the purpose of the SEFI recovery features is to achieve a faster repair in case of recoverable SEFI. The SEFI recovery actions can be disabled in the controller if unwanted, and the SEFI detection logic can then be used in a passive mode for diagnostic purposes instead.

When a SEFI is detected using the byte lane error counters, the SEFI handler it will first request a recalibration of the memory, and a re-write of the memory mode registers from the controller back-end. When this is complete, the scrubber will be setup to perform an accelerated scrub run over the whole memory range in order to quickly regenerate the check bits in case the SEFI was temporary.

After the regeneration, the scrubber will again run in the normal pace. If a full cycle (from start address to end address) of the scrubber is performed without the SEFI detection triggering on the byte lane that had the SEFI, then it is considered to be 'healed' from the SEFI. If the SEFI detection again triggered, it is considered to have a permanent SEFI. When that happens, the controller will issue a permanent SEFI interrupt. and can also optionally be configured to disable the output drivers for that byte-lane in the backend. The controller will continue tracking errors in the byte lane and may find it to have healed at a later stage.

34.5.12 PHY-based SEFI detection

The controller supports an optional sideband signal from the PHY indicating that it had a read failure on a byte lane, if available in the implementation then this can be used to trigger the SEFI handling functionality independently of the error counting. A dedicated interrupt also exists for this condition (back-end error signaled by PHY).

34.5.13 Address error counters

A second set of correctable error counters updated by the scrubber are also provided. These are saturating signed counters that will increase or decrease depending on which address the error occurred on. One counter is implemented for each address bit in the memory address space (CS/bank/row/column). If a correctable error is found where the corresponding address bit is 0, that counter is decremented, while if the address bit was 1, that counter is incremented. As long as errors are occurring randomly over the address space, these counters will vary randomly around zero, but if more errors happens when an address bit has one value, the corresponding counter will tend towards its maximum or minimum value.

The main purpose of these counters are for identifying the likely address when a SEFI is detected, and also to collect statistics over time to detect if some parts of memory are weaker than others.

34.6 Front-end to back-end interface implementation

34.6.1 Command FIFO format

Each port of the front-end sends commands to the back-end using a FIFO. The command words of the FIFO have been designed to match closely (but not exactly) with the commands going out to the memory.

The command words containing the following fields:

- Command ID (4-bit tag)
- 3-bit command word, similar to the (RASn,CASn,WEn) signals of the DDR memory interface
- Chip-select ID (number of bits depending on number of CS supported)
- Bank address bits not implied by port ID
- Address bits (16 bits)
- A special read-modify-write flag (1 bit)
- Prefetch flag (1 bit)

34.6.2 Commands

In normal operation, the commands row-open (011), read (101), write (100), and row-close (010) are issued.

The row-open command opens a specific CS, bank and row. Note that each port can only have one row open. If a new row-open command comes in when a port is open, the other open row will first be closed by the back-end.

The read command causes a read burst to be performed by the back-end. A row must have been selected already using the row-open command. The read out data is normally written into the read-data FIFO, unless the RMW flag or the prefetch flag is set. Note that the CS and bank fields are ignored for this command (these fields have special use for precharging, see below)

The write command performs a write burst to memory, analogous to the read command. The write data must have been placed in the write data FIFO before sending this command.

The row close command will close the currently opened row for the port, if no port is opened it will have no effect. Note that since the back-end does not require this between row-open commands, this is only needed when the port is idle to ensure that the row is not kept opened indefinitely. This also gives the front-end responsibility to manage the row-open policy of the controller.

Having a separate command to select/open the row has some benefits for performance. The front-end can be optimized to transfer the CS,bank,row immediately, and other work such as determining the exact column and transferring write data into the FIFO can be done in parallel with the back-end opening the row.

34.6.3 Response FIFO format

The response FIFO provides a response word for each command. This consists of:

- Command ID (4-bit)
- Result code (3-bit)
- Correctable error byte lane mask (12-bit)

The result code can be one of: OK (000), uncorrectable error (001), EDAC consistency error (010), TBD (011,100,101,110), internal fault detected (111)

34.6.4 Read-data FIFO format

The read-data FIFO words consists of:

- Data word 128-bit
- Correctable error mask (48-bit)
- Uncorrectable error indicator (2-bit)
- Last of burst indicator (1 bit)

Note that the data word is always 128 bits regardless of the width setting of the DDR memory, therefore each read burst will generate 2 or 4 words in the FIFO.

Which byte lanes had the correctable errors can be seen in the response word.

34.6.5 Write-data FIFO format

The write-data FIFO words consists of:

- Data word 128-bit
- Data byte valid 16-bit

The data word is always 128 bits wide regardless of the width setting of the DDR memory, therefore each write command must have 2 or 4 words in this FIFO corresponding to the data. The front-end must always pad data to full bursts by adding words with all valid bits set to 0.

34.6.6 Read-modify-write accesses

The controller allows read-modify-write accesses (8/16/32-bit writes) to be queued up internally without stalling the AHB bus. They are performed without doing round-trips up to the AHB front-end which allows them to be done much more efficiently.

Read-modify-write accesses are commanded by:

1. in the command FIFO, put a row-open command to the command FIFO (unless the desired row is already open)

2. in the command FIFO, put a read command for the desired column with the RMW flag bit set to 1. This will make the back-end store the first word of the read burst into an internal buffer.

3. in the write data FIFO, store the partial codeword with corresponding mask as the first word of the write burst, and then pad with additional words with all bytes masked to make a full burst. (this can be done in parallel with steps 1 and 2)

4. in the command FIFO, put a write command to the same column with RMW flag set to 0.

5. once the RMW is completed a result word will be put into the response FIFO

34.6.7 Prefetching

In order to command a prefetch, set the prefetch bit to 1 in the command word. The read out data will not be stored in the read data FIFO but in a separate two-port prefetch buffer RAM. The RAM might hold multiple bursts, CS,Bank and the top bits of the address part of the read command word that are otherwise ignored are used as address bits into the prefetch buffer in this case.

34.6.8 Register interface

The configuration registers in the back-end are accessed using a simple handshaked interface, allowing one write or one read at a time. Only one register interface is implemented and is internally multiplexed in the front-end between the user ports and the SEFI handler state machine.

34.7 Implementation

34.7.1 Code structure

The top level of the GRLIB version of the controller is named ftaddr_gr.

The main part of the controller logic is contained in the entities ftaddr_fe_ahb (AHB-clocked part), ftaddr_be (DFI-clocked part), ftaddr_edac (EDAC sub-block of ftaddr_be) and the package ftaddr_int.

34.7.2 Clocking and reset

The controller takes two clocks as input, one AHB clock used to clock the front-end ports, and one DFI clock used to clock the controller back-end. The inputs and output signals of the core all belong to one of these two clock domains.

The core has two synchronous reset inputs, one for each clock domain. The two should both be asserted for a number of cycles in both domains, long enough to reset each domain and then to allow all synchronization registers to stabilize. After that the resets can be raised in any order.

A dynsync signal is included in the controller that allows skipping a synchronization stage if the clock domains are synchronous to each other and if the FIFOs support this function. Note that this should not change during operation. Note also that using dynsync requires synchronous constraining between the clock domains.

34.7.3 Buffer memories

Each AMBA port has five FIFOs, all implemented using the syncfifo_2p entity from the GRLIB techmap library. Typically a suitable configuration is that the write and prefetch FIFOs are mapped to RAM blocks, and the command, response, and read data FIFOs are mapped to flip flops.

Using the generic fifoinfmask, it can be controlled per FIFO type if the technology of the FIFO is set to the same as the top level, or to inferred. Using the generic fifoftmask, it can be selected whether EDAC is enabled on the FIFO or not.

Table 348. Buffer memories instantiated inside the top-level entity

Buffer	Depth	Width (excluding ECC)	infmask / ftmask mask bit	Tech on syncfifo_2p		FT on syncfifo_2p		fwft
				infmask=0	infmask=1	ftmask=0	ftmask=1	
Command	8	26-31	0	tech	inferred	0	1	2
Response	8	18	1	tech	inferred	0	1	1
Write data	32	144	2	tech	inferred	0	4	1
Read data	4	179	3	tech	inferred	0	4	2
Prefetch data	32	179	4	tech	inferred	0	4	1

The FIFOs must be of the first-word-fall-through type, where data and valid come out on the same cycle, therefore fwft is set on all the FIFO instantiations. Since command and read data FIFO latency will affect the read access latency of the controller, the fwft generic has been set to 2 to activate a minimum-latency FIFO where that is available. The dynsync signal is passed on into the syncfifo_2p instantiation to allow bypassing synchronization when supported.

34.7.4 EDAC pipelining

The EDAC has been implemented with one full cycle for decoding, and one full cycle for encoding. The decoder has been designed to allow automatic retiming by synthesis tools, to use up any slack before or after the decoder stage.

Extra EDAC pipeline stages are currently not supported, as the pipelining currently implemented satisfies the performance requirements. However the controller has been designed to allow for expanding this in the future without major redesign effort.

34.7.5 Clock gating

The IP core is designed with a design style that allows for automatic clock gate insertion by the synthesis tool.

34.7.6 Performance and area

Trial synthesis of the controller was done to C65SPACE using DC Topo with configuration 4 ports, 128 bits AHB, stand-alone version with all features included, black boxed SRAMs. This met timing at 400 MHz frequency on both AHB and DDR domains and produced a standard cell area (not including the SRAM) of 0.46 mm^2, and at 550 MHz frequency with an area of 0.51 mm^2

The controller has also been built with all features including EDAC, for an Altera Stratix4 FPGA with AHB clock 100 MHz and AFI/DDR clock 125 MHz.

34.7.7 Stand-alone version

A separate top level called ftaddr_sa is available for building the controller without the surrounding GRLIB infrastructure. When building the stand-alone top-level together with the pure-RTL FIFO with flip flops, the IP core has no external dependencies, except for the standard IEEE VHDL libraries.

The stand-alone top level instantiates a block called ftaddr_sa_fifo to implement the FIFOs, instead of going through the GRLIB techmap. The ftaddr_sa_fifo entity can either implement a FIFO based on flip flops or on RAM blocks. When RAM blocks are selected then those are instantiated from the ftaddr_sa_techspec_fifobuf entity, which needs to be provided for the target technology.

Guard gates to prevent glitches propagating between the clock domains are implemented with the ftaddr_sa_cdcguard, in turn depending on the ftaddr_sa_nand2 block, which is a two-input NAND model. In order to avoid this from being optimized out, the command “set_boundary_optimization [get_designs ftaddr_sa_nand2] false” can be used in Design Compiler.

An additional wrapper also exists called ftaddr_sawrap, this takes the stand-alone version and gives it the same interface as the GRLIB version. The main purpose is to allow the stand-alone version to be tested inside a test bench designed to use the GRLIB version.

34.8 PHY specific implementation characteristics

34.8.1 Generic DFI implementation

The generic DFI implementation is intended to interface a DFI2.1.1 compliant DDR2/DDR3/DDR4 PHY with the following limitations:

- The PHY must use a 1:1, 1:2, or 1:4 frequency ratio DFI interface
- The PHY does not use “MC evaluation” mode for training. Note that it is impossible to implement a generic MC evaluation calibration algorithm since the meaning of the delays are not defined in DFI. For using MC evaluation mode, a PHY specific implementation is required.
- The DFI timing parameters are compatible. The timing parameters for the controller are listed in table 350.

The trden, twrlat and twrdata timings that the PHY supports are supplied to the controller through VHDL generics (genphy_trden, genphy_twrlat, and genphy_twrdata). The timings can be set either absolute or relative to the CAS latency depending on what the PHY requires so that they will always be correctly set when updating the CAS latency. The timings can also be overridden from software at run time through the PHY timing registers.

The update interfaces (both MC initiated and PHY initiated) are supported by the controller. The MC update request is asserted while waiting for periodic refresh command to finish.

Note that the DFI parity buses are not used, the regular data bus is used also for ECC/parity bits.

The controller offers some optional extra signals that may be used if the PHY supports it, or may be ignored otherwise:

- xdfi_term_en - Usable to enable local termination on the controller side during reads. Active high, default not used (always 0). The timing and usage of this signal is controlled via the ODT internal timing register.
- xdfi_rderr - Indicate that data on a specific byte lane was not received. Tie to all-0 if not used.
- xdfi_softrst - Active high “soft-reset” to reset the state machines of the PHY to idle state while idle.
- xdfi_phyctrl - Generic control signals tied to PHY generic control signal register.

If desired and supported by the PHY, the control and address signal buses may be duplicated through the ctrldup and csdup generics.

Note that for frequency ratio operation, instead of creating additional signals for the different phases, the DFI signals are widened with a factor corresponding to the frequency ratio with the first phase is stored in the lowest numbered bits. If a signal is N bits wide in full-rate operation, bits (N-1):0 will correspond to phase 0, (2N-1):N corresponds to phase 1, (3N-1):2N corresponds to phase 2, and so on.

In frequency ratio operation, the controller will re-align the write data so that it is delivered in the correct phase relative to the command and the wrdata_en signals for the PHY. For read, the controller requires the data to be delivered from the PHY phase aligned to the phases as mandated by DFI, with the first read data on phase 0, second read data item on phase 1, and so on. Since the controller always issues bursts of length 8, the data for each burst will always be delivered with the first data item starting on phase 0.

34.8.2 Altera UniPhy implementation

The Altera implementation is intended for FPGA prototyping. It has been tested on Stratix4 with DDR2 memory.

The PHY is built using the Quartus II “DDR2 SDRAM Controller with UniPHY” MegaWizard flow, with the “Generate PHY only” option. The burst length must be set to 8.

Some specific precautions need to be taken with this PHY:

- The controller should not reprogram the mode registers, and in particular not change the CAS latency. All period mode register reprogramming functions must be disabled, and the CAS latency setting in the controller must be set to the same as set up when building the PHY.
- Accessing a non-implemented rank will result in the PHY locking up. This is avoided by setting up the NCS field properly in the AHB address decode register and enabling out-of-range AHB error.

The interface used is AFI which is very similar to DFI. Two dedicated ports are added to the controller to support this PHY, afi_dqs_burst and afi_wlat.

The connections are listed in table 349.

*Table 349.*Port connections for UNIPHY implementation

FTADDR port	Connection on PHY	Comment
dfi_cs_n	afi_cs_n	
dfi_bank	afi_ba	
dfi_address	afi_addr	Connect lowest bits if afi_addr width lower
dfi_ras_n	afi_ras_n	
dfi_cas_n	afi_cas_n	
dfi_we_n	afi_we_n	
dfi_cke	afi_cke	
dfi_odt	afi_odt	
dfi_reset_n	Not connected	
dfi_wrdata	afi_wdata	
dfi_wrdata_en(0)	afi_wdata_valid	The PHY has one bit per byte of data bus. Connect bit 0 of wrdata_en to all bits on PHY. For half-rate connect bit 0 to lowest half and bit 1 to higher half, and similarly for quearter rate.
dfi_wrdata_mask	afi_dm	
dfi_rddata_en	afi_rdata_en, afi_rdata_en_full	
dfi_rddata	afi_rdata	
dfi_rddata_valid	afi_rdata_valid	
dfi_ctrlupd_ack	Not connected	Feed constant 0 into controller
dfi_phyupd_req	Not connected	Feed constant 0 into controller
dfi_phyupd_type	Not connected	Feed constant “00” into controller
dfi_dram_clk_disable	dram_clk_disable	
dfi_init_complete	afi_cal_success	
dfi_rdlvl_mode	Not connected	Feed constant 0 into controller
dfi_rdlvl_req	Not connected	Feed constant 0 into controller
dfi_rdlvl_gate_mode	Not connected	Feed constant 0 into controller
dfi_rdlvl_gate_req	Not connected	Feed constant 0 into controller
dfi_rdlvl_resp	Not connected	Feed constant 0 into controller
dfi_wrlvl_mode	Not connected	Feed constant 0 into controller
dfi_wrlvl_req	Not connected	Feed constant 0 into controller
dfi_wrlvl_resp	Not connected	Feed constant 0 into controller
xdfi_term_en	Not connected	
xdfi_rderr	Not connected	Feed constant 0 into controller
xdfi_softrst	Not connected	
xdfi_phyctrl	Not connected	
afi_dqs_burst(0)	afi_dqs_burst	The PHY has one bit per byte of data bus. Connect bit 0 of wrdata_en to all bits on PHY. For half-rate connect bit 0 to lowest half and bit 1 to higher half, and similarly for quearter rate.
afi_wlat	afi_wlat	

34.8.3 Xilinx Ultrascale implementation

The Xilinx Ultrascale implementation is intended to interface Xilinx's DDR3/DDR4 PHY that is included with the Vivado tool suite. The PHY operates in 1:4 frequency ratio.

As this PHY is not DFI compliant, several minor custom adaptations have been made in the controller:

- The ordering of the data bits of the read and write data buses and data mask are slightly unusual, the controller will internally shuffle the bits of these buses to match so that they can be directly tied to the PHY.
- The PHY requires every bit of the control signals to be duplicated to DDR format, this is not managed in the controller so it has to be done externally between the controller and PHY.
- The PHY only allows read/write commands in slots 0 or 2, to match up with that requirement that reason the 2T mode is forced to disabled in the controller which will make the regular 1:4 command scheduling (see section 34.3.11) to always use slot 0 for read/write commands.
- The PHY requires some custom signals to indicate when reading and writing that have been added to the controller.
- The PHY does not use a fixed delay between the write command and the write data, instead it uses a signal wrdataen from the PHY to indicate when write data should be sent.

The PHY uses a custom scheme to track voltage/tracking variations which the controller supports. Once every microsecond, the gtrdy signal is asserted. In each time window between gtrdy assertions, there should be at least one read command and three cycles in a row without read commands. To support the read command requirement, the controller has a programmable time threshold where a dummy read is inserted if there has not already been a read since the last gtrdy assertion. Because the read can not always be done immediately, for example due to an ongoing refresh command, the time threshold has to be set high enough to allow such commands to complete. There is a status register bit indicating if the read requirement was missed and a field indicating the lowest number of cycles left observed for a read that can be used for verifying that the threshold setting is high enough.

There is a convenience wrapper for the controller and PHY called ftaddr_xilus_ddr3 together with a script to extract the timings from the generated PHY and set the reset values of the controller to match.

34.9 DFI Spec Sheet

Below is a specification sheet of the controller in the same format as section 7.0 of the DFI specification. (note that version 2.1.1 of the specification has an error where min and max have been swapped).

Table 350. DFI Settings specification for FTADDR controller

Terms	Min	Max	Comment
DFI clock frequency	N/A		Synchronous logic, max frequency depends on implementation technology.
DFI Address width	16	16	
DFI Bank Width	3	3	
DFI Bank Group Width	2	2	Only for DDR4
DFI Control Width	1	1	
DFI Chip Select Width	1	8	Selected via numcs generic
DFI Data Width	64	256	Selected via ddrbits generic (times 2). Since maximum 2*96 bits are used, additional data bits are padded with zero/ignored.
DFI Data Enable Width	1	-	Set via numrwen generic, all bits driven to same value.
DFI Read Data Valid Width	1	-	Set via numrwen generic, only bit 0 is used.
DFI Read Leveling Delay Width	1	-	Set via rdblvlbits generic.
DFI Read Leveling Gate Delay Width	1	-	Set via rdglvlbits generic
DFI Read Leveling MC IF Width	1	-	Set via numrdlvlmc generic
DFI Read Leveling PHY IF Width	1	-	Set via numrdlvlphy generic
DFI Read Leveling Response Width	32	96	Selected via ddrbits generic (always same width as data bus)
DFI Write Leveling Delay Width	1	-	Set via wrlvlbits generic
DFI Write Leveling MC IF Width	1	-	Set via numwrlvlmc generic
DFI Write LEveling PHY IF Width	1	-	Set via numwrlvlphy generic
DFI Write Leveling Response Width	32	96	Selected via ddrbits generic (always same width as data bus)

Table 351. Timing parameter settings for FTADDR controller

Parameter	Min	Max	Comment
t_{ctrl_delay}	0	5	
t_{phy_wrdata}	0	11-tphy_wrlat	Set via genphy_twrdata generic
$t_{phy_wrdelay}$	0	0	Frequency ratio systems only
t_{phy_wrlat}	0	11	Set via genphy_twrlat generic
t_{phy_rdlat}	1	-	Controller waits indefinitely for read data. System level limitations for refresh etc.
t_{rddata_en}	0	11	Set via genphy_trden generic
$t_{ctrlupd_interval}$	500	-	Depends on refresh rate settings, one update offer per refresh.
$t_{ctrlupd_min}$	3	514	Depends on T_{RFC} setting of controller
$t_{ctrlupd_max}$	3	514	Depends on T_{RFC} setting of controller
t_{phyupd_type0}	0	-	
t_{phyupd_type1}	0	-	

Table 351. Timing parameter settings for FTADDR controller

Parameter	Min	Max	Comment
t_{phyupd_type2}	0	-	
t_{phyupd_type3}	0	-	
t_{phyupd_resp}	100	-	System dependent. Controller will in worst case wait for any already issued read or write commands followed by a precharge all to complete.
$t_{dram_clk_disable}$	0	10	
$t_{dram_clk_enable}$	0	10	
$t_{init_complete}$	N/A		dfi_init_start not implemented
t_{init_start}	N/A		dfi_init_start not implemented
$t_{phy_paritylat}$	N/A		dfi_init_start not implemented
t_{rdlvl_dll}	N/A		Only for MC evaluation mode
t_{rdlvl_en}	1	16	
t_{rdlvl_load}	N/A		Only for MC evaluation mode
t_{rdlvl_max}	-	-	Controller waits indefinitely for rdlvl_resp
t_{rdlvl_resp}	100	-	System dependent. Controller will in worst case wait for any already issued read or write commands followed by a precharge all to complete.
$t_{rdlvl_resplat}$	1	15	
t_{rdlvl_rr}	16	16	
t_{wrlvl_dll}	N/A		Only for MC evaluation mode
t_{wrlvl_en}	1	16	
t_{wrlvl_load}	N/A		Only for MC evaluation mode
t_{wrlvl_max}	-	-	Controller waits indefinitely for wrlvl_resp
t_{wrlvl_resp}	100	-	System dependent. Controller will in worst case wait for any already issued read or write commands followed by a precharge all to complete.
$t_{wrlvl_resplat}$	1	14	
t_{wrlvl_ww}	15	15	
t_{lp_resp}	N/A		Low-power interface not implemented
t_{lp_wakeup}	N/A		Low-power interface not implemented

34.10 Registers

The controller provides a 1024 byte memory-mapped register area. This is split into two halves, the first for configuration registers residing in the back-end, and the second for configuration registers residing in the front-end. For forward compatibility, reserved fields should be written either with 0 or with the last read-out value, and reserved registers should not be accessed at all.

Table 352. Configuration registers for FTADDR controller

Register address offset	Name	R/W	Reset value (dynrst=0)	Notes
0x000	Feature set register	R	*	1
0x004	Back-end status register	R	0x00000000	
0x008	Memory configuration register 1	R/W	0x808FFFFF	
0x00C	Memory configuration register 2	R/W	0xFFFFFFC0	
0x010	Memory configuration register 3	R/W	0x00076475	
0x014	Memory configuration register 4	R/W	0x00000009	
0x018	Service configuration register 1	R/W	0x0FF01512	
0x01C	Service configuration register 2	R/W	0x1900FFFF	
0x020	PHY timing register 1	R/W	*	1
0x024	PHY timing register 2	R/W	*	1
0x028	Diagnostic control register 1	R/W	*	2
0x02C	Diagnostic control register 2	R/W	*	2
0x030	Diagnostic status register	R	*	2
0x034	Diagnostic checkbit register	R/W	*	2
0x038	Diagnostic data register 1	R/W	*	2
0x03C	Diagnostic data register 2	R/W	*	2
0x040	ODT configuration register, rank #0	R/W	0x00000000	
0x044	ODT configuration register, rank #1	R/W	0x00000000	3
0x048	ODT configuration register, rank #2	R/W	0x00000000	3
0x04C	ODT configuration register, rank #3	R/W	0x00000000	3
0x050	ODT configuration register, rank #4	R/W	0x00000000	3
0x054	ODT configuration register, rank #5	R/W	0x00000000	3
0x058	ODT configuration register, rank #6	R/W	0x00000000	3
0x05C	ODT configuration register, rank #7	R/W	0x00000000	3
0x060	ODT external timing register	R/W	0x00008006	
0x064	ODT internal timing register	R/W	0x00000000	
0x068	Command register	W	0x00000000	
0x06C	Sleep mode configuration register	R/W	0x00000000	
0x070	Back-end EDAC configuration register	R/W	0x80000000	
0x074	Service time counter register	R/W	0x0FF340FF	4
0x078	PHY indirect address register	R/W	0x00000000	
0x07C	PHY indirect data register	R/W	*	1
0x080	PHY generic control register	R/W	*	1
0x084	Training time counter register	R/W	0xFFFF0200	4
0x088	Back-end FIFO error counter register	R/W	*	2
0x08C-0x1FC	RESERVED			
0x200	AHB address decode register	R/W	0x0000E6AA	
0x204	AHB access configuration register	R/W	0x4200000F	

Table 352. Configuration registers for FTADDR controller

Register address offset	Name	R/W	Reset value (dynrst=0)	Notes
0x208	Prefetch configuration register	R/W	0x03FFFFFF	
0x20C	Scrubber configuration register 1	R/W	0x00000000	
0x210	Scrubber configuration register 2	R/W	0x0000FFFF	
0x214	IRQ pending register	R/W	0x00000000	
0x218	IRQ enable register	R/W	0x00000000	
0x21C	Scrubber UE error register	R/W	*	2
0x220	Scrubber CE byte lane counter register 1	R	0x00000000	
0x224	Scrubber CE byte lane counter register 2	R	0x00000000	
0x228	Scrubber CE byte lane counter register 3	R	0x00000000	
0x22C	Scrubber CE byte lane counter register 4	R	0x00000000	
0x230	Scrubber CE address counter register 1	R	*	2
0x234	Scrubber CE address counter register 2	R	*	2
0x238	Scrubber CE address counter register 3	R	*	2
0x23C	Scrubber CE address counter register 4	R	*	2
0x240	Access CE location register, port #0	R	*	2
0x244	Access UE location register, port #0	R	*	2
0x248	Access CE location register, port #1	R	*	2, 3
0x24C	Access UE location register, port #1	R	*	2, 3
0x250	Access CE location register, port #2	R	*	2, 3
0x254	Access UE location register, port #2	R	*	2, 3
0x258	Access CE location register, port #3	R	*	2, 3
0x25C	Access UE location register, port #3	R	*	2, 3
0x260	Prefetch status register, port #0	R	*	2
0x264	Prefetch status register, port #1	R	*	2, 3
0x268	Prefetch status register, port #2	R	*	2, 3
0x26C	Prefetch status register, port #3	R	*	2, 3
0x270	Prefetch bank/CS register	R	*	2
0x274	Scrubber start address register	R/W	0x00000000	
0x278	Scrubber end address register	R/W	0x73FFFFFF	
0x27C	Front-end FIFO error counter register	R/W	*	2
0x280	Init pattern register 1	R/W	0x00000000	
0x284	Init pattern register 2	R/W	0x00000000	
0x288	Init pattern register 3	R/W	0x00000000	
0x28C	Init pattern register 4	R/W	0x00000000	
0x290	Scrubber position register, port #0	R	0x00000000	
0x294	Scrubber position register, port #1	R	0x00000000	3
0x298	Scrubber position register, port #2	R	0x00000000	3
0x29C	Scrubber position register, port #3	R	0x00000000	3
0x2A0-0x3FC	RESERVED			

Note 1: Reset value depends on IP configuration options

Note 2: Some fields in register are not reset

Note 3: Whether register is implemented depends on IP configuration options (port/CS count)

Note 4: Register starts counting directly after releasing reset, read-out value may be different

In addition to the AHB registers, an additional register space is accessible by the PHY indirect address and data registers. This provides access to PHY-specific registers and their address mappings are tabulated in table 353 below.

*Table 353.*PHY-specific configuration registers for FTADDR controller

Indirect address	Name	R/W	Reset value	Notes
Generic PHY implementation (phyimpl=0)				
0x00-0xFF	RESERVED			
Altera UniPHY implementation (phyimpl=2)				
0x00-0xFF	RESERVED			
Xilinx Ultrascale implementation (phyimpl=4)				
0x00	VT Tracking Configuration register	R/W	*	1, 2
0x01-0xFF	RESERVED			

Note 1: Reset value depends on IP configuration options

Note 2: Some fields in register are not reset

34.10.1 Feature set register

Table 354. 0x000 - FTADDR feature set register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PHYIMPL								RESERVED							
*								0							
r								r							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES	ECCS	PARS	NCSMAX			NPORT		DDRWIDTH							
0	1	1	*			*		*							
r	r	r	r			r		r							

31 : 24	PHY implementation ID (PHYIMPL). See section 34.3.12 for possible values.
23 : 15	Reserved
14	ECC support, set to 1 if external memory EDAC is supported
13	Parity support, set to 1 if parity checking mode is supported
12 : 10	Number of chip selects (external banks) supported, minus one.
	0=1 bank, 1= 2 banks, ... 7 = 8 banks supported
9 : 8	Number of AHB ports on controller, log2 format. 0=1 port, 1=2 ports 2=4 ports, 3=reserved
7 : 0	Width of DDR data bus, in bits.

34.10.2 Backend status register

Table 355. 0x004 - FTADDR backend status register

31	26	25	16
INTERROR		RESERVED	
000000		0	
r		r	

15	5	4	0
RESERVED		BESTATE	
0		*	
r		r	

31 : 26 Internal error. Diagnostic bits set only on data path inconsistency conditions that should ‘never happen’. See section 34.3.14 for further details.

25 : 5 Reserved

4 : 0 Current back-end state

0 = Down, waiting to be enabled

1 = Enabled, waiting for minimum time before beginning init

2 = Performing init sequence

3 = Initial gate training

4 = Initial data eye training

5 = Initial write leveling

6 = Normal state

7 = In service interval

8 = Returning from service interval, reopening rows

9 = Manual mode

10 = Closing rows for entering service interval

11 = Performing auto-refresh

12 = Rewriting mode registers

13 = Incremental gate training

14 = Incremental data eye training

15 = Incremental write leveling

16 = Performing diagnostic access

17 = Performing automatic retry

18 = Memories in self-refresh mode

19 = Performing memory byte lane reboot

20-31 = Reserved (currently unused)

34.10.3 Memory configuration register 1

Table 356. 0x008 - FTADDR memory configuration register 1

31	30	29	28	27	26	25 22	21 20	19 16
DDRT	HWID	ENX4	PWRU	SKIP-CALIB	SKIP-INIT	CASLAT	WLAT (1:0)	DRAS (6:3)
1	0	0	0	0	0	10	00	1111
rw	rw	rw	rw	rw	rw	rw	rw	rw

15 13	12 6	5 3	2 0
DRAS (2:0)	DRP	DFAW	DRRD
111	1111111	111	111
rw	rw	rw	rw

31 DDR memory type (DRRT(0)), LSB (0=DDR2/4, 1=DDR3)

30 Half width mode selection (HWDITH(0)), LSB

29 Half byte mode selection (0=full byte 1= half byte)

28 Power up (PWRU), set to 1 to start init sequence.

27 Skip calibration, set to 1 before (or at the same time as) setting PWRU to skip calibration

26 Skip initalization, set to 1 before (or at the same time as) setting PWRU to skip initialization

25 : 22 CAS latency (CASLAT), programmed into the mode registers of the memory devices

For DDR2: 0-1=Reserved, 2=CL2, 3=CL3, 4=CL4, 5=CL5, 6=CL6, 7-15=Reserved

For DDR3: 0=Reserved, 1=CL5, 2=CL6, 3=CL7, 4=CL8, 5=CL9, ..., 10=CL14, 11-15=Reserved

For DDR4: 0=CL9, 1=CL10, 2=CL11, 3=CL12, 4=CL13, 5=CL14 6=CL15, 7=CL16, 8=CL18, 9=CL20, 10=CL22, 11=CL24, 12=CL23, 13=CL17, 14=CL19, 15=CL21

21 : 20 CAS write latency (WLAT[1:0]), setting has no effect when DDR2 is used

For DDR3: 0=WL5, 1=WL6, 2=WL7, 3=WL8, 4=WL9, 5=WL10, 6=WL11, 7=WL12

For DDR4: 0=WL9, 1=WL10, 2=WL11, 3=WL12, 4=WL14, 5=WL16, 6=WL18, 7=WL20
MSB for this register is stored in Memory Configuration Register 4 (0x014)

19 : 13 Delay setting for tRAS (DRAS)

12 : 6 Delay setting for tRP (DRP)

5 : 3 Delay setting for tFAW (DFAW)

2 : 0 Delay setting for tRRD (DRRD)

34.10.4 Memory configuration register 2

Table 357. 0x00C - FTADDR memory configuration register 2

31	30	29 25	24 20	19 16
DDRT	HWID	DWTR	DWR	DRTP
0	0	11111	11111	1111
rw	rw	rw	rw	rw

15 12	11 3	2	1	0
DRCD	DRFC	DLLDIS	RDIMM	T2
1111	111111111	0	0	0
rw	rw	rw	rw	rw

31 DDR memory type (DRRT(1)), (0=DDR2/3, 1=DDR4)

30 Half width mode selection (HWDITH(1)), MSB

29 : 25 Delay setting for tWTR (DWTR)

24 : 20 Delay setting for tWR (DWR)

19 : 16 Delay setting for tRTP (DRTP)

15 : 12 Delay setting for tRCD (DRCD)

11 : 3 Delay setting for tRFC (DRFC)

2 DLL disable setting for external memories (1=DLL disabled, 0=DLL enabled)

Table 357. 0x00C - FTADDR memory configuration register 2

1	Registered DIMM mode (0=regular memory, 1=registered DIMM)
0	2T signaling mode enable (0=1T signaling, 1=2T signaling)

34.10.5 Memory configuration register 3

Table 358. 0x010 - FTADDR memory configuration register 3

31	30 24	23	22 20	19 16
M3OVR	RESERVED	X16	RESERVED	DRTW
0	0	0	0	*
rw	r	rw	r	rw

15 12	11 8	7 4	3 0
DRTRXCS	DWTWXCS	DRTWXCS	DWTRXCS
0110	0100	*	*
rw	rw	rw	rw

31	Memory configuration register 3 override (M3OVR). This must be set to 1 to configure the other fields in this register, otherwise they are automatically set to default values as indicated below.
30 : 20	Reserved
23	DDR4 Reduced bank-group address mapping (DDR4 x16 only)
19 : 16	Delay between read-to-write commands to same CS, by default set to CL-WL+6
15 : 12	Delay between read-to-read commands to different CS, by default set to 6 cycles
11 : 8	Delay between write-to-write commands to different CS, by default set to 4 cycles
7 : 4	Delay between read-to-write commands to different CS, by default set to CL-WL+6
3 : 0	Delay between write-to-read commands to different CS, by default set to WL-CL+6

34.10.6 Memory configuration register 4

Table 359. 0x014 - FTADDR memory configuration register 4

31 20	19 16
DATA BYTE DISABLE	RESERVED
000000000000	0
rw	r

15 13	12 9	8 6	5	4 0
DRRDL	DWTRL	DCCDL	WLAT (2)	DMOD
0	0	0	0	1001
rw	rw	rw	rw	rw

31 : 20	Data byte disable mask. Set bit to 1 to disable corresponding byte lane (if supported by PHY).
15 : 13	Delay setting for tRRD (DRRDL)
12 : 9	Delay setting for tWTR_L (DWTRL)
8 : 6	Delay setting for tCCD_L (DCCDL)
5	CAS write latency (WLAT(2)). WLAT(1:0) for this register is stored in Memory Configuration Register 1 (0x008). Refer to this register for detailed bit definitions.
4: 0	Delay setting for tMOD (DMOD)

34.10.7 Service configuration register 1

Table 360. 0x018 - FTADDR service configuration register 1

31	30	29	28	27	20	19	16
SHOLD	RESERVED		FREOP	MRREPIVAL		REFIVAL (7:4)	
0	0		0	11111111		0000	
rw	r		rw	rw		rw	

15	12	11	0
REFIVAL (3:0)		SRVBASEPER	
0001		010100010010	
rw		rw	

31	Hold-off service intervals, set to 1 to delay triggering any refresh or mode reprog cycles
30 : 29	Reserved
28	Force always reopening rows after service interval (see section 34.3.3)
27 : 20	Mode register reprogramming interval, in multiples of base period, minus one Setting this register to all-ones disables mode register reprogramming
19 : 12	Refresh interval, in multiples of base period, minus one Setting this register to all-ones disables periodic refresh cycles
11 : 0	Service interval base period in DFI clock cycles, minus one

34.10.8 Service configuration register 2

Table 361. 0x01C - FTADDR service configuration register 2

31	24	23	16
U100MULT		RESERVED	
00011001		0	
rw		r	

15	8	7	0
FTRAINIVAL		ITRAINIVAL	
11111111		11111111	
rw		rw	

31 : 24	Multiple of service interval base periods needed to get to 100 us, minus one. This counter is used for the training intervals in this register. The value for this register and the base period is also used for timing of init sequence delays.
23 : 16	Reserved
15 : 8	Full re-training interval, in multiples of 10 ms units (100*U100MULT*SRVBASEPER), minus one Setting this register to all-ones disables full re-training
7 : 0	Incremental re-training interval, in multiples of 10 ms units (100*U100MULT*SRVBASEPER), minus one. Setting this register to all-ones disables incremental training

34.10.9 PHY timing register 1

Table 362. 0x020 - FTADDR PHY timing register 1

31	30	16
PTOR	RESERVED	
0	0	
rr	r	

15	12	11	0
RESERVED		RDEN_MASK	
0		*	
r		rw	

31 PHY timing register override (PTOR)

This is set to 1 to allow manual configuration of the fields in PHY timing register 1 and 2, otherwise they are set up automatically.

30 : 12 Reserved

11 : 0 Read-enable timing mask. This controls when the read-enable signal to the PHY (dfi_rddata_enable) is sent out relative to the read command. If bit 0 is set, the read-enable signal is asserted the same time as the command, if bit 1 is set then the read-enable is asserted one cycle after, and so on.

Only one bit in the mask should be set for DFI compliant operation, however it may in some cases be useful for debugging to set multiple bits to stretch the dfi_rddata_enable longer than the actual burst.

By default (if PTOR is not set) this will be set automatically to an expected good value based on CAS latency values and which PHY implementation is used.

34.10.10PHY timing register 2

Table 363. 0x024 - FTADDR PHY timing register 2

31	21	20	16
RESERVED		WRDATA_MASK (8:4)	
0			
r			

15	12	11	9	8	0
WRDATA_MASK (3:0)		RESERVED		WREN_MASK	
*		0		*	
rw		r		rw	

31 : 21 Reserved

20 : 12 Write data timing mask. Controls the timing of delivery of write data to the PHY relative to the write command, analogous to the RDEN_MASK.

By default (if PTOR in PHY timing register 1 is not set), this is set automatically to the expected good value based on CAS/write latency settings and PHY implementation.

11 : 9 Reserved

8 : 0 Write enable mask. Controls the timing of the dfi_wrdata_en signal to the PHY relative to the write command, analogous to RDEN_MASK.

By default (if PTOR in PHY timing register 1 is not set) this is set automatically to the expected good value based on CAS/write latency settings and PHY implementation.

34.10.11 Diagnostic access control register 1

Table 364. 0x028 - FTADDR diagnostic access control register 1

31	24	23	20	19	18	16
RESERVED		DIAGCS		RES	DIAGBANK	
0		(nr)		0	(nr)	
r		rw		r	rw	

15	0
DIAGROW	
(nr)	
rw	

31 : 24 Reserved

23 : 20 Chip select number for next diagnostic access (DIAGCS)

Actual width of this field depends on number of CS lines implemented.

19 Reserved

18 : 16 Bank number used for next diagnostic access (DIAGBANK)

15 : 0 Row number used for next diagnostic access (DIAGROW)

This register is also used to control the address lines in manual mode.

34.10.12 Diagnostic access control register 2

Table 365. 0x02C - FTADDR diagnostic access control register 2

31	30	29	28	27	26	25	24	23	16
DODG	DGWR	MANE	MRASN	MCASN	MWEN	MRDC	MWRC	MANUAL_CSN	
0	(nr)	0	(nr)	(nr)	(nr)	0	0	11111111	
rw	rw	rw	rw	rw	rw	w	w	rw	

15	14	12	11	0
MGTRD	RESERVED		DIAGCOL	
0	0		(nr)	
w	r		rw	

31 Do diagnostic access (DODG)

Write 1 to this field to start a single diagnostic access corresponding to DIAGCS,DIAG-BANK,DIAGCOL,DGWR fields. Self clearing. Poll DIAGDONE to detect completion.

30 Write/Read control for diagnostic access. Set to 1 to make next diagnostic access a write, set to 0 to make it a read.

29 Manual mode request. Set this field to 1 to request the controller to go into manual mode and clear when done. This is a request signal, poll the Backend status register to see when controller is ready.

28 Manual mode RASN signal control

27 Manual mode CASN signal control

26 Manual mode WEN signal control

25 Manual mode read pipeline activation. Write 1 to this the same time as asserting the manual_csn with a read command to fetch the read data into the diagnostic data register.

24 Manual mode write pipeline activation. Write 1 to this field the same time as asserting the manual_csn with a write command to do a write burst with the data from the diagnostic data register.

23 : 16 Manual chip select assertion. If one or more bits in this field are written then the corresponding chip select lines will be asserted for one cycle. This field is self-clearing and returns immediately to all-ones state.

15 Manual gate training read sequence. This will lower one of the MANUAL_CSN bits (which one is selected by the DIAGCS field) two times with 4 cycles apart. The first time the MRDC bit is also written to one.

14 : 12 Reserved

11 : 0 Column number for diagnostic access.

34.10.13Diagnostic access status register

Table 366. 0x030 - FTADDR diagnostic access status register

31	30 25	24	23 16
DDONE	RESERVED	DGUE	DIAGCEMASK (23:16)
0	0	(nr)	(nr)
r	r	r	r

15 0
DIAGCEMASK (15:0)
(nr)
r

31 Diagnostic access done (DDONE)

Poll this bit after writing DODG, when it is set to 1 the access has completed, diagnostic data registers have been updated and DGUE/DGCEMASK fields are valid.

30 : 25 Reserved

24 Diagnostic access uncorrectable EDAC error detected (DGUE)

23 : 0 Diagnostic access correctable EDAC error mask (DIAGCEMASK)

One bit per nibble read asserted if the EDAC corrected an error on that nibble.

34.10.14Diagnostic checkbit register

Table 367. 0x034 - FTADDR diagnostic checkbit register

31 16
DIAGCB (31:16)
(nr)
rw

15 0
DIAGCB (15:0)
(nr)
rw

31 : 0 Diagnostic checkbit register. Holds the 32-bit checkbit part of the codeword before or after diagnostic access

34.10.15Diagnostic data register 1

Table 368. 0x038 - FTADDR diagnostic data register 1

31 16
DIAGDATA1 (31:16)
(nr)
rw

15 0
DIAGDATA1 (15:0)
(nr)
rw

31 : 0 Diagnostic data register 1. Holds the high half of the 64-bit data part of the codeword before or after diagnostic access

34.10.16Diagnostic data register 2

Table 369. 0x03C - FTADDR diagnostic data register 2

31	16
DIAGDATA2 (31:16)	
(nr)	
rw	

15	0
DIAGDATA2 (15:0)	
(nr)	
rw	

31 : 0 Diagnostic data register 2. Holds the low half of the 64-bit data part of the codeword before or after diagnostic access

34.10.17ODT configuration register for CS #N

Note: 1-8 separate registers with the same format is implemented, one for each chip select

Table 370. 0x040-0x05C, FTADDR ODT configuration register for CS #N

31 27	26	25 24	23 21	20 19	18 17	16
RESERVED	ODTWL	RTTWL	RTT	RTTWR	DIC	RLODT
0	1	01	000	00	00	0
r	rw	rw	rw	rw	rw	rw

15 8	7 0
RDODT	WRODT
00000000	00000000
rw	rw

31 : 27 Reserved

26 Enable ODT during write leveling (DDR3/4 only)

25 : 24 RTT setting during write leveling (DDR3/4 only)

23 : 21 RTT setting for programming into mode register

20 : 19 RTT_WR setting for programming into mode register (DDR3/4 only)

18 : 17 DIC setting for programming into mode register

16 Enable local on-chip termination when reading from this chip select.

15 : 8 Mask of which ODT signals to enable when reading from this chip select

7 : 0 Mask of which ODT signals to enable when writing to this chip select

34.10.18ODT external timing register

Table 371. 0x060 - FTADDR ODT external timing register

31	24	23	13	12	11	1	0
ODTDEFAULT		ODTRDMASK (12:5)	ODTRDMASK (4:1)	RES	ODTWRMASK(12:1)		RES
00000000		00000000	1000	0	00000000110		0
rw		rw	rw	r	rw		r

31 : 24 ODT default value

Mask specifying which ODT signals are enabled when neither read nor write ODT configuration has been activated.

23 : 13 ODT read timing mask.

Bit mask specifying the timing relative to a read command when the read ODT configuration for that chip select (based on the per-CS ODT registers) is activated. This mask is set automatically based on current CAS latency settings and can only be manually set if the PTOR field in PHY timing register 1 is set.

12 Reserved

Controller does not support asserting ODT simultaneously with the read command, therefore bit 0 of the timing mask is not implemented.

11 : 1 ODT write timing mask

Bit mask specifying the timing relative to a write command, when the write ODT configuration for that chip select (based on the per-CS ODT registers) is activated. This mask is set automatically based on current CAS latency settings and can only be manually set if the PTOR field in PHY timing register 1 is set.

0 Reserved

Controller does not support asserting ODT simultaneously with the write command, therefore bit 0 of the ODT write timing mask is not implemented.

34.10.19ODT internal timing register

Table 372. 0x064 - FTADDR ODT internal timing register

31	24	23	16	15	12	11	0
RESERVED		LODTRMASK (11:4)		LODTRMASK (3:0)		RESERVED	
*		00000000		0000		0	
r		rw		rw		r	

31 : 24 Reserved

23 : 12 Local ODT read timing mask.

Bit mask specifying the timing relative to a read command when the internal on-chip termination is activated. This mask is set analogous to the RDEN_MASK.

11 : 0 Reserved

34.10.20Command register

Table 373. 0x068 - FTADDR command register

31										20	19	18	17	16
SET BYTE LANE DISABLE											RES	MREB	DOZQL	DOZQS
0											0	0	0	0
ws											r	ws	ws	ws

15	14	11	10	9	8	7	6	5	4	3	2	1	0
SPRST	TRLANE		BLTR	BLTG	BLTW	INCTR	INCTG	INCTW	FTRR	FTRG	FTRW	DOMR	DOREF
0	r0		0	0	0	0	0	0	0	0	0	0	0
ws	rw		ws	ws	ws	ws	ws	ws	ws	ws	ws	ws	ws

31 : 20 Disable byte lane. Writing 1 to one of the bits in this field sets the corresponding bit in the byte lane disable register in memory configuration register 4 (see 34.10.6). Reading returns a copy of the byte lane disable register.

19 Reserved

18 Write 1 to trigger memory reboot of single byte lane selected by bits 14:11, if supported by controller. The read value of this field will be 1 after writing it until the action has been performed.

17 Write 1 to trigger a single DDR3/4 ZQ Calibration long command. The read value of this field will be 1 after writing it until the action has been performed.

16 Write 1 to trigger a single DDR3/4 ZQ Calibration short command. The read value of this field will be 1 after writing it until the action has been performed.

15 Write 1 to this bit to trigger a soft reset of the PHY (if supported). The read value of this field will be 1 after writing it until the action has been performed.

14 : 11 Selects which byte lane to target for byte lane re-training or memory reboot:

0=bits 7:0, 1=bits 15:8, ..., 11 = bits 95:88, 12-15 = reserved

This field is only written when one or more of the bits 10:8 or 18 is also written with 1. The field needs to have its value kept until the re-training and re-boot actions have completed.

10 Write 1 to trigger read data eye training on single byte lane selected by bits 14:11. The read value of this field will be 1 after writing it until the action has been performed.

9 Write 1 to trigger gate training on single byte lane selected by bits 14:11. The read value of this field will be 1 after writing it until the action has been performed.

8 Write 1 to trigger write leveling on single byte lane selected by bits 14:11. The read value of this field will be 1 after writing it until the action has been performed.

7 Write 1 to trigger full read data eye training. The read value of this field will be 1 after writing it until the action has been performed.

6 Write 1 to trigger full gate training. The read value of this field will be 1 after writing it until the action has been performed.

5 Write 1 to trigger full write leveling. The read value of this field will be 1 after writing it until the action has been performed.

4 Write 1 to trigger incremental read data eye training. The read value of this field will be 1 after writing it until the action has been performed.

3 Write 1 to trigger incremental gate training. The read value of this field will be 1 after writing it until the action has been performed.

2 Write 1 to trigger incremental write leveling. The read value of this field will be 1 after writing it until the action has been performed.

1 Write 1 to trigger a mode register reprogramming. The read value of this field will be 1 after writing it until the action has been performed.

0 Write 1 to trigger an auto-refresh. The read value of this field will be 1 after writing it until the action has been performed.

34.10.21Sleep mode configuration register

Table 374. 0x06C - FTADDR sleep mode configuration register

31	16
RESERVED	
0	
r	

15	4	3	1	0
RESERVED		SREFTHRES		ASRFE
0		000		0
r		rw		rw

31 : 4 Reserved

3 : 1 Self refresh threshold (SREFTHRES). Number of idle refresh periods needed in order to trigger going to self-refresh mode, minus one. Only used if SLPEN is set.

0 Automatic self-refresh enable. If set to 1 the controller will automatically put memories in self-refresh mode when idle for a longer time.

34.10.22EDAC configuration register

Table 375. 0x070 - FTADDR backend EDAC and FIFO error counter register

31	30	29	28	27	26	16
ECCMODE		ERET	RDEEN	DRMW	RESERVED	
10		0	0	0	0	
rw		rw	rw	rw	r	

15	4	3	0
RESERVED		RETTEST	
0		0000	
r		rw	

31 : 30 Error Correction Code Mode

Hwidth(1)=0: 11 = 80/72 ECC, 10 = EDAC, 01 = parity, 00 = none

Hwidth(1)=1: 11 = 5MR, 10 = TMR, 01 = parity, 00 = none

29 Automatic retry on EDAC internal errors

28 Enable read-error signal from PHY (if supported by PHY) to trigger byte lane failure messages to the front-end.

27 Disable read-modify-write and perform partially masked writes instead. Should only be used when EDAC is disabled (ECCMODE=00)

26 : 4 Reserved

3 : 0 Retry test. Writing a 1 to bit 3 and a count of 1-7 to bits 2:0 will trigger an EDAC internal error flag once in order to test the automatic retry feature enabled by bit 29. The count determines how many read data words from the PHY are received before the test is triggered. Writing 0 to bit 3 or writing 0 to bits 2:0 will have no effect and the field gets cleared.

34.10.23Service time counter register

This register provides access to the internal running counters of the back-end that trigger the periodic service functions. Writing to this register is mainly intended for testability purposes (for example to produce a specific timing corner case for testing) and is not expected to be used in a typical application.

Table 376. 0x074 - FTADDR service time counter register

31 29	28	27 20	19 16
RESERVED	SITICK	MRREPCTR	REFCTR(7:4)
0	0	00000000	0000
r	rw	rw	rw

15 12	11 0
REFCTR(3:0)	SICTR
0000	000000000000
rw	rw

31 : 29 Reserved

28 Service interval tick, set to 1 when service interval counter reached zero and was reset to the programmed interval.

27 : 20 Mode register reprogramming counter. Decreased when SITICK is set.

19 : 12 Refresh counter. Decreased when SITICK is set.

11 : 0 Service interval counter. Decreased each cycle.

34.10.24PHY indirect address register

Table 377. 0x078 - FTADDR PHY indirect address register

31 20	19 16
PHY READ ERROR INDICATOR	RESERVED
000000000000	0
r	r

15 8	7 0
RESERVED	PHY REGISTER ADDRESS
0	00000000
r	rw

31 : 20 Phy read error status indicator, read only. This indicates for which byte lanes the PHY has signaled a read error. The bits are cleared on reset or when the corresponding byte lane is disabled via the byte lane disable mask.

19 : 8 Reserved

7 : 0 PHY indirect register address. This field controls which PHY-internal register is accessed via the PHY indirect data register

34.10.25PHY indirect data register

Table 378. 0x07C - FTADDR PHY indirect data register

31	16
PHY REGISTER DATA(31:16)	
*	
rw	
15	0
PHY REIGSTER DATA(15:0)	
*	
rw	

31 : 0 PHY register data, read/write. Contents of the register depends on which PHY is implemented and which register has been selected in the PHY indirect address register.

34.10.26PHY generic control register

Table 379. 0x080 - FTADDR PHY generic control register

31	16
PHY GENERIC CONTROL(31:16)	
*	
rw	
15	0
PHY GENERIC CONTROL(15:0)	
*	
rw	

31 : 0 Generic control bus intended to control PHY or system-specific pseudo-static signals.

Number of bits implemented in the controller depends on the *phyctrlbits* generic.

34.10.27Training time counter register

This register provides access to the internal running counters of the back-end that trigger the periodic training functions, and is also used for power-up and initialization sequence delays. Writing to this register is mainly intended for testability purposes (for example to produce a specific timing corner case for testing) and is not expected to be used in a typical application.

Table 380. 0x084 - FTADDR training time counter register

31	24	23	16
FTRAINCTR		ITRAINCTR	
00000000		00000000	
rw		rw	

15	14	8	7	0
TICK10	M10CTR		U100CTR	
0	0000000		00000000	
rw	rw		rw	

31 : 24 Full incremental training counter value, decreased each 10 millisecond tick.

23 : 16 Periodic incremental training counter value. decreased each 10 millisecond tick.

15 10 millisecond tick, set for one cycle when 10 millisecond counter wraps.

14 : 8 10 millisecond counter, decreased when 100 microsecond counter wraps.

7 : 0 100 microsecond counter, decreased when SITICK is set.

34.10.28Back-end FIFO error counter register

Table 381. 0x088 - FTADDR back-end FIFO error counter register

31							24	23	22	21	20	19	18	17	16
RESERVED								CM3UE	CM3CE		WD3UE	WD3CE		CM2UE	CM2CE (1)
0								(nr)	(nr)		(nr)	(nr)		(nr)	(nr)
r								wc	wc		wc	wc		wc	wc
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CM2CE (0)	WD2UE	WD2CE		CM1UE	CM1CE		WD1UE	WD1CE		CM0UE	CM0CE		WD0UE	WD0CE	
(nr)	(nr)	(nr)		(nr)	(nr)		(nr)	(nr)		(nr)	(nr)		(nr)	(nr)	
wc	wc	wc		wc	wc		wc	wc		wc	wc		wc	wc	

31 : 24	Reserved
23	Uncorrectable error occurred on command FIFO for port #3
22 : 21	Correctable error saturating counter for command FIFO for port #3
20	Uncorrectable error occurred on write-data FIFO for port #3
19 :18	Correctable error saturating counter for write-data FIFO for port #3
17	Uncorrectable error occurred on command FIFO for port #2
16 : 15	Correctable error saturating counter for command FIFO for port #2
14	Uncorrectable error occurred on write-data FIFO for port #2
13 : 12	Correctable error saturating counter for write-data FIFO for port #2
11	Uncorrectable error occurred on command FIFO for port #1
10 : 9	Correctable error saturating counter for command FIFO for port #1
8	Uncorrectable error occurred on write-data FIFO for port #1
7 : 6	Correctable error saturating counter for write-data FIFO for port #1
5	Uncorrectable error occurred on command FIFO for port #0
4 : 3	Correctable error saturating counter for command FIFO for port #0
2	Uncorrectable error occurred on write-data FIFO for port #0.
1 : 0	Correctable error saturating counter for write-data FIFO for port #0.

Note that error counters are only implemented if the port is implemented and the corresponding FIFO implements ECC. Counters are not reset and should be written with ones to be cleared.

34.10.29AHB address decode register

Table 382. 0x200 - FTADDR AHB address decode register

31	30	23	22	21	20	16
OORER	RESERVED		MEMMAP		HADDRW	
0	0		00		00	
rw	r		rw		r	

15	13	12	9	8	5	4	2	1	0
NCS		CSBASE		BANKBASE		ROWBASE		COLBASE	
111		0011		0101		010		10	
rw		rw		rw		rw		rw	

31 Out-of-range AHB error.

Set to 1 to trigger an AHB error response if an address above the amount of installed memory is made (determined based on CSBASE and NCS).

30 : 23 Reserved

22 : 21 Memory map selction (MEMMAP):
00: Standard DDR2/ DDR3 address map

01: Reserved

10: DDR4 Bank-group address map

11: DDR4 Reduced bank-group address mapping (DDR4 x16 only)

20 : 16 Indicates extra bits beyond 32 in the configured address width. (HADDRW)

15 : 13 Number of chip selects (ranks / external banks) installed, minus one. Used for out-of-range detection and for the scrubber.

12 : 9 AHB address bits used to determine chip select (CSBASE)

0: HADDR(24 : 22), (nports*4) MiB data/rank

1: HADDR(25 : 23), (nports*8) MiB data/rank

...

9: HADDR(31 : 31), (nports * 2) GiB data/rank (only two ranks used)

10: All addresses map to first chip select, (nports * 4) GiB data/rank

11-15: Reserved

See section 34.2.8 for guidance on setting up this field

8 : 5 AHB address bits used to determine internal bank index (BANKBASE)

Same range as CSBASE.

Note: If no address bits are needed to determine internal bank (number of ports on controller equals number of internal banks on memory), this should be set equal to CSBASE.

See section 34.2.8 for guidance on setting up this field

4 : 2 AHB address bits used to determine row (ROWBASE)

0: HADDR(..: 10), 1 KiB/row (over whole data bus)

1: HADDR(.. : 11), 2 KiB/row (over whole data bus)

...

5: HADDR(.. : 15), 32 KiB/row (over whole data bus)

6-7: Reserved

See section 34.2.8 for guidance on setting up this field

1 : 0 AHB address bits used to determine column (COLBASE)

0: HADDR(N:1), 16 data bits/column (hwidth=1, enx4=1)

1: HADDR(N:2), 32 data bits/column (hwidth=1,enx4=0 or hwidth=0,enx4=1)

2: HADDR(N:3), 64 data bits/column (hwidth=0, enx4=0)

3: Reserved

Note this field must be set corresponding to the backend hwidth/enx4 configuration for correct controller operation.

34.10.30AHB access configuration register

Table 383. 0x204 - FTADDR AHB access configuration register

31 28	27 24	23 19	18 16
PRELIMHI	PREFLIMLO	RESERVED	WQUEUEMAX
0100	0010	r	000
rw	rw	0	rw

15 13	12 5	4 3	2	1	0
ROWIDLECNT	ROWOTM	ROWOPOL	UEERR	UESIG	CESIG
000	00000000	01	1	1	1
rw	rw	rw	rw	rw	rw

31 : 28 High prefetch threshold, controls how many read commands, plus 1, should be issued ahead of time by the prefetcher. Must not be set higher than the capacity of the prefetch buffer. Only used if prefetching is enabled in the prefetch configuration register.

27 : 24 Low prefetch threshold, controls at what fill level of the prefetch buffer the prefetcher will fetch more data. If set to 0 the prefetcher will not fetch more data until the prefetch buffer is empty, setting it higher will allow the prefetching latency to be masked by remaining data already in the prefetch buffer. Only used if prefetching is enable the prefetch configuration register.

23 : 19 Reserved

18 : 16 Maximum writes allowed to be queued up inside the controller before giving wait states.

Setting the field to 0 disables the maximum, and it is then limited only by FIFO depth.

15 : 13 Row idle cycle count. This allows, in addition to other row policy settings in this register, to controls the minimum number of AHB cycles, plus 1, the AHB port will stay in idle state with a row opened before sending a row-close hint to the back-end.

12 : 5 Row open cycle count. This allows, in addition to the other row policy settings in this register, to set a minimum number of AHB cycles between opening a row and sending a row-close hint to the back-end. This can be matched to the tRAS configured in the back-end.

4 : 3 Front-end row close policy:

00 = aggressive, send row-close hint as early as possible. (for reads, before even receiving data)
01 = standard, send row-close hint whenever AHB port is in idle state.
10 = delayed, as standard but in addition waits for writes queued up in the command FIFO to be emptied by the back-end before send.
11 = keep rows open as long as possible, do not send row-close hints to back-end

2 Enable AHB error responses on uncorrectable EDAC errors when reading

1 Enable sideband signal for uncorrectable error when reading

This setting has no functional impact, except for masking the UE output signal

0 Enable sideband signal for correctable error when reading

This setting has no functional impact, except for masking the CE output signal

34.10.31 Prefetch configuration register

Table 384. 0x208 - FTADDR prefetch configuration register

31	26	25	16
RESERVED		PREFTO	
0		1111111111	
r		rw	

15	0
PREFETCH MASTER MASK	
1111111111111111	
rw	

31 : 26 Reserved

25 : 16 Prefetch timeout, if prefetch data has not been fetched within the specified number of AHB clock cycles, the prefetch data is discarded.

15 : 0 Prefetch enable mask per master.

Each register bit controls whether the corresponding master will trigger the prefetching unit.

Less bits may be implemented depending on the controller configuration.

34.10.32 Scrubber configuration register 1

Table 385. 0x20C - FTADDR scrubber configuration register 1

31	28	27	24	23	20	19	16
INITEN		READBACK		SCRUBPEND		RESERVED	
0000		0000		0000		0	
rw		rw		rw		r	

15	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED		RGALL	IEXD	IADR	SFCAL	BESFI	SFIDBL	SEFISTATE		SHOLD	SEFIEN	SCRUBCNT		SCREN
0		0	0	0	0	0	0	00		0	0	00		0
r		rw	rw	rw	rw	rw	rw	r		rw	rw	rw		rw

31 : 28 Initialization mode start, one bit per AHB port. When this bit is set to 1, the port will perform a full initialization run through the whole memory. The bit is cleared after completion.

Reads 1 if currently performing initialization.

27 : 24 Read-back start, one bit per AHB port. When this bit is set to 1, the port will perform a read-back and compare with expected initialization data (corresponding to current configuration).

This can be set to 1 at the same time as the INITEN field in order to trigger a initialization followed back-to-back with a read-back without allowing any other accesses in between.

23 : 20 Scrubber iteration pending, one bit per AHB port. This reads 1 when the periodic scrubber counter has expired, and a scrubber iteration is pending. This can be set to 1 manually by writing, and will in that case trigger a single iteration of the scrubber.

19 : 13 Reserved

12 SEFI handler regenerate all. If set to 0, regeneration will only scrub between scrubber start and scrubber end address, if set to 1 all memory will be scubbed during regeneration.

11 Initialize with existing data mode enable.

When performing initialization (using INITEN field of SEFI handler and init register) and this field is set to 1, use read-write cycles to keep the existing data content and only update the checkbits using read-write cycles.

Note that read-back after initialization will return errors after initializing the data in this mode, since the data contents will not match the init pattern.

10 Initialize with address mode enable.

When set to 1, the lower bits of the initialization and read-back pattern will not be taken from the initialization pattern registers but instead be set to the current rank,bank,row and column of the address being written.

9 Perform PHY re-calibration on byte lane when a SEFI is detected on that byte lane.

Table 385. 0x20C - FTADDR scrubber configuration register 1

8	Set to 1 to handle byte lane errors signaled by PHY as a detected SEFI. If set to 0, byte lane errors are not handled as a detected SEFI but are still reported through the BEREI interrupt. Note that this only has effect if the read-error signal is supported by the PHY and the PHY read error is enabled in the back-end's EDAC configuration register (see 34.10.22)
7	Set to 1 to disable the byte lane in the PHY when a SEFI is detected on that byte lane.
6 : 5	Current SEFI handling state machine state: 00 = Idle, scanning for SEFI 01 = SEFI detected, reprogramming mode registers 10 = SEFI detected, performing regeneration process 11 = Reserved for future use (currently never used)
4	SEFI handler hold off. While set to 1, triggered SEFI will not start regeneration process until cleared. If set to 1 while regeneration ongoing, the regeneration will be suspended until cleared.
3	SEFI handler enabled If set to 0, a detected SEFI will set the byte lane mask but not trigger any regeneration action. If set to 1, a detected SEFI will trigger the SEFI state machine to perform reprogramming and regeneration.
2 : 1	Length of each scrub iteration in 8-column bursts, minus one.
0	Enable scrubber

34.10.33Scrubber configuration register 2

Table 386. 0x210 - FTADDR scrubber configuration register 2

31	28	27	16
RESERVED		SEFI BYTE LANE MASK	
0		000000000000	
r		r	

15	0
SCRUBBER INTERVAL	
1111111111111111	
rw	

31 : 28	Reserved
27 : 16	Mask of SEFI status for each byte lane, 0=normal 1=SEFI
15 : 0	Scrubber interval, time in AHB cycles between each scrubber iteration, minus one

34.10.34IRQ pending register

Table 387. 0x214 - FTADDR IRQ pending register

31	16
RESERVED	
0	
r	

15 12	11	10	9	8	7 6 5 4	3 2 1 0
RESERVED	PSFI	BEREI	BESCI	SFI	UEI	CEI
0	0	0	0	0	0000	0000
r	wc	wc	wc	wc	wc	wc

31 : 12	Reserved
11	Permanent SEFI detected. Write 1 to clear
10	Backend read error signaled by PHY. Write 1 to clear
9	Backend EDAC self-check triggered. Write 1 to clear.
8	SEFI detected IRQ occurred. Write 1 to clear.
7 : 4	Uncorrectable error IRQ occurred, uncorrectable error address register is valid. Write 1 to clear. A separate IRQ bit for each AHB port is implemented
3 : 0	Correctable error IRQ occurred, correctable error address register is valid. Write 1 to clear. A separate IRQ bit for each AHB port is implemented.

34.10.35IRQ enable register

This register controls whether the interrupt line from the controller is asserted when the corresponding bit in the IRQ pending register goes from 0 to 1. If the IRQ is disabled in this register, then the IRQ pending bit will still be set in the same way, but the interrupt line will not be asserted.

Table 388. 0x218 - FTADDR IRQ enable register

31	16
RESERVED	
0	
r	

15 12	11	10	9	8	7 6 5	4	3 2 1	0
RESERVED	PSFIE	BEREIE	BESCIE	SFIE	RESERVED	UEIE	RESERVED	CEIE
0				0	0	0	0	0
r				wc	r	wc	r	wc

31 : 12	Reserved
11	Permanent SEFI IRQ enable
10	Backend read error IRQ enable
9	Backend EDAC self-check IRQ enable
8	SEFI detected IRQ enable
7 : 5	Reserved
4	Uncorrectable error IRQ enable. Enables/disables the IRQ for all AHB ports
3 : 1	Reserved
0	Correctable error IRQ enable. Enables/disables the IRQ for all AHB ports

34.10.36Scrubber UE register

Table 389. 0x21C - FTADDR scrubber UE register

31	30	29 27	26 24	23 16
SCRUE	RBERR	SECSNO	SEBANK	SEROW(15:8)
(nr)	(nr)	(nr)	(nr)	(nr)
wc	wc	r	r	r

15 8	7 0
SEROW(7:0)	SECOLHI
(nr)	(nr)
r	r

31	Set to 1 when uncorrectable error was detected while scrubbing on one of the ports. Must be cleared before another error can be logged in this register.
30	Set to 1 when an error during initialization readback was detected on one of the ports. Must be cleared before another error can be logged in this register.
29 : 27	Chip select number of scrubber uncorrectable error or readback error. Only valid when SCRUE or RBERR fields in this register are set.
26 : 24	Internal bank number of scrubber uncorrectable error or readback error. Only valid when SCRUE or RBERR fields in this register are set.
23 : 8	Row number of scrubber uncorrectable error or readback error. Only valid wen SCRUE or RBERR fields in this register are set.
7 : 0	Top bits (11:4) of column number where scrubber uncorrectable error or readback error occurred. Only valid when SCRUE or RBERR fields in this register are set.

34.10.37Scrubber CE byte lane counter register 1,2,3

Register 1 holds counters for bits 95:64 (check bits), register 2 holds counters for bits 63:32 (high part of data bits), register 3 holds counters for bits 31:0. All registers are cleared by writing to the BLCLR bit in byte lane error counter register 4.

Table 390. 0x220-0x228, FTADDR byte lane error counter register N (N=1..3)

31 30	29 24	23 22	21 16
RESERVED	LANE 4N+3 ERROR COUNTER	RESERVED	LANE 4N+2 ERROR COUNTER
0	000000	0	000000
r	r	r	r

15 14	13 8	7 6	5 0
RESERVED	LANE 4N+1 ERROR COUNTER	RESERVED	LANE 4N+0 ERROR COUNTER
0	000000	0	000000
r	r	r	r

31 : 30	Reserved
29 : 24	Wrapping scrubber corrected error counter for bits 95:88 / 63:56 / 31:24
23 : 22	Reserved
21 : 16	Wrapping scrubber corrected error counter for bits 87:80 / 55:48 / 23:16
15 : 14	Reserved
13 : 8	Wrapping scrubber corrected error counter for bits 79:72 / 47:40 / 15:8
7 : 6	Reserved
5 : 0	Wrapping scrubber corrected error counter for bits 71;64 / 39:32 / 7:0

34.10.38Scrubber CE byte lane counter register 4

Table 391. 0x22C - FTADDR byte lane error counter register 4

31	30														16
BLCLR	RESERVED														
0	0														
w	r														
15													6	5	0
RESERVED														BYTE LANE WATERMARK	
0														000000	
r														r	

31 Byte lane counter clear (BLCLR). Writing 1 to this bit clears the byte lane counters and resets the water mark

30 : 6 Reserved

5 : 0 Byte lane error counter watermark. This tracks the slowest moving of the byte lane error counters.

34.10.39Scrubber CE address counter register 1

Table 392. 0x230 - FTADDR address error counter register 1

31	30	28	27	24	23	20	19	16
AECLR	RESERVED		ADCTRCS2		ADCTRCS1		ADCTRCS0	
0	0		(nr)		(nr)		(nr)	
w	r		r		r		r	
15		12	11	8	7	4	3	0
RESERVED			ADCTRBA2		ADCTRBA1		ADCTRBA0	
0			(nr)		(nr)		(nr)	
r			r		r		r	

31 Address error counter clear (AECLR), writing 1 to this field clears all the counters in the address error counter registers.

30 : 28 Reserved

27 : 24 Saturating signed counter for chip select bit 2 (raised for CE in CS#7,6,5,4 lowered for CS#3,2,1,0)
Not implemented if support for 4 or less chip selects implemented.

23 : 20 Saturating signed counter for chip select bit 1 (raised for CE in CS#7,6,3,2, lowered for CS#5,4,1,0)
Not implemented if support for 2 or less chip selects implemented

19 : 16 Saturating signed counter for chip select bit 0 (raised for CE in CS#7,5,3,1, lowered for CS#6,4,2,0)
Not implemented if support for 1 chip select implemented

15 : 12 Reserved

11 : 8 Saturating signed counter for bank address bit 2 (raised if BA2=1, lowered if BA2=0 in address)

7 : 4 Saturating signed counter for bank address bit 1 (raised if BA1=1, lowered if BA1=0 in address)

3 : 0 Saturating signed counter for bank address bit 0 (raised if BA0=1, lowered if BA0=0 in address)

34.10.40Scrubber CE address counter register 2,3

Table 393. 0x234-0x238, FTADDR address error counter register 2/3

31 28	27 24	23 20	19 16
ADCTRROW15 / 7	ADCTRROW14 / 6	ADCTRROW13 / 5	ADCTRROW 12 / 4
(nr)	(nr)	(nr)	(nr)
r		r	
15 12	**11 8**	**7 4**	**3 0**
ADCTRROW 11 / 3	ADCTRROW10 / 2	ADCTRROW9 / 1	ADCTRROW8 / 0
(nr)	(nr)	(nr)	(nr)
r	r	r	r

31 : 28 Saturating signed counter for row address bit 15 / 7
27 : 24 Saturating signed counter for row address bit 14 / 6
23 : 20 Saturating signed counter for row address bit 13 / 5
19 : 16 Saturating signed counter for row address bit 12 / 4
15 : 12 Saturating signed counter for row address bit 11 / 3
11 : 8 Saturating signed counter for row address bit 10 / 2
7 : 4 Saturating signed counter for row address bit 9 / 1
3 : 0 Saturating signed counter for row address bit 8 / 0

34.10.41Scrubber CE address counter register 4

Table 394. 0x23C, FTADDR address error counter register 4

31 28	27 24	23 20	19 16
ADCTRCOL11	ADCTRCOL10	ADCTRCOL9	ADCTRCOL8
(nr)	(nr))	(nr)	(nr)
r		r	
15 12	**11 8**	**7 4**	**3 0**
ADCTRCOL7	ADCTRCOL6	ADCTRCOL5	ADCTRCOL4
(nr)	(nr)	(nr)	(nr)
r	r	r	r

31 : 28 Saturating signed counter for column address bit 11 (physically on AD13)
27 : 24 Saturating signed counter for column address bit 10 (physically on AD11)
23 : 20 Saturating signed counter for column address bit 9
19 : 16 Saturating signed counter for column address bit 8
15 : 12 Saturating signed counter for column address bit 7
11 : 8 Saturating signed counter for column address bit 6
7 : 4 Saturating signed counter for column address bit 5
3 : 0 Saturating signed counter for column address bit 4

34.10.42Access CE/UE location register, port 0,1,2,3

For each AHB port two registers are implemented, the first holding the address of a correctable error, the second holding the address of an uncorrectable error. These are only valid if the corresponding IRQ pending bit is set, and will not be updated until the IRQ pending bit has been cleared.

Table 395. 0x240-0x25C - FTADDR AHB CE/UE location register, port 0,1,2,3

31	16
CE/UE ADDRESS (31:16)	
(nr)	
r	

15	0
CE/UE ADDRESS (15:0)	
(nr)	
r	

31 : 0 AHB address of access with error.

34.10.43Prefetch status register, port N

Table 396. 0x260-0x26C, FTADDR prefetch status register for port #N

31	30	29 25	24 16
PFACT	PFDRN	PREF FETCH POSITION	PREF READOUT POSITION
0	0	(nr)	(nr)
r	r	r	r

15	0
PREFETCH ROW	
(nr)	
r	

31 Prefetcher activated status. 1=activated, 0=idle

30 Prefetcher draining status, 1=draining, 0=normal

This will be set after a master accessing a different address than predicted by the prefetcher before all prefetch data in flight has been discarded

29 : 25 Current fetch position of the prefetcher, column bits 7:3. Only valid when PFACT is 1.

The other, higher, column bits can be deducted from the read position

24 : 16 Current read position of the prefetcher, column bits 11:3. Only valid when PFACT is 1.

15 : 0 Current prefetcher row. Only valid when PFACT is 1.

Note: 1-4 separate registers with the same format is implemented, one for each AHB port. Additional bank/CS address bits are in the prefetch bank register shared between all the ports.

34.10.44Prefetch bank status register

Table 397. 0x270 - FTADDR prefetch bank status register

31 30	29 27	26 24	23 22	21 19	18 16
PREF3BG	PREF3CS	PREF3IBANK	PREF2BG	PREF2CS	PREF2IBANK
(nr)	(nr)	(nr)	(nr)	(nr)	(nr)
r	r	r	r	r	r

15 14	13 11	10 8	7 6	5 3	2 0
PREF1BG	PREF1CS	PREF1IBANK	PREF0BG	PREF0CS	PREF0IBANK
(nr)	(nr)	(nr)	(nr)	(nr)	(nr)
r	r	r	r	r	r

31 : 30 Port 3 prefetcher current bank group number. Only valid when PFACT is set in prefetch status register for port 0.

29 : 27 Port 3 prefetcher current chip select number. Only valid when PFACT is set in prefetch status register for port 3. Only implemented in 4-port configuration

26 : 24 Port 3 prefetcher internal bank number high bits (lower bits implied “11” by port number). Only valid when PFACT is set in prefetch status register for port 3. Only implemented in 4-port configuration (1 bit wide).

23 : 22 Port 2 prefetcher current bank group number. Only valid when PFACT is set in prefetch status register for port 0.

21 : 19 Port 2 prefetcher current chip select number. Only valid when PFACT is set in prefetch status register for port 2. Only implemented in 4-port configuration

18 : 16 Port 2 prefetcher internal bank number high bits (lower bits implied “10” by port number). Only valid when PFACT is set in prefetch status register for port 2. Only implemented in 4-port configuration (1 bit wide).

15 : 14 Port 1 prefetcher current bank group number. Only valid when PFACT is set in prefetch status register for port 0.

13 : 11 Port 1 prefetcher current chip select number. Only valid when PFACT is set in prefetch status register for port 1. Only implemented in 4-port and 2-port configurations.

10 : 8 Port 1 prefetcher internal bank number high bits (lower bits implied “1” or “01” by port number). Only valid when PFACT is set in prefetch status register for port 1. Field is 0-2 bits wide depending on number of ports configured.

7 : 6 Port 0 prefetcher current bank group number. Only valid when PFACT is set in prefetch status register for port 0.

5 : 3 Port 0 prefetcher current chip select number. Only valid when PFACT is set in prefetch status register for port 0.

2 : 0 Port 0 prefetcher internal bank number high bits (in 2/4-port configurations, lower bits implied “0” or “00” by port number). Only valid when PFACT is set in prefetch status register for port 0. Field is 1-3 bits wide depending on number of ports configured.

34.10.45Scrubber start address register

Table 398. 0x274 - FTADDR scrubber start address register

31	30 28	27 25	24 16
RES	SCSTCS	SCSTIBANK	SCSTROW(15:7)
0	000	000	000000000
r	rw	rw	rw

15 9	8 0
SCSTROW(6:0)	SCSTCOL
0000000	000000000
rw	rw

31 Reserved

30 : 28 Scrubber start address chip select

27 : 25 Scrubber start address internal bank. For 2/4 port configurations, this field holds only the high internal bank bits, since the lower bits are inferred from the port number.

24 : 9 Scrubber start address row

8 : 0 Scrubber start address column number bits 11:3. Bits 2:0 of the starting column are always 0 by design.

34.10.46Scrubber end address register

Table 399. 0x278 - FTADDR scrubber end address register

31	30 28	27 25	24 16
RES	SCEDCS	SCEDIBANK	SCEDROW(15:7)
0	111	111	111111111
r	rw	rw	rw

15 9	8 0
SCEDROW(6:0)	SCEDCOL
1111111	111111111
rw	rw

31 Reserved

30 : 28 Scrubber end address chip select

27 : 25 Scrubber end address internal bank. For 2/4 port configurations, this field holds only the high internal bank bits, since the lower bits are inferred from the port number.

24 : 9 Scrubber end address row

8 : 0 Scrubber end address column number bits 11:3. Bits 2:0 of the starting column are always 0 by design.

34.10.47Frontend FIFO error counter register

Table 400. 0x27C - FTADDR frontend FIFO error counter register

31	30	29 28	27	26	25 24	23	22	21 20	19	18	17 16
RES	RS3UE	RS3CE	RD3UE	PF3UE	RDPF3CE	RES	RS2UE	RS2CE	RD2UE	PF2UE	RDPF2CE
0	(nr)	(nr)	(nr)	(nr)	(nr)	0	(nr)	(nr)	(nr)	(nr)	(nr)
r	wc	wc	wc	wc	wc	r	wc	wc	wc	wc	wc
15	14	13 12	11	10	9 8	7	6	5 4	3	2	1 0
RES	RS1UE	RS1CE	RD1UE	PF1UE	RDPF1CE	RES	RS0UE	RS0CE	RD0UE	PF0UE	RDPF0CE
0	(nr)	(nr)	(nr)	(nr)	(nr)	0	(nr)	(nr)	(nr)	(nr)	(nr)
r	wc	wc	wc	wc	wc	r	wc	wc	wc	wc	wc

31	Reserved
30	Uncorrectable error occurred on response FIFO for port #3
29 : 28	Correctable error counter for response FIFO for port #3
27	Uncorrectable error occurred on read-data FIFO for port #3
26	Uncorrectable error occurred on prefetch-data FIFO for port #3
25 :24	Correctable error saturating counter for read-data and prefetch-data FIFOs for port #3
23	Reserved
22	Uncorrectable error occurred on response FIFO for port #2
21 : 20	Correctable error counter for response FIFO for port #2
19	Uncorrectable error occurred on read-data FIFO for port #2
18	Uncorrectable error occurred on prefetch-data FIFO for port #2
17 :16	Correctable error saturating counter for read-data and prefetch-data FIFOs for port #2
15	Reserved
14	Uncorrectable error occurred on response FIFO for port #1
13 : 12	Correctable error counter for response FIFO for port #1
11	Uncorrectable error occurred on read-data FIFO for port #1
10	Uncorrectable error occurred on prefetch-data FIFO for port #1
9 :8	Correctable error saturating counter for read-data and prefetch-data FIFOs for port #1
7	Reserved
6	Uncorrectable error occurred on response FIFO for port #0
5 : 4	Correctable error counter for response FIFO for port #0
3	Uncorrectable error occurred on read-data FIFO for port #0
2	Uncorrectable error occurred on prefetch-data FIFO for port #0
1 :0	Correctable error saturating counter for read-data and prefetch-data FIFOs for port #0

Note that error counters are only implemented if the port is implemented and the corresponding FIFO implements ECC. Counters are not reset and should be written with ones to be cleared.

34.10.48Initialization pattern register 1,2,3,4

Table 401. 0x280-0x28C - FTADDR initialization pattern register N

31	16
INITIALIZATION PATTERN DATA (31:16)	
*	
rw	
15	0
INITIALIZATION PATTERN DATA(15:0)	
*	
rw	

31 : 0 Data to use for initialization and readback operation.

34.10.49Scrubber position register, port N

Table 402. 0x290-0x29C - FTADDR scrubber position register

31	30 28	27 25	24 16
SCWR	SCCURCS	SCCURIBANK	SCCURROW(15:7)
0	000	000	000000000
r	r	r	r

15 9	8 0
SCCURROW(6:0)	SCCURCOL
0000000	000000000
r	r

31 Wrapped bit. Indicates that the scrubber for this port has completed a full iteration.

30 : 28 Scrubber current chip select

27 : 25 Scrubber current internal bank. For 2/4 port configurations, this field holds only the high internal bank bits, since the lower bits are inferred from the port number.

24 : 9 Scrubber current row

8 : 0 Scrubber current column number bits 11:3

34.10.50PHY specific registers for Ultrascale PHY

The PHY specific registers are accessed using the indirect PHY address and data registers (see sections 34.10.24 and 34.10.25).

Table 403. PHY indirect address 0 - VT Tracking configuration register

31	30 24	23 16
RMISS	RESERVED	RDMARGIN
(nr)	0	(nr)
wc	r	r

15 8	7 0
DRTHR	VTPER
*	*
rw	rw

31 VT read miss, indicate that a VT tracking period passed without a read. This field is not reset, write 1 to clear and also reset the read margin counter.

30 : 24 Reserved.

23 : 16 Read margin indicator, this shows the lowest amount of cycles left when a read was performed. Writing a 1 to the RMISS field will reset this field to the maximum value to restart the tracking.

Table 403. PHY indirect address 0 - VT Tracking configuration register

15 : 8	Dummy read threshold, when this number of cycles are left until the current VT period (one micro-second) expires and no read has been made, the controller will insert a dummy read as soon as possible. This field is expressed in controller clock cycles (1/4 of memory frequency), not memory clock cycles. The reset value is set via a generic into the core.
7 : 0	VT tracking period, controls the interval between periodic activities to track VT variations. To get the 1 us period required by the PHY, set this field to (MEMFREQ/4)-2, where MEMFREQ is the DDR memory clock frequency in MHz. The reset value is set via a generic into the core.

34.11 Vendor and device identifiers

This controller is identified with vendor ID 0x001 (Frontgrade Gaisler) and device ID 0x0AE. The version described in this document is version 0.

34.12 Implementation

34.12.1 Endianness

The core does not change endianness behavior depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). Since accesses should match the bus width the core supports both big-endian and little-endian systems.

34.13 Configuration options

Tables 404 shows the configuration options (VHDL generics) of the core.

*Table 404.*Configuration options for FTADDR (ftaddr_gr entity)

Generic	Function	Allowed range	Default
hindex0	AHB slave index for port 0	0 - NAHBSLV-1	0
hindex1	AHB slave index for port 1	0 - NAHBSLV-1	0
hindex2	AHB slave index for port 2	0 - NAHBSLV-1	0
hindex3	AHB slave index for port 3	0 - NAHBSLV-1	0
haddr0	ADDR field of the AHB BAR0 defining the SDRAM address area on port 0.	0 - 16#FFF#	0
hmask0	MASK field of the AHB BAR0 defining the SDRAM address area on port 0.	0 - 16#FFF#	0
haddr1	ADDR field of the AHB BAR0 defining the SDRAM address area on port 1.	0 - 16#FFF#	0
hmask1	MASK field of the AHB BAR0 defining the SDRAM address area on port 1.	0 - 16#FFF#	0
haddr2	ADDR field of the AHB BAR0 defining the SDRAM address area on port 2.	0 - 16#FFF#	0
hmask2	MASK field of the AHB BAR0 defining the SDRAM address area on port 2.	0 - 16#FFF#	0
haddr3	ADDR field of the AHB BAR0 defining the SDRAM address area on port 3.	0 - 16#FFF#	0
hmask3	MASK field of the AHB BAR0 defining the SDRAM address area on port 3.	0 - 16#FFF#	0
ioaddr0	ADDR field of the AHB BAR1 defining I/O address space on port 0 where control registers are mapped.	0 - 16#FFF#	16#000#
iomask0	MASK field of the AHB BAR1 defining I/O address space on port 0.	0 - 16#FFF#	16#FFF#
ioaddr1	ADDR field of the AHB BAR1 defining I/O address space on port 1 where control registers are mapped.	0 - 16#FFF#	16#000#

*Table 404.*Configuration options for FTADDR (ftaddr_gr entity)

Generic	Function	Allowed range	Default
iomask1	MASK field of the AHB BAR1 defining I/O address space on port 1.	0 - 16#FFF#	16#FFF#
ioaddr2	ADDR field of the AHB BAR1 defining I/O address space on port 2 where control registers are mapped.	0 - 16#FFF#	16#000#
iomask2	MASK field of the AHB BAR1 defining I/O address space on port 2.	0 - 16#FFF#	16#FFF#
ioaddr3	ADDR field of the AHB BAR1 defining I/O address space on port 3 where control registers are mapped.	0 - 16#FFF#	16#000#
iomask3	MASK field of the AHB BAR1 defining I/O address space on port 3.	0 - 16#FFF#	16#FFF#
hirq	Index of the interrupt line.	0 - NAHBIRQ-1	0
tech	Technology value for FIFO implementation.	0 - NTECH	inferred
ahbbits	Width of AHB read/write data buses and maximum access size.	32, 64, 128	AHBDW
ddrbits	Width of DDR data bus. This is the width of the external data bus, so the DFI data buses will have double the width.	32 - 96	96
freqratio	Frequency ratio of DFI interface	1, 2, 4	1
nports	Number of implemented AHB ports	1-4	1
nahbmst	Number of AHB masters on each bus (detemines maximum HMASTER value)	1 - 16	16
numcs	Number of chip select signals implemented	1 - 8	1
ctrldup	Option to create duplicate identical copies of memory control signals. Set to 1 for single copy corresponding to standard DFI interface.	1 - 4	1
numrwen	Width of dfi_rddata_en, dfi_rddata_valid, dfi_wrdata_en signals. Corresponds to ‘DFI Data Enable Width’ and ‘DFI Read Data Enable Width’ in DFI standard.	1 - N	1
numrdlvlphy	Width of dfi_rdlvl_req and dfi_rdlvl_gate_req signals. Corresponds to ‘DFI Read Leveling PHY IF Width’ in DFI standard	1 - N	1
numrdlvlmc	Width of dfi_rdlvl_en, dfi_rdlvl_gate_en, dfi_rdlvl_edge, dfi_rdlvl_load signals. Corresponds to ‘DFI Read Leveling MC IF Width’ in DFI standard.	1 - N	1
numwrlvlphy	Width of dfi_wrlvl_req signal. Corresponds to ‘DFI Write Leveling PHY IF Width’ in DFI standard.	1 - N	1
numwrlvlmc	Width of dfi_wrlvl_en signal. Corresponds to ‘DFI Write Leveling MC IF Width’ in DFI standard	1 - N	1
rdblvlbits	Width per byte laneof dfi_rdlvl_delay vector. Corresponds to ‘DFI Read Leveling Delay Width’ in DFI standard	1 - N	1
rdglvlbits	Width per byte lane of dfi_rdlvl_gate_delay signal. Corresponds to ‘DFI Read Leveling Gate Delay Width’ in DFI standard	1 - N	1
wrlvlbits	Width per byte lane of dfi_wrlvl_delay signal. Corresponds to ‘DFI Write Leveling Delay Width’ in DFI standard	1 - N	1

*Table 404.*Configuration options for FTADDR (ftaddr_gr entity)

Generic	Function	Allowed range	Default
phyimpl	PHY implementation ID. See section 34.3.12.	0 - PHY_MAX	0
genphy_trden	Specifies trddata_en parameter for generic DFI implementation. Add 100 to specify CAS latency relative delay.	0-11, 95-110	0
genphy_twrlat	Specifies tphy_wrlat parameter for generic DFI implementation. Add 100 to specify write latency relative delay.	0-11, 95-110	0
genphy_twrdata	Specifies twrdata parameter for generic DFI implementation.	0 - 11	0
fifoftmask	Bit mask specifying which syncfifo_2p instances should have the FT generic set. Sum of: 1: Command FIFOs 2: Response FIFOs 4: Write data FIFOs 8: Read data FIFOs 16: Prefetch read data FIFOs	0-31	31
fifoinfmask	Bit mask specifying which syncfifo_2p instances should have the tech generic set to inferred instead of tech value given in generic. Same encoding as fifoftmask.	0-31	0
dynrst	Enable reset values control from input signals	0 - 1	0
phyctrlbits	Number of bits implemented in generic PHY control register	0 - 32	0
xilphy_vtper	Default value of VT tracking period for Ultrascale PHY	0 - 255	73
xilphy_drthr	Default value of dummy read threshold for Ultrascale PHY	0 - 255	48

34.14 Signal descriptions

Table 405 shows the interface signals (VHDL ports) of the of the core.

*Table 405.*Signal descriptions for FTADDR (ftaddr_gr entity)

Signal name	Type	Function	Active
AHB_CLK	Input	AHB clock	Rising
AHB_RSTN	Input	Reset input for AHB clock domain	Low
AHBSI[nports-1:0]	Input	AMBA AHB signal records.	-
AHBSO[nports-1:0]	Output	Port 0 signals are contained in ahbsi[0]/ahbso[0], port 1 in ahbsi[1]/ahbso[1], and so on. All ports are clocked by the same AHB clock but are otherwise independent. The ports are not required to be on the same AHB bus. The IRQ line is driven on AHB port 0 only.	-
AHB_CE[nports-1:0]	Output	EDAC corrected error signal for each port	High
AHB_UE[nports-1:0]	Output	EDAC uncorrectable error signal for each port	High
DYNSYNC	Input	Dynamic synchronization control signal, see section 34.7.2. This is a pseudo-static signal that should only change during reset.	-
DFI_CLK	Input	DFI interface clock	Rising
DFI_RSTN	Input	Reset input for DFI clock domain	Low

Table 405. Signal descriptions for FTADDR (ftaddr_gr entity)

Signal name	Type	Function	Active
DFI_CS_N[freqratio*ctrldup*numcs-1:0]	Output	DFI interface memory control signals. Forwarded to the memory devices by the PHY. DFI_RESET_N is used for DDR3 only. If ctrldup is set higher than one, multiple identical copies of the output signals are generated. DFI_BG and DFI_ACT_N are used for DDR4 only.	Low
DFI_BANK[freqratio*ctrldup*3-1:0]	Output		-
DFI_ADDRESS[freqratio*ctrldup*16-1:0]	Output		-
DFI_RAS_N[freqratio*ctrldup-1:0]	Output		Low
DFI_CAS_N[freqratio*ctrldup-1:0]	Output		Low
DFI_WE_N[freqratio*ctrldup-1:0]	Output		Low
DFI_CKE[freqratio*ctrldup*numcs-1:0]	Output		High
DFI_ODT[freqratio*ctrldup*numcs-1:0]	Output		High
DFI_RESET_N[freqratio*ctrldup*numcs-1:0]	Output		Low
DFI_BG[freqratio*ctrldup*2-1:0]	Output		Low
DFI_ACT_N[freqratio*ctrldup-1:0]	Output		HIGH
DFI_WRDATA[freqratio*2*ddrbits-1:0]	Output	DFI memory write interface. All bits of DFI_WRDATA_EN for the same phase are driven with identical values.	-
DFI_WRDATA_EN[freqratio*numrwen-1:0]	Output		High
DFI_WRDATA_MASK[freqratio*ddrbits/4-1:0]	Output		-
DFI_RDDATA_EN[freqratio*numrwen-1:0]	Output	DFI memory read interface. All bits of DFI_RDDATA_EN for the same phase are driven with identical values. Only bit 0 of DFI_RDDATA_VALID is used.	High
DFI_RDDATA[freqratio*2*ddrbits-1:0]	Input		-
DFI_RDDATA_VALID[freqratio*numrwen-1:0]	Input		High
DFI_CTRLUPD_REQ	Output	DFI update interface	High
DFI_CTRLUPD_ACK	Input		High
DFI_PHYUPD_REQ	Input		High
DFI_PHYUPD_TYPE[1:0]	Input		-
DFI_PHYUPD_ACK	Output		High
DFI_DATA_BYTE_DISABLE[ddrbits/8-1:0]	Output	DFI data byte disable signal	High
DFI_DRAM_CLK_DISABLE	Output	DFI clock output disable signal	High
DFI_INIT_COMPLETE	Input	DFI initialization complete signal from PHY	High
DFI_RDLVL_MODE[1:0]	Input	DFI read leveling (gate and eye training) interface.	-
DFI_RDLVL_REQ[numrdlvlphy-1:0]	Input		High
DFI_RDLVL_EN[numrdlvlmc-1:0]	Output		High
DFI_RDLVL_GATE_MODE[1:0]	Input		-
DFI_RDLVL_GATE_REQ[numrdlvlphy-1:0]	Input		High
DFI_RDLVL_GATE_EN[numrdlvlmc-1:0]	Output		High
DFI_RDLVL_CS_N[numcs-1:0]	Output		Low
DFI_RDLVL_EDGE[numrdlvlmc-1:0]	Output		-
DFI_RDLVL_DELAY[(ddrbits/8)*rdblvlbits-1:0]	Output		-
DFI_RDLVL_GATE_DELAY[(ddrbits/8)*rdglvlbits-1:0]	Output		
DFI_RDLVL_LOAD[numrdlvlmc-1:0]	Output		High
DFI_RDLVL_RESP[ddrbits-1:0]	Input		

Table 405. Signal descriptions for FTADDR (ftaddr_gr entity)

Signal name	Type	Function	Active
DFI_WRLVL_MODE[1:0]	Input	DFI write leveling interface	-
DFI_WRLVL_REQ[numwrlvlphy-1:0]	Input		High
DFI_WRLVL_EN[numwrlvlmc-1:0]	Output		High
DFI_WRLVL_CS_N[numcs-1;0]	Output		Low
DFI_WRLVL_DELAY[(ddrbits/8)*rdblvl-bits-1:0]	Output		-
DFI_WRLVL_LOAD[numwrlvlmc-1:0]	Output		High
DFI_WRLVL_STROBE[numwrlvlmc-1:0]	Output		High
DFI_WRLVL_RESP[ddrbits-1:0]	Input		-
XDFI_TERM_EN[freqratio-1:0]	Output	Enable for local termination on DQ bus. (PHY specific signal)	High
XDFI_RDERR[(ddrbits/8)-1:0]	Input	Byte lane error detection signal from PHY PHY-specific signal, tie low if unused	High
XDFI_SOFTRST	Output	Soft reset signal to PHY (PHY-specific signal)	High
XDFI_PHYCTRL[31:0]	Output	Generic PHY control register output	-
XDFI_BL_RESETN[(ddrbits/8)-1:0]	Output	RESETN and CKE signals per byte lane, to use instead of regular DFI signals when implementing a memory system with reboot support (see section 34.3.13)	Low
XDFI_BL_CKE[(ddrbits/8)-1:0]	Output		High
XDFI_BL_PDN[(ddrbits/8)-1:0]	Output	Output signal for power cycling control per byte lane, only for use when implementing a memory system with reboot support (see section 34.3.13)	Low
XDFI_CLK_ZERO	Output	Output signal for forcing clock outputs low, only for use when implementing a memory system with reboot support (see section 34.3.13)	High
AFI_DQS_BURST[freqratio-1:0]	Output	Additional control signal needed when interfacing AFI PHY	High
AFI_WLAT[5:0]	Input	AFI write latency signal, only used when interfacing AFI PHY	-
XPCLK	Input	These signals are used for a reserved PHY implementation and should not be used, tie the inputs to zero.	
XPRSTN	Input		
XPSEL	Output		
XPWRITE	Output		
XPENABLE	Output		
XPADDR[7:0]	Output		
XPWDATA[31:0]	Output		
XPREADY	Input		
XPRDATA[31:0]	Input		
XPSLVERR	Input		
XILP_RDCAS	Output	Signals used only for Ultrascale PHY implementation, connect to correspondonding signal on PHY. For other PHYs, tie inputs to constant zero.	
XILP_WRCAS	Output		
XILP_WINRANK[1:0]	Output		
XILP_WININJTXN	Output		
XILP_WRDATAEN	Input		
XILP_PERRDDONE	Input		
XILP_GTRDY	Output		

Table 405. Signal descriptions for FTADDR (ftaddr_gr entity)

Signal name	Type	Function	Active
RSTVAL008[35:0]	Input	Reset values for configuration registers, used only if dynrst generic is set. The name corresponds to the offset in the register area of the register. Bits 31:0 are used as the reset value if dynrst is set. When bit 32 is set, and dynrst is set, this forces the register permanently to the supplied reset value and the corresponding register bits can be optimized out by the synthesis. Bit 35:33 are currently unused.	-
RSTVAL00C[35:0]	Input		-
RSTVAL010[35:0]	Input		-
RSTVAL014[35:0]	Input		-
RSTVAL018[35:0]	Input		-
RSTVAL01C[35:0]	Input		-
RSTVAL020[35:0]	Input		-
RSTVAL024[35:0]	Input		-
RSTVAL040[35:0]	Input		-
RSTVAL044[35:0]	Input		-
RSTVAL048[35:0]	Input		-
RSTVAL04C[35:0]	Input		-
RSTVAL050[35:0]	Input		-
RSTVAL054[35:0]	Input		-
RSTVAL058{35:0]	Input		-
RSTVAL05C[35:0]	Input		-
RSTVAL060[35:0]	Input		-
RSTVAL064[35:0]	Input		-
RSTVAL06C[35:0]	Input		-
RSTVAL070[35:0]	Input		-
RSTVAL080[35:0]	Input		-
RSTVAL200[35:0]	Input		-
RSTVAL204[35:0]	Input		-
RSTVAL208[35:0]	Input		-
RSTVAL20C[35:0]	Input		-
RSTVAL210[35:0]	Input		-
RSTVAL218[35:0]	Input		-
RSTVAL274[35:0]	Input		-
RSTVAL278[35:0]	Input		-
RSTVAL280[35:0]	Input		-
RSTVAL284[35:0]	Input		-
RSTVAL288[35:0]	Input		-
RSTVAL28C[35:0]	Input		-

34.15 Library dependencies

Tables 406 shows libraries used when instantiating the core (VHDL libraries).

Table 406. Library dependencies for instantiating ftaddr_gr top level

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	Signal record definitions
GAISLER	FTADDR_PKG	Component, Types	Component declaration for GRLIB version
TECHMAP	GENCOMP	Constants	Constants for tech generic

34.16 Component declaration

The component declaration for the stand-alone and GRLIB versions are provided below.

```
component ftaddr_gr is
    generic (
      hindex0     : integer := 0;
      hindex1     : integer := 0;
      hindex2     : integer := 0;
      hindex3     : integer := 0;
      haddr0      : integer := 0;
      hmask0      : integer := 0;
      haddr1      : integer := 0;
      hmask1      : integer := 0;
      haddr2      : integer := 0;
      hmask2      : integer := 0;
      haddr3      : integer := 0;
      hmask3      : integer := 0;
      ioaddr0     : integer := 16#000#;
      iomask0     : integer := 16#FFC#;
      ioaddr1     : integer := 16#000#;
      iomask1     : integer := 16#FFC#;
      ioaddr2     : integer := 16#000#;
      iomask2     : integer := 16#FFC#;
      ioaddr3     : integer := 16#000#;
      iomask3     : integer := 16#FFC#;
      hirq        : integer := 0;
      tech        : integer := inferred;
      ahbbits     : integer := AHBDW;
      ddrbits     : integer := 96;
      nports      : integer := 1;
      nahbmst     : integer := 16;
      numcs       : integer := 1;
      ctrldup     : integer := 1;
      csdup       : integer := 1;
      numrwen     : integer := 1;
      numrdlvlphy : integer := 1;
      numrdlvlmc  : integer := 1;
      numwrlvlphy : integer := 1;
      numwrlvlmc  : integer := 1;
      rdblvlbits  : integer := 1;
      rdglvlbits  : integer := 1;
      wrlvlbits   : integer := 1;
      phyimpl     : integer := 0;
      genphy_trden   : integer := 98;
      genphy_twrlat  : integer := 100;
      genphy_twrdata : integer := 0;
      fifoftmask  : integer := 16#1F#;
      fifoinfmask : integer := 0;
      dynrst      : integer := 0;
      phyctrlbits : integer := 0
      );
    port (
      -- AMBA clock/reset
      ahb_clk               : in std_ulogic;
      ahb_rstn              : in std_ulogic;
      -- AMBA ports
      ahbsi                 : in  ahb_slv_in_vector_type(nports-1 downto 0);
      ahbso                 : out ahb_slv_out_vector_type(nports-1 downto 0);
      -- EDAC sideband signals
      ahb_ce                : out std_logic_vector(nports-1 downto 0);
      ahb_ue                : out std_logic_vector(nports-1 downto 0);
      -- Sync configuration
      dynsync               : in std_ulogic;
      -- DFI clock/reset
      dfi_clk               : in  std_ulogic;
      dfi_rstn              : in  std_ulogic;
      -- DFI command interface
      dfi_cs_n              : out std_logic_vector(csdup*numcs-1 downto 0);
      dfi_bank              : out std_logic_vector(ctrldup*3-1 downto 0);
      dfi_address           : out std_logic_vector(ctrldup*16-1 downto 0);
```

```
      dfi_ras_n              : out std_logic_vector(ctrldup-1 downto 0);
      dfi_cas_n              : out std_logic_vector(ctrldup-1 downto 0);
      dfi_we_n               : out std_logic_vector(ctrldup-1 downto 0);
      dfi_cke                : out std_logic_vector(csdup*numcs-1 downto 0);
      dfi_odt                : out std_logic_vector(csdup*numcs-1 downto 0);
      dfi_reset_n            : out std_logic_vector(csdup*numcs-1 downto 0);
      dfi_wrdata             : out std_logic_vector(2*ddrbits-1 downto 0);
      dfi_wrdata_en          : out std_logic_vector(numrwen-1 downto 0);
      dfi_wrdata_mask        : out std_logic_vector(ddrbits/4-1 downto 0);
      dfi_rddata_en          : out std_logic_vector(numrwen-1 downto 0);
      dfi_rddata             : in  std_logic_vector(2*ddrbits-1 downto 0);
      dfi_rddata_valid       : in  std_logic_vector(numrwen-1 downto 0);
      dfi_bg                 : in  std_logic_vector(ctrldup*2-1 downto 0);
      dfi_act_n              : in  std_logic_vector(ctrldup-1 downto 0);
      -- DFI update interface
      dfi_ctrlupd_req        : out std_ulogic;
      dfi_ctrlupd_ack        : in  std_ulogic;
      dfi_phyupd_req         : in  std_ulogic;
      dfi_phyupd_type        : in  std_logic_vector(1 downto 0);
      dfi_phyupd_ack         : out std_ulogic;
      -- DFI init/power-down signals
      dfi_data_byte_disable  : out std_logic_vector(ddrbits/8-1 downto 0);
      dfi_dram_clk_disable   : out std_logic_vector(csdup*numcs-1 downto 0);
      dfi_init_complete      : in  std_ulogic;
      -- DFI trainig/leveling interface
      dfi_rdlvl_mode         : in  std_logic_vector(1 downto 0);
      dfi_rdlvl_req          : in  std_logic_vector(numrdlvlphy-1 downto 0);
      dfi_rdlvl_en           : out std_logic_vector(numrdlvlmc-1 downto 0);
      dfi_rdlvl_gate_mode    : in  std_logic_vector(1 downto 0);
      dfi_rdlvl_gate_req     : in  std_logic_vector(numrdlvlphy-1 downto 0);
      dfi_rdlvl_gate_en      : out std_logic_vector(numrdlvlmc-1 downto 0);
      dfi_rdlvl_cs_n         : out std_logic_vector(csdup*numcs-1 downto 0);
      dfi_rdlvl_edge         : out std_logic_vector(numrdlvlmc-1 downto 0);
      dfi_rdlvl_delay        : out std_logic_vector((ddrbits/8)*rdblvlbits-1 downto 0);
      dfi_rdlvl_gate_delay   : out std_logic_vector((ddrbits/8)*rdglvlbits-1 downto 0);
      dfi_rdlvl_load         : out std_logic_vector(numrdlvlmc-1 downto 0);
      dfi_rdlvl_resp         : in  std_logic_vector(ddrbits-1 downto 0);
      dfi_wrlvl_mode         : in  std_logic_vector(1 downto 0);
      dfi_wrlvl_req          : in  std_logic_vector(numwrlvlphy-1 downto 0);
      dfi_wrlvl_en           : out std_logic_vector(numwrlvlmc-1 downto 0);
      dfi_wrlvl_cs_n         : out std_logic_vector(csdup*numcs-1 downto 0);
      dfi_wrlvl_delay        : out std_logic_vector((ddrbits/8)*wrlvlbits-1 downto 0);
      dfi_wrlvl_load         : out std_logic_vector(numwrlvlmc-1 downto 0);
      dfi_wrlvl_strobe       : out std_logic_vector(numwrlvlmc-1 downto 0);
      dfi_wrlvl_resp         : in  std_logic_vector(ddrbits-1 downto 0);
      -- PHY specific signals
      xdfi_term_en           : out std_ulogic;
      xdfi_rderr             : in std_logic_vector((ddrbits/8)-1 downto 0) := (others =>
'0');
      xdfi_softrst           : out std_ulogic;
      xdfi_phyctrl           : out std_logic_vector(31 downto 0);  -- Generic PHY control
register
      xdfi_bl_resetn         : out std_logic_vector(ddrbits/8-1 downto 0);
      xdfi_bl_cke            : out std_logic_vector(ddrbits/8-1 downto 0);
      xdfi_bl_pdn            : out std_logic_vector(ddrbits/8-1 downto 0);
      xdfi_clk_zero          : out std_ulogic;
      afi_dqs_burst          : out std_ulogic;
      afi_wlat               : in  std_logic_vector(5 downto 0) := "000000";
      -- Reset values
      rstval008 : in std_logic_vector(35 downto 0) := x"000000000";
      rstval00C : in std_logic_vector(35 downto 0) := x"000000000";
      rstval010 : in std_logic_vector(35 downto 0) := x"000000000";
      rstval014 : in std_logic_vector(35 downto 0) := x"000000000";
      rstval018 : in std_logic_vector(35 downto 0) := x"000000000";
      rstval01C : in std_logic_vector(35 downto 0) := x"000000000";
      rstval020 : in std_logic_vector(35 downto 0) := x"000000000";
      rstval024 : in std_logic_vector(35 downto 0) := x"000000000";
      rstval040 : in std_logic_vector(35 downto 0) := x"000000000";
      rstval044 : in std_logic_vector(35 downto 0) := x"000000000";
      rstval048 : in std_logic_vector(35 downto 0) := x"000000000";
      rstval04C : in std_logic_vector(35 downto 0) := x"000000000";
```

```
        rstval050 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval054 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval058 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval05C : in std_logic_vector(35 downto 0) := x"000000000";
        rstval060 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval064 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval06C : in std_logic_vector(35 downto 0) := x"000000000";
        rstval070 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval080 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval200 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval204 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval208 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval20C : in std_logic_vector(35 downto 0) := x"000000000";
        rstval210 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval218 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval274 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval278 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval280 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval284 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval288 : in std_logic_vector(35 downto 0) := x"000000000";
        rstval28C : in std_logic_vector(35 downto 0) := x"000000000"
        );
    end component;
```

35 FTAHBRAM - On-chip SRAM with EDAC and AHB interface

35.1 Overview

The FTAHBRAM core is a version of the AHBRAM core with added Error Detection And Correction (EDAC). The on-chip memory is accessed via an AMBA AHB slave interface. The memory implements a configurable amount of accessible memory (configured via the *kbytes* VHDL generic). Registers are accessed via an AMB APB interface.

The on-chip memory implements volatile memory that is protected by means of Error Detection And Correction (EDAC). One error can be corrected and two errors can be detected, which is performed by using a (32, 7) BCH code or by technology specific protection provided by the target technology RAMs (implementation option, if supported by target technology). Some of the optional features available are single error counter, diagnostic reads and writes and additional pipeline registers. Configuration is performed via a configuration register.

Figure 73 shows a block diagram of the internals of the controller. The block diagram shows the technology agnostic implementation. If target technology specific protection is selected then the encoder and decoder are not implemented in the FTAHBRAM.

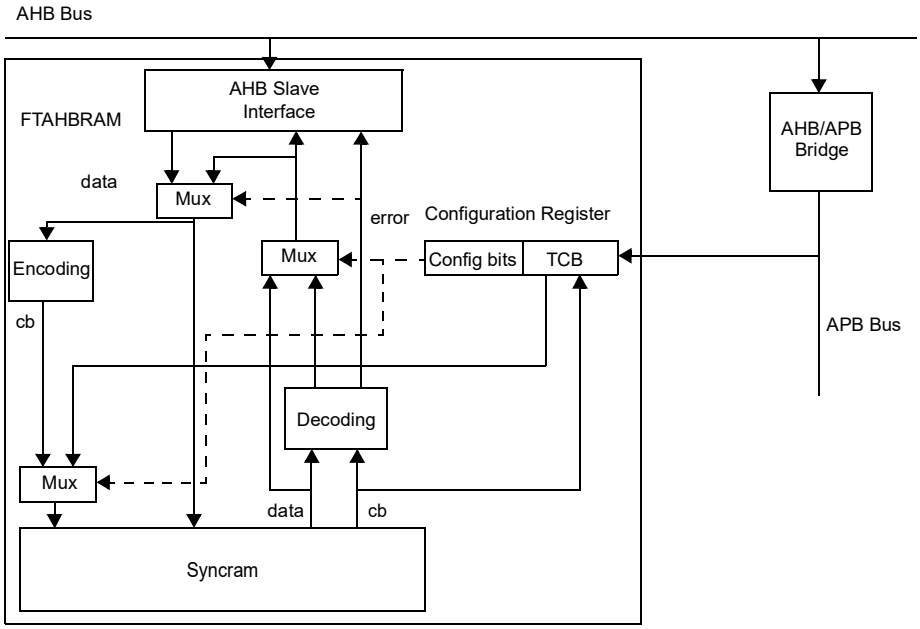


Figure 73. Block diagram

35.2 Operation

35.2.1 Overview

The on-chip fault tolerant memory is accessed through an AMBA AHB slave interface. The maximum AMBA access size supported is configurable through the *maccsz* VHDL generic. The controller supports all access sizes up to *maccsz* and the default value for *maccsz* is set to the maximum bus width configured for GRLIB (AHBDW constant).

The memory address range is configurable with VHDL generics. As for the standard AHB RAM, the memory technology and size is configurable through the *tech* and *kbytes* VHDL generics. The minimum size is 1 KiB and the maximum is technology dependent.

If the core is implemented without AHB pipeline registers then the EDAC functionality can be completely removed by setting the edacen VHDL generic to zero during synthesis. The APB interface is

also removed since it is redundant without EDAC. If AHB pipeline registers are included then EDAC is always enabled and the APB interface is present.

Run-time configuration is done by writing to a configuration register accessed through an AMBA APB interface. The following can be configured during run-time: EDAC can be enabled and disabled. When it is disabled, reads and writes will behave as the standard memory. Read and write diagnostics can be controlled through separate bits. The single error counter can be reset.

35.2.2 Read and write behaviour

If EDAC is disabled (EN bit in configuration register set to 0) write data is passed directly to the memory area and read data will appear on the AHB bus immediately after it arrives from memory. If EDAC is enabled write data is passed to an encoder which outputs a 7-bit checksum. The checksum is stored together with the data in memory and the whole operation is performed without any added waitstates. This applies to word and wider stores (32-bit, 64-bit, 128-bit). If a byte or halfword store is performed, the whole word to which the byte or halfword belongs must first be read from memory (read - modify - write). A new checksum is calculated when the new data is placed in the word and both data and checksum are stored in memory. This is done with 1 - 2 additional waitstates compared to the non EDAC case.

Reads with EDAC disabled are performed with 0 or 1 waitstates while there could also be 2 waitstates when EDAC is enabled. There is no difference between wide, word and subword reads. Table 407 shows a summary of the number of waitstates for the different operations with and without EDAC.

Table 407. Summary of the number of waitstates for the different operations for the memory.

Operation	Waitstates with EDAC Disabled	Waitstates with EDAC Enabled
Read	0 - 1	0 - 2
Word, DWord, 4Word write	0	0
Subword write	0	1 - 2

When EDAC is used, the data is decoded the first cycle after it arrives from the memory and appears on the bus the next cycle if no uncorrectable error is detected. The decoding is done by comparing the stored checksum with a new one which is calculated from the stored data. This decoding is also done during the read phase for a subword write. A so-called syndrome is generated from the comparison between the checksum and it determines the number of errors that occurred. One error is automatically corrected and this situation is not visible on the bus. Two or more detected errors cannot be corrected so the operation is aborted and the required two cycle error response is given on the AHB bus (see the AMBA manual for more details). If no errors are detected data is passed through the decoder unaltered.

35.2.3 Read and write diagnostics

As mentioned earlier the memory provides read and write diagnostics when EDAC is enabled. When write diagnostics are enabled, the calculated checksum is not stored in memory during the write phase. Instead, the TCB field from the configuration register is used. In the same manner, if read diagnostics are enabled, the stored checksum from memory is stored in the TCB field during a read (and also during a subword write). This way, the EDAC functionality can be tested during run-time. Note that checkbits are stored in TCB during reads and subword writes even if a multiple error is detected. Also note that the TCB field contains the check bits for a 32-bit word. If the controller has been implemented with support for wider accesses then it is recommended to load and bypass via TCB using only word accesses. For larger write accesses, the contents of TCB will be written as the checksum for all of the words within the larger access. For wide read accesses, the TCB field will hold the check bits for the least significant word.

35.2.4 Error counter

An additional feature is the single error counter which can be enabled with the *errcnten* VHDL generic or by enabling AHB pipeline registers. A single error counter (SEC) field is present in the configuration register, and is incremented each time a single databit error is encountered (reads or subword writes). The number of bits of this counter is 8, set with the *cntbits* VHDL generic. It is accessed through the configuration register. Each counter bit can be reset to zero by writing a one to it. The counter saturates at the value 2^8 - 1 ($2^{cntbits}$ - 1). For each access where single errors are detected the aramo.ce signal will be driven high for one cycle. This signal should be connected to an AHB status register which stores information and generates interrupts (see the AHB Status register documentation for more information). Note that if the maximum supported access size is 32 bits then only one single error can be detected. To support wider access sizes, the core implements several EDAC protected memories with 32-bit data in parallel. This means that a 64- or 128-bit access can trigger multiple single errors. If this happens then the error counter will be incremented with one.

35.2.5 Auto scrubbing

The core provides an optional auto scrubbing feature that is enabled with the *autoscrub* VHDL generic. If this is enabled, the core will automatically write back the word whenever a correctable error is detected in order to repair the error. Note that this feature can not be enabled if AHB pipeline registers are enabled.

35.2.6 Periodic scrubbing

A optional periodic scrubbing feature is available and enabled with the *perscrub* generic. This feature allows to periodically inject a read and rewrite cycle of one word of memory, then over time cycling through every memory location in order to prevent errors building up over time in areas that are not written often. The scrubbing cycles are done transparently to the surrounding AHB bus system and are only visible by adding additional wait states in case an access comes in during a scrubing iteration.

If auto scrubbing is also enabled, the periodic scrubber will take advantage of this by only reading during the scrub cycle and letting the auto scrubbing logic already present take care of the potential write-back in case of correctable error.

The periodic scrubber can in addition perform initialization of the memory with an all-ones or all-zeros pattern.

When the core is configured with periodic scrubbing support, two additional registers are included for configuration and status of the scrubber. A feature bit in the configuration register indicates that the periodic scrubber is available and can be checked to know at run time whether these registers are available.

35.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 408.*FTAHBRAM registers

APB Address offset	Register
0x0	Configuration Register
0x4	Reserved
0x8	Periodic scrubber configuration (only if perscrub is set)
0xC	Periodic scrubber location (only if perscrub is set)

Table 409. 0x00 - CFG - Configuration Register

31 30	29	28	27 24	23 21	12+8 13	12 10	9	8	7	6 0
DIAG	R	PS	EDACEN	MEMSIZE	SEC	MEMSIZE	WB	RB	EN	TCB
0	0	*	*		0	*	0	0	0	NR
rw	r	r	r		wc	r	*	*	*	rw

28 Indicates that periodic scrubber support is enabled and additional registers are available (1: enabled, 0: disabled)

27 24 Value of edacen VHDL generic.

0: EDAC not implemented
1: Technology agnostic BCH EDAC (traditional FTAHBRAM EDAC)
2: Technology agnostic BCH EDAC, provided by SYNCRAMFT
3: Technology specific EDAC (SECDED)

23: 21 Log2 of the current memory size, bits 5:3 of value. Only used when ahbpipe VHDL generic is non-zero.

12+8: 13 Single error counter (SEC): Incremented each time a single error is corrected (includes errors on checkbits). Each bit can be set to zero by writing a one to it. This feature is only available if the errcnten VHDL generic is set.

12: 10 Log2 of the current memory size, bits 2:0 of value

9 Write Bypass (WB): When set, the TCB field is stored as check bits when a write is performed to the memory.

8 Read Bypass (RB) : When set during a read or subword write, the check bits loaded from memory are stored in the TCB field.

7 EDAC Enable (EN): When set, the EDAC is used otherwise it is bypassed during read and write operations.

If edacen (bits 27:24 of this register) is 2 or 3 then the core always behaves as if it is enabled for write and read timing.

6: 0 Test Check Bits (TCB) : Used as checkbits when the WB bit is set during writes and loaded with the check bits during a read operation when the RB bit is set.

When the core makes use of technology specific EDAC then the behaviour of error injection is different. The TCB checkbits are propagated to the error injection bits of the SYNCRAMFT entity used within FTAHBRAM when the WB field is set to 1. Normally only one or two bits are used and the type of error injection supported is technology specific and described further in the SYNCRAMFT documentation. The RB field has no effect for technology specific EDAC.

Any unused most significant bits are reserved. Always read as '000...0'.

All fields except TCB are initialised at reset. The EDAC is initally disabled (EN = 0), which also applies to diagnostics fiels (RB and WB are zero).

When available, the single error counter (SEC) field is cleared to zero.

Table 410. 0x04 - PSCFG -Periodic Scrubber Configuration Register

31 12	11	10	9	8	7 4	3	2	1	0
RESERVED	SIP	FZ	UE	CE	INTERVAL	IV	IM	LP	SE
0	0	0	nr	nr	1111	0	0	0	0
r	ws	rw	wc	wc	rw	rw	rw	rw	rw

11 Scrubber iteration pending status. This bit reads one when the scrubbing interval timer has expired/wrapped and cleared when the scrub iteration is performed. Can be set to 1 by writing 1 in order to force a scrub iteration immediately.

10 Scrubber freeze. When set to 1, the scrubber will be blocked from starting any new scrub iteration.

9 Flag set if an uncorrectable error is seen during scrub iteration. Cleared by writing 1. Not reset.

8 Flag set if a correctable error is seen during a scrub iteration. Cleared by writing 1. Not reset.

7: 4 Scrubbing interval as a power of two. 0: 1 cycle, 1: 2 cycles, 2: 4 cycles, ..., 15: 32768 cycles.

3 Data pattern used for initialization mode. 0:all-zero, 1:all-ones

Table 410. 0x04 - PSCFG -Periodic Scrubber Configuration Register

2	Initialization mode enable, 0:scrubbing mode, 1:initialization mode Note that scrubber enable needs to also be set to run operation. If looping is enabled, this bit is automatically cleared when the scrubber position wraps around in order to switch over from initialization to scrubbing mode once all memory has been initialized.
1	Scrubbing loop. When set to 1, the scrubber will continue indefinitely. When set to 0, the scrubber will stop once the scrubber location wraps around to zero.
0	Scrubber enable. Setting this to 1 enables the periodic scrubbing or initialization. This bit is cleared when the full memory has been scrubbed (or initialized) and the position counter wraps around, unless the loop bit has been set. When set to 0, the position counter and interval timers are cleared.
7	EDAC Enable (EN): When set, the EDAC is used otherwise it is bypassed during read and write operations. If edacen (bits 27:24 of this register) is 2 or 3 then the core always behaves as if it is enabled for write and read timing.
6: 0	Test Check Bits (TCB) : Used as checkbits when the WB bit is set during writes and loaded with the check bits during a read operation when the RB bit is set. When the core makes use of technology specific EDAC then the behaviour of error injection is different. The TCB checkbits are propagated to the error injection bits of the SYNCRAMFT entity used within FTAHBRAM when the WB field is set to 1. Normally only one or two bits are used and the type of error injection supported is technology specific and described further in the SYNCRAMFT documentation. The RB field has no effect for technology specific EDAC.

Table 411. 0x0C - SCRPOS - Periodic scrubber position register

31 N+1	N W	1 0
RESERVED	SCRUBBER POSITION	RES
0	0	0
r	rw	r

N: 2	Current value of scrubber position register, indicating the location to be scrubbed in the next scrub iteration, or in the currently ongoing scrubber iteration if one is active. The width of this field depends on the configured size of the memory. For wide bus systems, the lower bits will always be 0 (bit 2 for maccsz=64, bits 3:2 for maccsz=128, and so on) The placement of the position field in the register is made so the 32-bit register value as a whole can be viewed as a byte offset into the memory. The scrubber position can be written to update the position, however writes in the middle of a scrub cycle will be ignored in order to avoid incorrect behavior.

35.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x050. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

35.5 Implementation

35.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

35.5.2 Endianness

The core can support both big-endian and little-endian operation, and will change endianness behaviour depending on the sideband signal distributed through the AMBA records (see GRLIB User's Manual).

35.6 Configuration options

Table 412 shows the configuration options of the core (VHDL generics).

*Table 412.*Configuration options

Generic	Function	Allowed range	Default
hindex	Selects which AHB select signal (HSEL) will be used to access the memory.	0 to NAHBMAX-1	0
haddr	ADDR field of the AHB BAR	0 to 16#FFF#	0
hmask	MASK field of the AHB BAR	0 to 16#FFF#	16#FFF#
tech	Memory technology	0 to NTECH	0
kbytes	SRAM size in KiB. The RAM size needs to be a power of two. Otherwise the RAM size will be rounded up to the nearest power of two. For implementations with ahbpipe=1 it is allowed to specify RAM sizes that are of the form 2^x+2^x-1. For example. Specifying a 192 KiB RAM size with ahbpipe = 0 will lead to a 256 KiB RAM area. Specifying a 192 KiB RAM size with ahbpipe = 0 allows a 192 KiB RAM to be implemented. However, specifying a 160 KiB RAM size will still lead to a 192 KiB RAM.	1 to targetdep.	1
pindex	Selects which APB select signal (PSEL) will be used to access the memory configuration registers	0 to NAPBMAX-1	0
paddr	The 12-bit MSB APB address	0 to 16#FFF#	0
pmask	The APB address mask	0 to 16#FFF#	16#FFF#
edacen	Enable and select on-chip EDAC. Must be set to 1 or larger if ahbpipe generic is set to 1. 0: Disabled 1: Technology agnostic BCH EDAC (traditional FTAHBRAM EDAC) 2: Technology agnostic BCH EDAC, provided by SYNCRAMFT 3: Technology specific EDAC (SECDED) Technology specific protection is further documented in the GRLIB-FT User's Manual (grlib-ft.pdf).	0 to 3	0
autoscrub	Automatically store back corrected data with new check-bits during a read when a single error is detected. Is ignored when edacen is deasserted. This generic must be set to 0 if the ahbpipe generic is set to 1.	0 to 1	0
errcnten	Enables a single error counter. This generic must be set to 1 if the ahbpipe generic is set to 1.	0 to 1	0
cntbits	number of bits in the single error counter. This generic must be set to 8 if the ahbpipe generic is set to 1.	1 to 8	1
ahbpipe	Selects to use FTAHBRAM2 architecture. Adds pipeline registers and requires edacen = 1, autoscrub = 0, errcnten = 1, cntbits = 8.	0 to 1	0
testen	Test enable	0 to 1	0
maccsz	Maximum access size supported by core	32 to 128	AHBDW
perscrub	Enables periodic scrubbing functionality in the core	0 to 1	0

35.7 Signal descriptions

Table 413 shows the interface signals of the core (VHDL ports).

*Table 413.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
ARAMO	CE	Output	Single error detected	High
MTESTI**	N/A	Input	Memory BIST input signal	-
MTESTO**	N/A	Output	Memory BIST output signal	-
MTESTCLK**	N/A	Input	Memory BIST clock	-

* see GRLIB IP Library User's Manual

** not available in FPGA releases

The aramo.ce signal is normally used to generate interrupts which starts an interrupt routine that corrects errors. Since this is not necessary when autoscrubbing is enabled, aramo.ce should not be connected to an AHB status register or the interrupt should be disabled in the interrupt controller

35.8 Library dependencies

Tabel 414 shows libraries used when instantiating the core (VHDL libraries).

*Table 414.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Signals and component declaration

35.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
library gaisler;

use grlib.amba.all;
use gaisler.misc.all;

entity ftram_ex is
  port(
    rst : std_ulogic;
    clk : std_ulogic;

    .... --others signals
  );
end;

architecture rtl of ftram_ex is
```

```
--AMBA signals
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_type;
signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector;



--other needed signals here
signal stati   : ahbstat_in_type;
signal aramo   : ahbram_out_type;

begin

--other component instantiations here
...

-- AHB Status Register
  astat0 : ahbstat generic map(pindex => 13, paddr => 13, pirq => 11,
    nftslv => 3)
    port map(rstn, clkm, ahbmi, ahbso, stati, apbi, apbo(13));
    stati.cerror(1 to NAHBSLV-1) <= (others => '0');

--FT AHB RAM
a0 : ftahbram generic map(hindex => 1, haddr => 1, tech => inferred,
  kbytes => 64, pindex => 4, paddr => 4, edacen => 1, autoscrub => 0,
  errcnt => 1, cntbits => 4)
  port map(rst, clk, ahbsi, ahbso(1), apbi, apbo(4), aramo);
  stati.cerror(0) <= aramo.ce;

end architecture;
```

36 FTMCTRL - 8/16/32-bit Memory Controller with EDAC

36.1 Overview

The FTMCTRL combined 8/16/32-bit memory controller provides a bridge between external memory and the AHB bus. The memory controller can handle four types of devices: PROM, asynchronous static ram (SRAM), synchronous dynamic ram (SDRAM) and memory mapped I/O devices (IO). The PROM, SRAM and SDRAM areas can be EDAC-protected using a (39,7) BCH code. The BCH code provides single-error correction and double-error detection for each 32-bit memory word.

The SDRAM area can optionally also be protected using Reed-Solomon coding. In this case a 16-bit checksum is used for each 32-bit word, and any two adjacent 4-bit (nibble) errors can be corrected.

The EDAC capability is determined through a VHDL generic.

The memory controller is configured through three configuration registers accessible via an APB bus interface. The PROM, IO, and SRAM external data bus can be configured in 8-, 16-, or 32-bit mode, depending on application requirements. The controller decodes three address spaces on the AHB bus (PROM, IO, and SRAM/SDRAM). The addresses are determined through VHDL generics. The IO area is marked as non-cacheable in the core's AMBA plug'n'play information record.

External chip-selects are provided for up to four PROM banks, one IO bank, five SRAM banks and two SDRAM banks. Figure 74 below shows how the connection to the different device types is made.

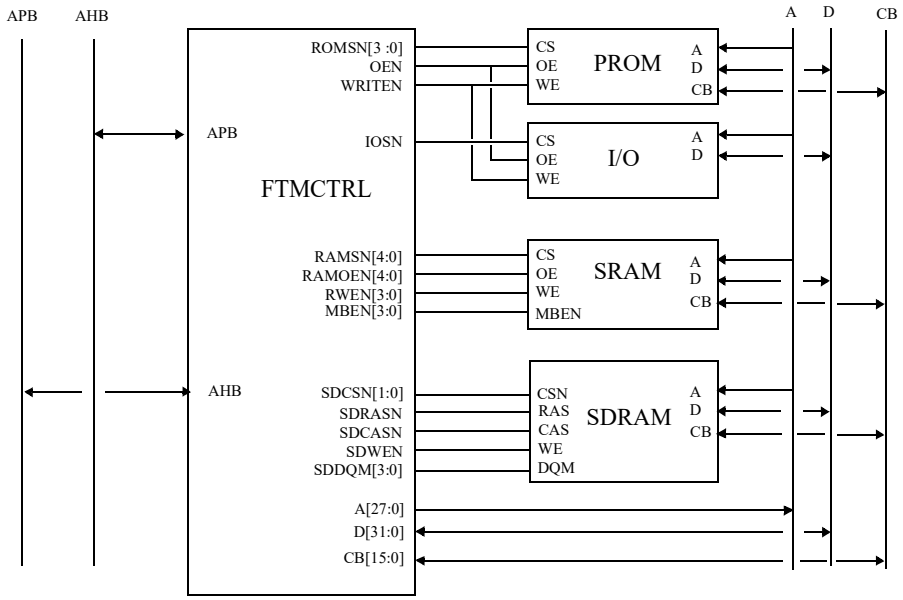


Figure 74. FTMCTRL connected to different types of 32+cb-bit memory devices

36.2 PROM access

Up to four PROM chip-select signals are provided for the PROM area, ROMSN[3:0]. There are two modes: one with two chip-select signals and one with four. The size of the banks can be set in binary steps from 16KiB to 256MiB. If the AHB memory area assigned to the memory controller for PROM accesses is larger than the combined size of the memory banks then the PROM memory area will wrap, starting with the first chip-select being asserted again when accessing addresses higher than the last decoded bank.

A read access to PROM consists of two data cycles and between 0 and 30 waitstates (in the default configuration, see *wsshift* VHDL generic documentation for details). The read data (and optional

EDAC check-bits) are latched on the rising edge of the clock on the last data cycle. On non-consecutive accesses, a idle cycle is placed between the read cycles to prevent bus contention due to slow turn-off time of PROM devices. Figure 75 shows the basic read cycle waveform (zero waitstate) for non-consecutive PROM reads. Note that the address is undefined in the idle cycle. Figure 76 shows the timing for consecutive cycles (zero waitstate). Waitstates are added by extending the data2 phase. This is shown in figure 77 and applies to both consecutive and non-consecutive cycles. Only an even number of waitstates can be assigned to the PROM area.

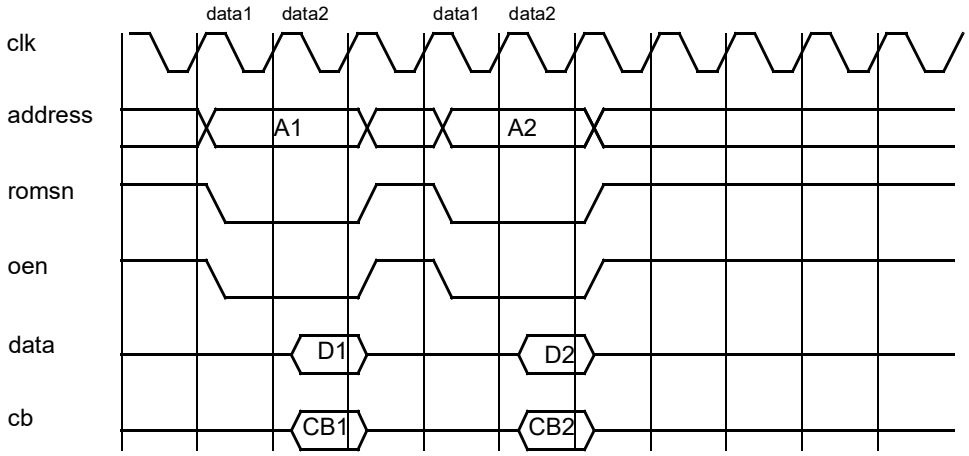


Figure 75. Prom non-consecutive read cyclecs.

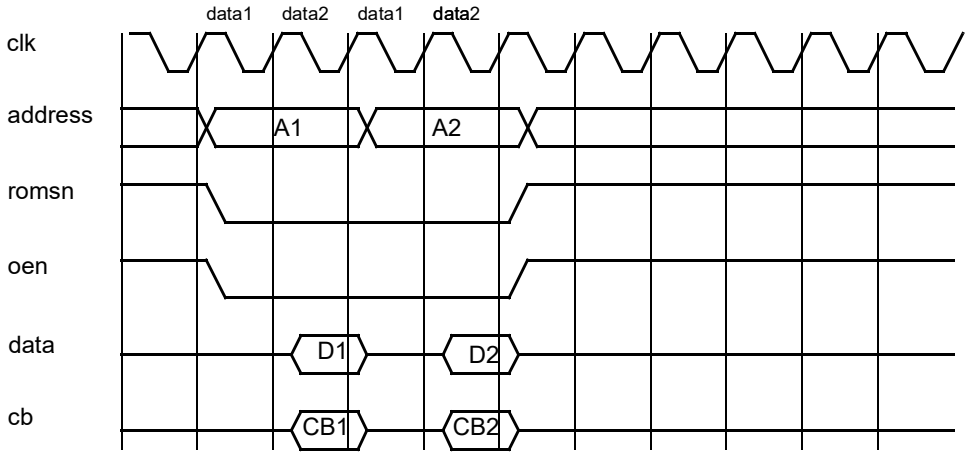


Figure 76. Prom consecutive read cyclecs.

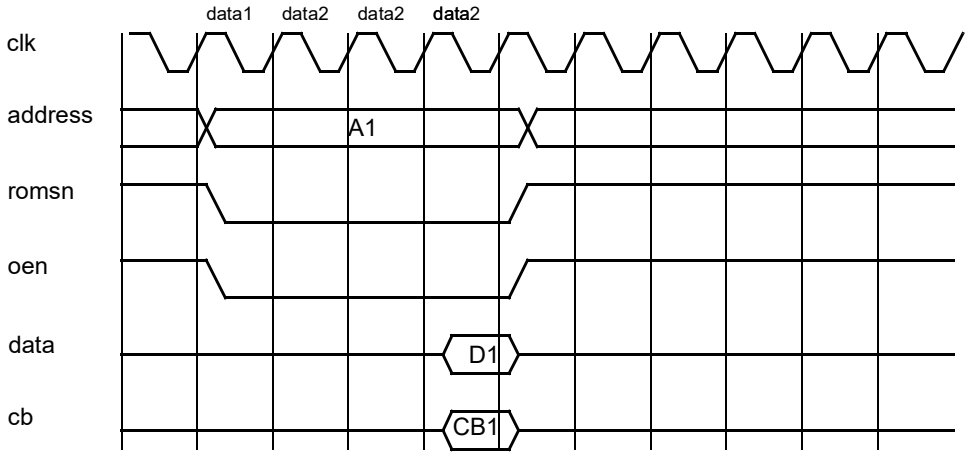


Figure 77. Prom read access with two waitstates.

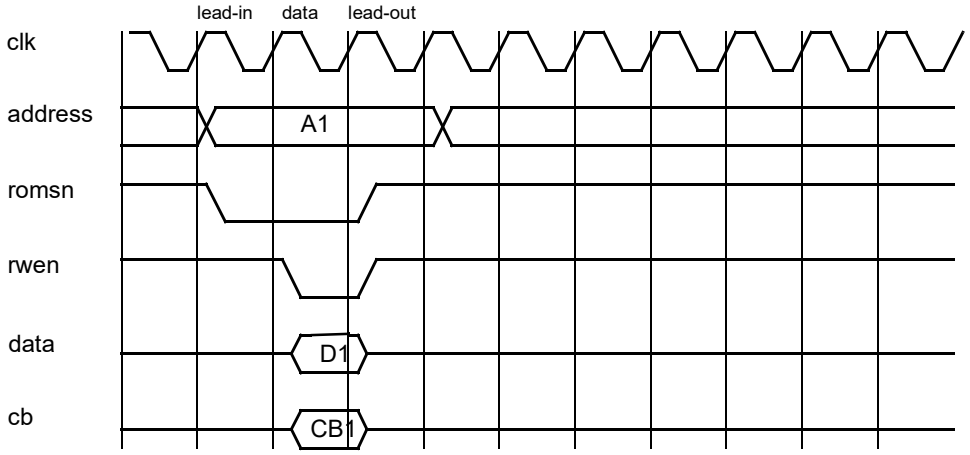


Figure 78. Prom write cycle (0-waitstates)

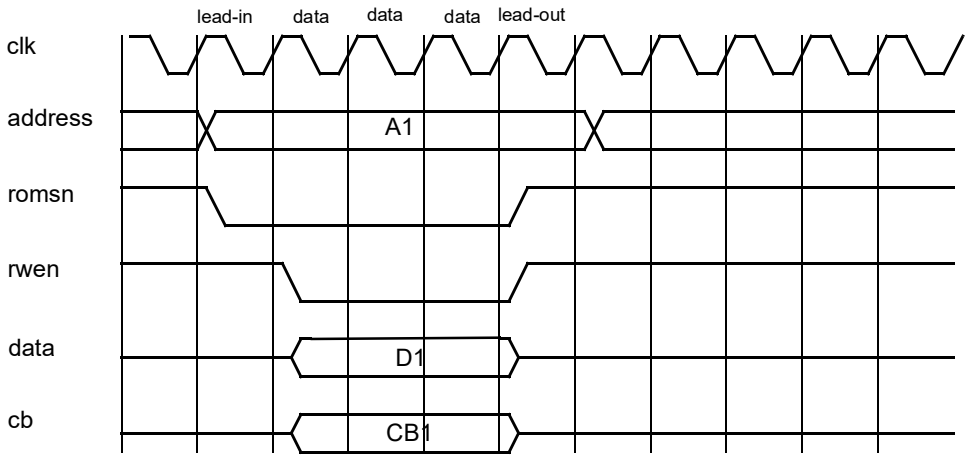


Figure 79. Prom write cycle (2-waitstates)

36.3 Memory mapped IO

Accesses to IO have similar timing as PROM accesses. The IO select (IOSN) and output enable (OEN) signals are delayed one clock to provide stable address before IOSN is asserted. All accesses are performed as non-consecutive accesses as shown in figure 80. The data2 phase is extended when waitstates are added.

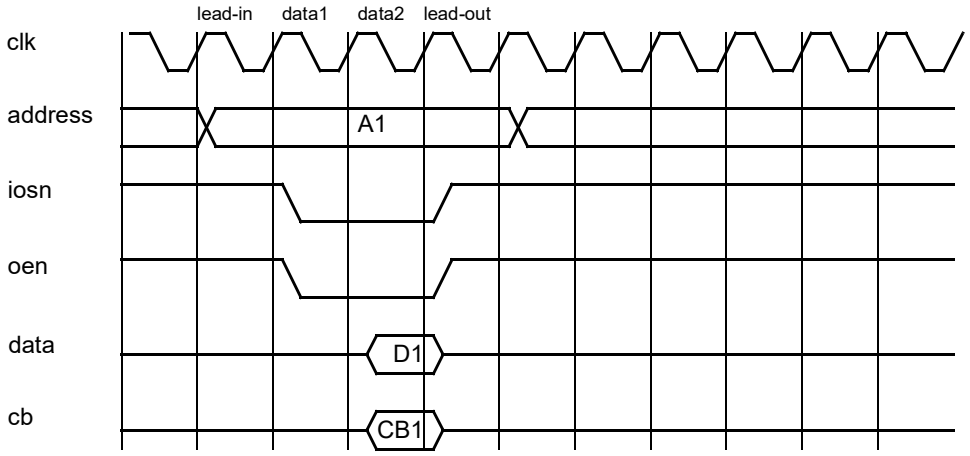


Figure 80. I/O read cycle (0-waitstates)

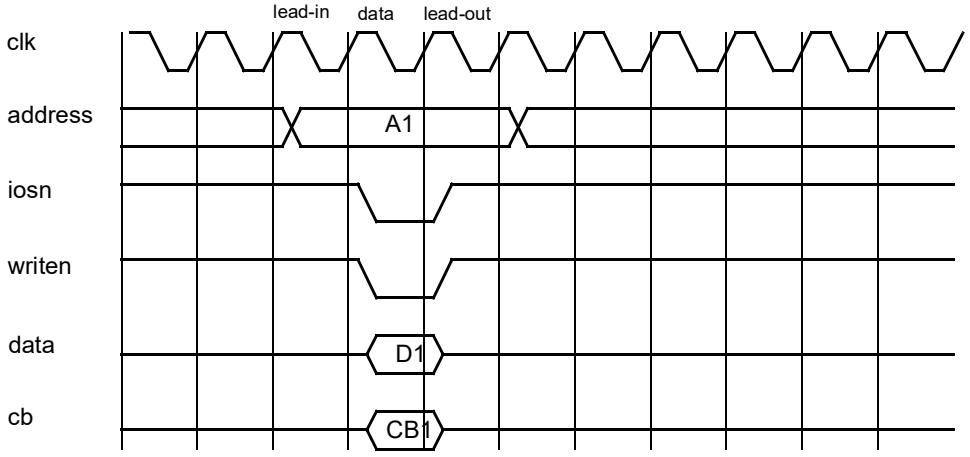


Figure 81. I/O write cycle (0-waitstates)

36.4 SRAM access

The SRAM area is divided on up to five RAM banks. The size of banks 1-4 (RAMSN[3:0]) is programmed in the RAM bank-size field (MCFG2[12:9]) and can be set in binary steps from 8KiB to 256MiB. The fifth bank (RAMSN[4]) decodes the upper 512MiB (controlled by means of the *sdrasel* VHDL generic) and cannot be used simultaneously with SDRAM memory. A read access to SRAM consists of two data cycles and between zero and three waitstates (in the default configuration, see *wsshift* VHDL generic documentation for details). The read data (and optional EDAC check-bits) are latched on the rising edge of the clock on the last data cycle. Accesses to RAMSN[4] can further be stretched by de-asserting BRDYN until the data is available. On non-consecutive accesses, a idle cycle is added after a read cycle to prevent bus contention due to slow turn-off time of memories. Fig-

ure 82 shows the basic read cycle waveform (zero waitstate). Waitstates are added in the same way as for PROM in figure 77.

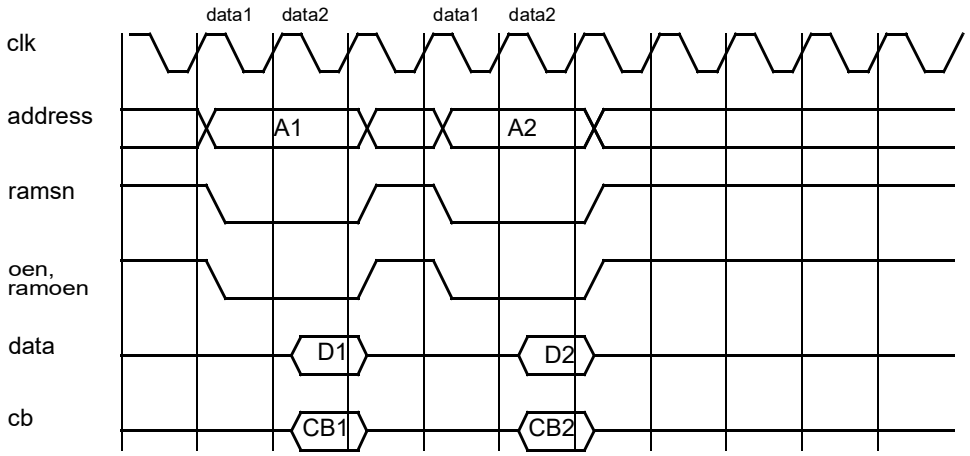


Figure 82. Sram non-consecutive read cyclecs.

For read accesses to RAMSN[4:0], a separate output enable signal (RAMOEN[n]) is provided for each RAM bank and only asserted when that bank is selected. A write access is similar to the read access but takes a minimum of three cycles. Waitstates are added in the same way as for PROM.

Each byte lane has an individual write strobe to allow efficient byte and half-word writes. If the memory uses a common write strobe for the full 16- or 32-bit data, the read-modify-write bit MCFG2 should be set to enable read-modify-write cycles for sub-word writes.

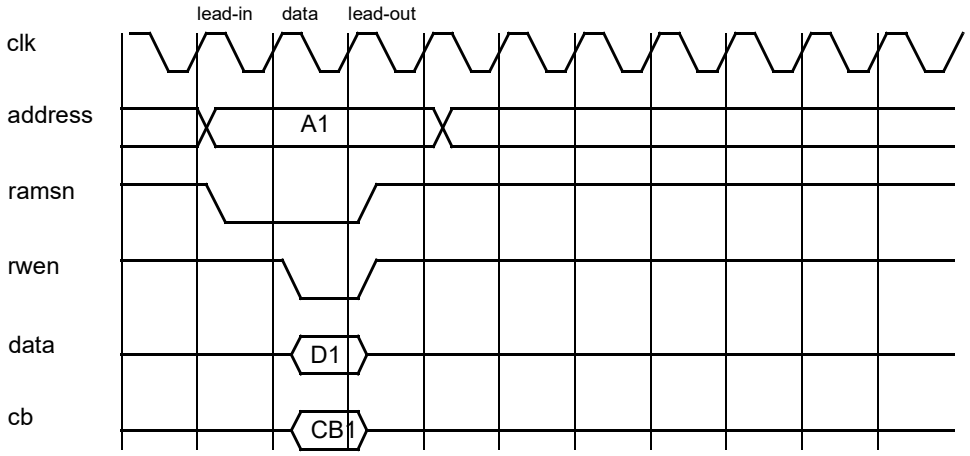


Figure 83. Sram write cycle (0-waitstates)

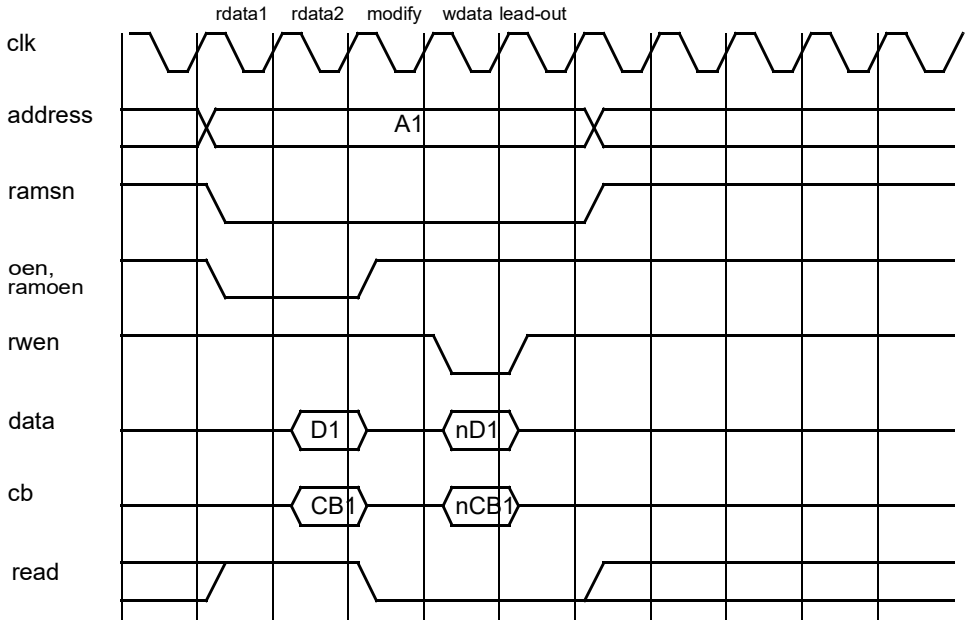


Figure 84. Sram read-modify-write cycle (0-waitstates)

36.5 8-bit and 16-bit PROM and SRAM access

To support applications with low memory and performance requirements efficiently, the SRAM and PROM areas can be individually configured for 8- or 16-bit operation by programming the ROM and RAM width fields in the memory configuration registers. Since reads to memory are always done on 32-bit word basis, read access to 8-bit memory will be transformed in a burst of four read cycles while access to 16-bit memory will generate a burst of two 16-bit reads. During writes, only the necessary bytes will be written. Figure 85 shows an interface example with 8-bit PROM and 8-bit SRAM. Figure 86 shows an example of a 16-bit memory interface.

There is an erratum associated with sub-word (byte or half-word, 8- or 16-bit) writes to 8-bit SRAM areas. When SDRAM and 8-bit SRAM with EDAC are simultaneously enabled, there is a condition where a sub-word (byte or half-word, 8- or 16-bit) write to SRAM that is in between two SDRAM accesses will be performed incorrectly. Refer to section 36.15.2 for further information.

All possible combinations of width, EDAC, and RMW are not supported. The supported combinations are given in table 415, and the behavior of setting an unsupported combination is undefined. It is not allowed to set the ROM or RAM width fields to 8-bit or 16-bit width if the core does not implement support for these widths.

*Table 415.*FTMCTRL supported SRAM and PROM configurations

PROM/SRAM bus width	RWEN resolution (SRAM)	EDAC	RMW bit (SRAM)	Core configuration
8	Bus width	None	0	8-bit support
8	Bus width	BCH	1	8-bit support, EDAC
16	Byte	None	0	16-bit support
16	Bus width	None	1	16-bit support
32	Byte	None	0	
32	Bus width	None	1	
32+7	Bus width	BCH	1	EDAC support

8-bit width support is set with *ram8* VHDL generic and 16-bit width support is set with *ram16* VHDL generics.

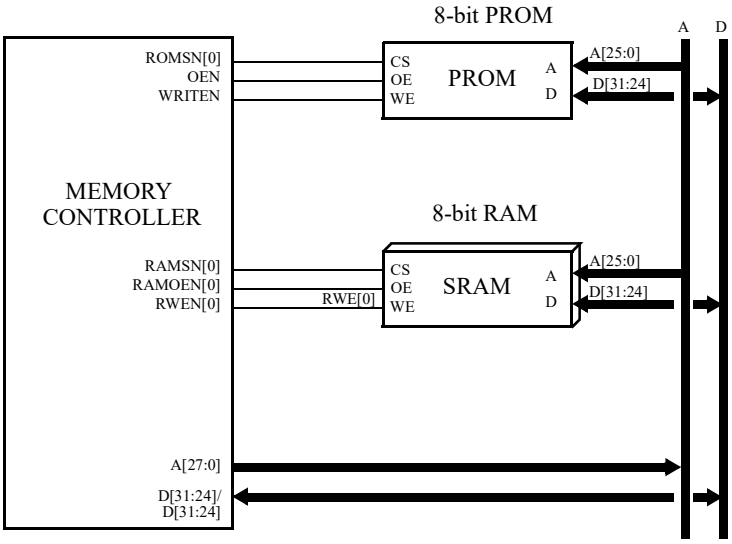


Figure 85. 8-bit memory interface example

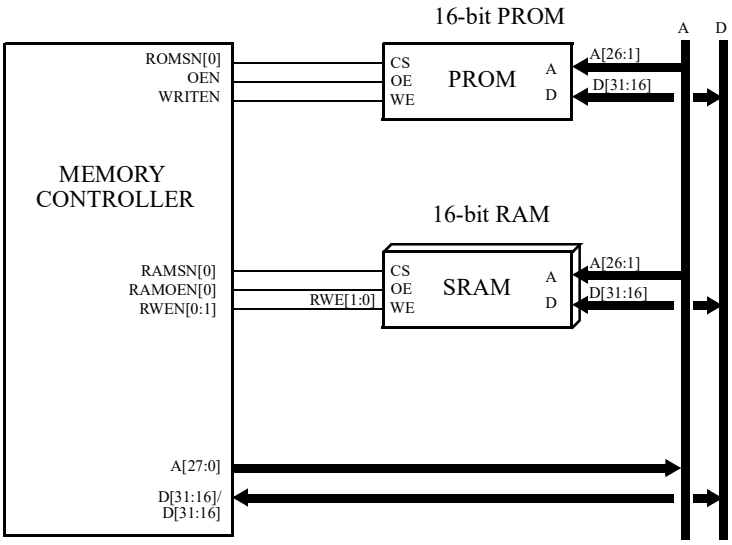


Figure 86. 16-bit memory interface example

In 8-bit mode, the PROM/SRAM devices should be connected to the MSB byte of the data bus (D[31:24]). The LSB address bus should be used for addressing (A[25:0]). In 16-bit mode, D[31:16] should be used as data bus, and A[26:1] as address bus.

36.6 8- and 16-bit I/O access

Similar to the PROM/SRAM areas, the IO area can also be configured to 8- or 16-bits mode. However, the I/O device will NOT be accessed by multiple 8/16 bits accesses as the memory areas, but only with one single access just as in 32-bit mode. To access an IO device on an 8-bit bus, only byte accesses should be used (LDUB/STB instructions for the CPU). To accesses an IO device on a 16-bit bus, only halfword accesses should be used (LDUH/STH instructions for the CPU).

To access the I/O-area in 8- or 16-bit mode, *ram8* VHDL generic or *ram16* VHDL generic must be set respectively.

36.7 Burst cycles

To improve the bandwidth of the memory bus, accesses to consecutive addresses can be performed in burst mode. Burst transfers will be generated when the memory controller is accessed using an AHB burst request. These includes instruction cache-line fills, double loads and double stores. The timing of a burst cycle is identical to the programmed basic cycle with the exception that during read cycles, the idle cycle will only occurs after the last transfer. Burst cycles will not be generated to the IO area.

Only word (HSIZE = “010”) bursts of incremental type (HBURST=INCR, INCR4, INCR8 or INCR16) are supported.

36.8 SDRAM access

36.8.1 General

Synchronous dynamic RAM (SDRAM) access is supported to two banks of PC100/PC133 compatible devices. This is implemented by a special version of the SDCTRL SDRAM controller core from Frontgrade Gaisler, which is optionally instantiated as a sub-block. The SDRAM controller supports 64M, 256M and 512M devices with 8 - 12 column-address bits, and up to 13 row-address bits. The size of the two banks can be programmed in binary steps between 4MiB and 512MiB. The operation of the SDRAM controller is controlled through MCFG2 and MCFG3 (see below). Both 32- and 64-bit data bus width is supported, allowing the interface of 64-bit DIMM modules. The memory controller can be configured to use either a shared or separate bus connecting the controller and SDRAM devices.

36.8.2 Address mapping

The two SDRAM chip-select signals are decoded. SDRAM area is mapped into the upper half of the RAM area defined by BAR2 register, and cannot be used simultaneously with fifth SRAM bank (RAMSN[4]). When the SDRAM enable bit is set in MCFG2, the controller is enabled and mapped into upper half of the RAM area as long as the SRAM disable bit is not set. If the SRAM disable bit is set, all access to SRAM is disabled and the SDRAM banks are mapped into the lower half of the RAM area.

36.8.3 Initialisation

When the SDRAM controller is enabled, it automatically performs the SDRAM initialisation sequence of PRECHARGE, 8x AUTO-REFRESH and LOAD-MODE-REG on both banks simultaneously. The controller programs the SDRAM to use single location access on write. The controller programs the SDRAM to use line burst of length 8 when *pageburst* VHDL generic is 0. The controller programs the SDRAM to use page burst when *pageburst* VHDL generic is 1. The controller programs the SDRAM to use page burst or line burst of length 8, selectable via the MCFG2 register, when *pageburst* VHDL generic is 2.

36.8.4 Configurable SDRAM timing parameters

To provide optimum access cycles for different SDRAM devices (and at different frequencies), three SDRAM parameters can be programmed through memory configuration register 2 (MCFG2): TCAS, TRP and TRFCD. The value of these field affects the SDRAM timing as described in table 416.

*Table 416.*SDRAM programmable minimum timing parameters

SDRAM timing parameter	Minimum timing (clocks)
CAS latency, RAS/CAS delay (t_{CAS}, t_{RCD})	TCAS + 2
Precharge to activate (t_{RP})	TRP + 2
Auto-refresh command period (t_{RFC})	TRFC + 3
Activate to precharge (t_{RAS})	TRFC + 1
Activate to Activate (t_{RC})	TRP + TRFC + 4

If the TCAS, TRP and TRFC are programmed such that the PC100/133 specifications are fulfilled, the remaining SDRAM timing parameters will also be met. The table below shows typical settings for 100 and 133 MHz operation and the resulting SDRAM timing (in ns):

*Table 417.*SDRAM example programming

SDRAM settings	t_{CAS}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}
100 MHz, CL=2; TRP=0, TCAS=0, TRFC=4	20	80	20	70	50
100 MHz, CL=3; TRP=0, TCAS=1, TRFC=4	30	80	20	70	50
133 MHz, CL=2; TRP=1, TCAS=0, TRFC=6	15	82	22	67	52
133 MHz, CL=3; TRP=1, TCAS=1, TRFC=6	22	82	22	67	52

36.8.5 Refresh

The SDRAM controller contains a refresh function that periodically issues an AUTO-REFRESH command to both SDRAM banks. The period between the commands (in clock periods) is programmed in the refresh counter reload field in the MCFG3 register. Depending on SDRAM type, the required period is typically 7.8 or 15.6 µs (corresponding to 780 or 1560 clocks at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by setting bit 31 in MCFG2.

36.8.6 SDRAM commands

The controller can issue three SDRAM commands by writing to the SDRAM command field in MCFG2: PRE-CHARGE, AUTO-REFRESH and LOAD-MODE-REG (LMR). If the LMR command is issued, the CAS delay as programmed in MCFG2 will be used. Line burst of length 8 will be set for read when *pageburst* VHDL generic is 0. Page burst will be set for read when *pageburst* VHDL generic is 1. Page burst or line burst of length 8, selectable via the MCFG2 register will be set, when *pageburst* VHDL generic is 2. Remaining fields are fixed: single location write, sequential burst. The command field will be cleared after a command has been executed. When changing the value of the CAS delay, a LOAD-MODE-REGISTER command should be generated at the same time. NOTE: when issuing SDRAM commands, the SDRAM refresh must be disabled.

36.8.7 Read cycles

A read cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a READ command after the programmed CAS delay. A read burst is performed if a burst access has been requested on the AHB bus. The read cycle is terminated with a PRE-CHARGE command, no banks are left open between two accesses.

36.8.8 Write cycles

Write cycles are performed similarly to read cycles, with the difference that WRITE commands are issued after activation. A write burst on the AHB bus will generate a burst of write commands without idle cycles in-between.

After the WRITE command has completed, if there is an immediately following read or write access (not RMW) to the same 1KiB page on the AHB bus, this access is performed during the same access cycle without closing and re-opening the row.

36.8.9 Read-modify-write cycles

If EDAC is enabled and a byte or half-word write is performed, the controller will perform a read-modify-write cycle to update the checkbits correctly. This is done by performing an ACTIVATE command, followed by READ, WRITE and PRE-CHARGE. The write command interrupts the read burst and the data mask signals will be raised two cycles before this happens as required by the SDRAM standard.

36.8.10 Address bus

The memory controller can be configured to either share the address and data buses with the SRAM, or to use separate address and data buses. When the buses are shared, the address bus of the SDRAMs should be connected to A[14:2], the bank address to A[16:15]. The MSB part of A[14:2] can be left unconnected if not used. When separate buses are used, the SDRAM address bus should be connected to SA[12:0] and the bank address to SA[14:13].

36.8.11 Data bus

SDRAM can be connected to the memory controller through the common or separate data bus. If the separate bus is used the width is configurable to 32 or 64 bits. 64-bit data bus allows the 64-bit SDRAM devices to be connected using the full data capacity of the devices. 64-bit SDRAM devices can be connected to 32-bit data bus if 64-bit data bus is not available but in this case only half the full data capacity will be used. There is a drive signal vector and separate data vector available for SDRAM. The drive vector has one drive signal for each data bit. These signals can be used to remove timing problems with the output delay when a separate SDRAM bus is used.

36.8.12 Clocking

The SDRAM controller is designed for an external SDRAM clock that is in phase or slightly earlier than the internal AHB clock. This provides the maximum margin for setup and hold on the external signals, and allows highest possible frequency. For Xilinx and Altera device, the GRLIB Clock Generator (CLKGEN) can be configured to produce a properly synchronized SDRAM clock. For other FPGA targets, the custom clock synchronization must be designed. For ASIC targets, the SDRAM clock can be derived from the AHB clock with proper delay adjustments during place&route.

36.8.13 Initialisation

Each time the SDRAM is enabled (bit 14 in MCFG2), an SDRAM initialisation sequence will be sent to both SDRAM banks. The sequence consists of one PRECHARGE, eight AUTO-REFRESH and one LOAD-COMMAND-REGISTER command.

36.9 Memory EDAC

Note that there is an erratum associated with sub-word writes to EDAC protected 8-bit RAM areas when SDRAM is also enabled. Refer to section 36.15.2 for details.

36.9.1 BCH EDAC

The FTMCTRL is provided with an BCH EDAC that can correct one error and detect two errors in a 32-bit word. For each word, a 7-bit checksum is generated according to the equations below. A correctable error will be handled transparently by the memory controller, but adding one waitstate to the access. If an un-correctable error (double-error) is detected, the current AHB cycle will end with an error response. The EDAC can be used during access to PROM, SRAM and SDRAM areas by setting the corresponding EDAC enable bits in the MCFG3 register. The equations below show how the EDAC checkbits are generated:

```
CB0 = D0 ^ D4 ^ D6 ^ D7 ^ D8 ^ D9 ^ D11 ^ D14 ^ D17 ^ D18 ^ D19 ^ D21 ^ D26 ^ D28 ^ D29 ^ D31
CB1 = D0 ^ D1 ^ D2 ^ D4 ^ D6 ^ D8 ^ D10 ^ D12 ^ D16 ^ D17 ^ D18 ^ D20 ^ D22 ^ D24 ^ D26 ^ D28
CB2 = D0 ^ D3 ^ D4 ^ D7 ^ D9 ^ D10 ^ D13 ^ D15 ^ D16 ^ D19 ^ D20 ^ D23 ^ D25 ^ D26 ^ D29 ^ D31
CB3 = D0 ^ D1 ^ D5 ^ D6 ^ D7 ^ D11 ^ D12 ^ D13 ^ D16 ^ D17 ^ D21 ^ D22 ^ D23 ^ D27 ^ D28 ^ D29
CB4 = D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D14 ^ D15 ^ D18 ^ D19 ^ D20 ^ D21 ^ D22 ^ D23 ^ D30 ^ D31
CB5 = D8 ^ D9 ^ D10 ^ D11 ^ D12 ^ D13 ^ D14 ^ D15 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
CB6 = D0 ^ D1 ^ D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
```

If the SRAM is configured in 8-bit mode, the EDAC checkbit bus (CB[7:0]) is not used but it is still possible to use EDAC protection. Data is always accessed as words (4 bytes at a time) and the corresponding checkbits are located at the address acquired by inverting the word address (bits 2 to 27) and using it as a byte address. The same chip-select is kept active. A word written as four bytes to addresses 0, 1, 2, 3 will have its checkbits at address 0xFFFFFFF, addresses 4, 5, 6, 7 at 0xFFFFFFE and so on. All the bits up to the maximum bank size will be inverted while the same chip-select is always asserted. This way all the bank sizes can be supported and no memory will be unused (except for a maximum of 4 byte in the gap between the data and checkbit area). A read access will automatically read the four data bytes individually from the nominal addresses and the EDAC checkbit byte from the top part of the bank. A write cycle is performed the same way. Byte or half-word write accesses will result in an automatic read-modify-write access where 4 data bytes and the checkbit byte are firstly read, and then 4 data bytes and the newly calculated checkbit byte are writen back to the memory. This 8-bit mode applies to SRAM while SDRAM always uses 32-bit accesses. The size of the memory bank is determined from the settings in MCFG2. The EDAC cannot be used on memory areas configured in 16-bit mode.

If the ROM is configured in 8-bit mode, EDAC protection is provided in a similar way as for the SRAM memory described above. The difference is that write accesses are not being handled automatically. Instead, write accesses must only be performed as individual byte accesses by the software, writing one byte at a time, and the corresponding checkbit byte must be calculated and be written to the correct location by the software.

NOTE: when the EDAC is enabled in 8-bit bus mode, only the first bank select (RAMSN[0], PROMSN[0]) can be used.

The operation of the EDAC can be tested trough the MCFG3 register. If the WB (write bypass) bit is set, the value in the TCB field will replace the normal checkbits during memory write cycles. If the RB (read bypass) is set, the memory checkbits of the loaded data will be stored in the TCB field during memory read cycles. NOTE: when the EDAC is enabled, the RMW bit in memory configuration register 2 must be set.

Data access timing with EDAC enabled is identical to access without EDAC, if the *edac* VHDL generic is set to 1. To improve timing of the HREADY output, a pipeline stage can be inserted in the EDAC error detection by setting the *edac* VHDL generic to 2. One clock extra latency will then occur on single word reads, or on the first data word in a burst.

EDAC is not supported for 64-bit wide SDRAM data buses.

36.9.2 Reed-Solomon EDAC

The Reed-Solomon EDAC provides block error correction, and is capable of correcting up to two 4-bit nibble errors in a 32-bit data word or 16-bit checksum. The Reed-Solomon EDAC can be enabled

for the SDRAM area only, and uses a 16-bit checksum. Operation and timing is identical to the BCH EDAC with the pipeline option enabled. The Reed-Solomon EDAC is enabled by setting the RSE and RE bits in MCFG3, and the RMW bit in MCFG2. The Reed-Solomon EDAC is not supported for 64-bit wide SDRAM buses.

The Reed-Solomon data symbols are 4-bit wide, represented as GF(2^4). The basic Reed-Solomon code is a shortened RS(15, 13, 2) code, represented as RS(6, 4, 2). It has the capability to detect and correct a single symbol error anywhere in the codeword. The EDAC implements an interleaved RS(6, 4, 2) code where the overall data is represented as 32 bits and the overall checksum is represented as 16 bits. The codewords are interleaved nibble-wise. The interleaved code can correct two 4-bit errors when each error is located in a nibble and not in the same original RS(6, 4, 2) codeword.

The Reed-Solomon RS(15, 13, 2) code has the following definition:

- there are 4 bits per symbol;
- there are 15 symbols per codeword;
- the code is systematic;
- the code can correct one symbol error per codeword;
- the field polynomial is

$$f(x) = x^4 + x + 1$$

- the code generator polynomial is

$$g(x) = \prod_{i=7}^{8} (x + \alpha^i) = \sum_{j=0}^{2} g_j \cdot x^j$$

  for which the highest power of x is stored first;
- a codeword is defined as 15 symbols:

  $c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}$

  where c_0 to c_{12} represent information symbols and c_{13} to c_{14} represent check symbols.

The shortened and interleaved RS(6, 4, 2) code has the following definition:

- the codeword length is shortened to 4 information symbols and 2 check symbols and as follows:

  $c_0 = c_1 = c_2 = c_3 = c_4 = c_5 = c_6 = c_7 = c_8 = 0$

  where the above information symbols are suppressed or virtually filled with zeros;
- two codewords are interleaved (i.e. interleaved depth I=2) with the following mapping to the 32-bit data and 16-bit checksum, were $c_{i,j}$ is a symbol with codeword index i and symbol index j:

  $c_{0,9}$ = sd[31:28]

  $c_{1,9}$ = sd[27:24]

  $c_{0,10}$ = sd[23:20]

  $c_{1,10}$ = sd[19:16]

  $c_{0,11}$ = sd[15:12]

  $c_{1,11}$ = sd[11:8]

  $c_{0,12}$ = sd[7:4]

  $c_{1,12}$ = sd[3:0]

  $c_{0,13}$ = scb[15:12]

$c_{1,13}$ = scb[11:8]

$c_{0,14}$ = scb[7:4]

$c_{1,14}$ = scb[3:0]

where SD[ ] is interchanable with DATA[] and SCB[ ] is interchangable with CB[ ]

Note that the FTMCTRL must have the *edac* VHDL generic set to 3 to enable the RS EDAC functionality. The Reed-Solomon EDAC is not supported for 64-bit wide SDRAM buses.

36.9.3 EDAC Error reporting

As mentioned above an un-correctable error results in an AHB error response which can be monitored on the bus. Correctable errors however are handled transparently and are not visible on the AHB bus. A sideband signal is provided which is asserted during one clock cycle for each access for which a correctable error is detected. This can be used for providing an external scrubbing mechanism and/or statistics. The correctable error signal is most commonly connected to the AHB status register which monitors both this signal and error responses on the bus. Please see the AHB status register section for more information.

36.10 Bus Ready signalling

The BRDYN signal can be used to stretch all types of access cycles to the PROM, I/O area and the SRAM area decoded by RAMSN[4]. This covers read and write accesses in general, and additionally read-modify-write accesses to the SRAM area. The accesses will always have at least the pre-programmed number of waitstates as defined in memory configuration registers 1 & 2, but will be further stretched until BRDYN is asserted. BRDYN should be asserted in the cycle preceding the last one. If bit 29 in MCFG1 is set, BRDYN can be asserted asynchronously with the system clock. In this case, the read data must be kept stable until the de-assertion of OEN/RAMOEN and BRDYN must be asserted for at least 1.5 clock cycle. The use of BRDYN can be enabled separately for the PROM, I/O and RAMSN[4] areas. It is recommended that BRDYN is asserted until the corresponding chip select signal is de-asserted, to ensure that the access has been properly completed and avoiding the system to stall.

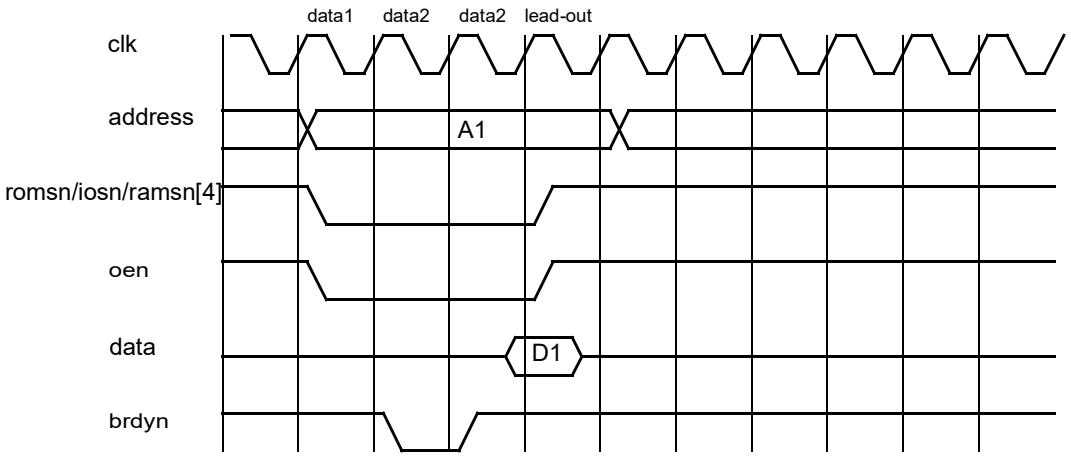


Figure 87. READ cycle with one extra data2 cycle added with BRDYN (synchronous sampling). Lead-out cycle is only applicable for I/O accesses.

Figure 88 shows the use of BRDYN with asynchronous sampling. BRDYN is kept asserted for more than 1.5 clock-cycle. Two synchronization registers are used so it will take at least one additional cycle from when BRDYN is first asserted until it is visible internally. In figure 88 one cycle is added to the data2 phase.

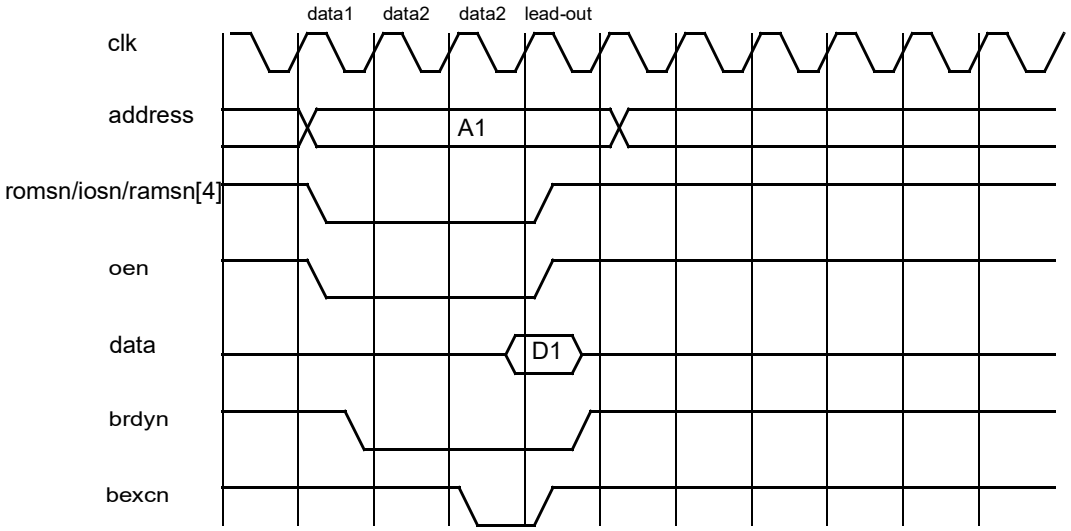


Figure 88. BRDYN (asynchronous) sampling and BEXCN timing. Lead-out cycle is only applicable for I/O-accesses.

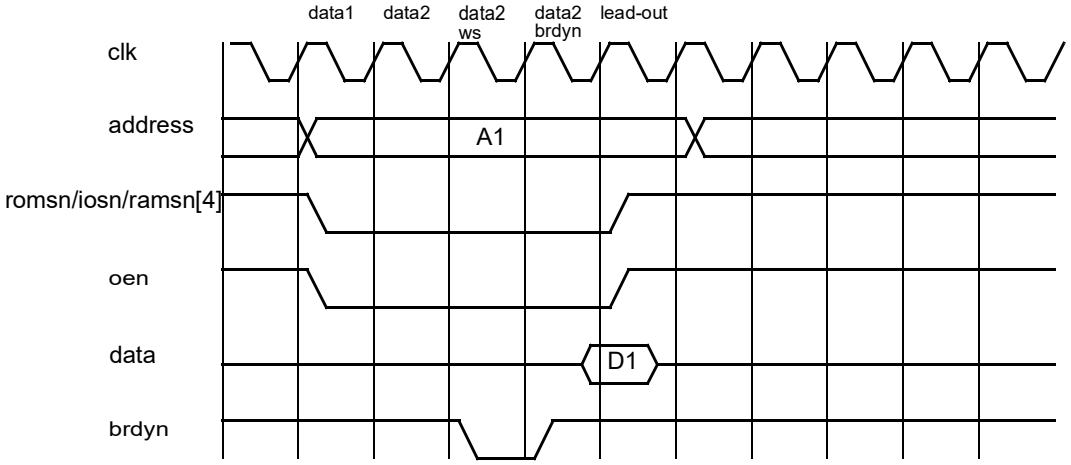


Figure 89. Read cycle with one waitstate (configured) and one BRDYN generated waitstate (synchronous sampling).

If burst accesses and BRDYN signaling are to be used together, special care needs to be taken to make sure BRDYN is raised between the separate accesses of the burst. The controller does not raise the select and OEN signal (in the read case) between accesses during the burst so if BRDYN is kept asserted until the select signal is raised, all remaining accesses in the burst will finish with the configured fixed number of wait states.

The core can optionally be implemented with a bus ready timeout counter. The counter value and counter reload value are then available in MCFG7. The counter will be reloaded whenever the bus ready signal is low (asserted). If the reload value is nonzero, then the counter will decrement with one each clock cycle the core is waiting for bus ready to be asserted. If the counter reaches zero, the action taken depends on the state of Bus Error Enable (BEXCN) in MCFG1. If BEXCN is ‘1’, then an AMBA ERROR response will be generated and the counter will be reloaded. If BEXCN is ‘0’, then the bus ready enable for the accessed memory area will be disabled and the core will ignore bus ready for the accessed area.

Bus ready timeout functionality is disabled when the bus ready counter reload value is zero (MCFG7.BRDYCNTRLD = 0).

36.11 Access errors

An access error can be signalled by asserting the BEXCN signal for read and write accesses. For reads it is sampled together with the read data. For writes it is sampled on the last rising edge before chip select is de-asserted, which is controlled by means of waitstates or bus ready signalling. If the usage of BEXCN is enabled in memory configuration register 1, an error response will be generated on the internal AHB bus. BEXCN can be enabled or disabled through memory configuration register 1, and is active for all areas (PROM, IO and RAM). BEXCN is only sampled in the last access for 8- and 16-bit mode for RAM and PROM. That is, when four bytes are written for a word access to 8-bit wide memory BEXCN is only sampled in the last access with the same timing as a single access in 32-bit mode.

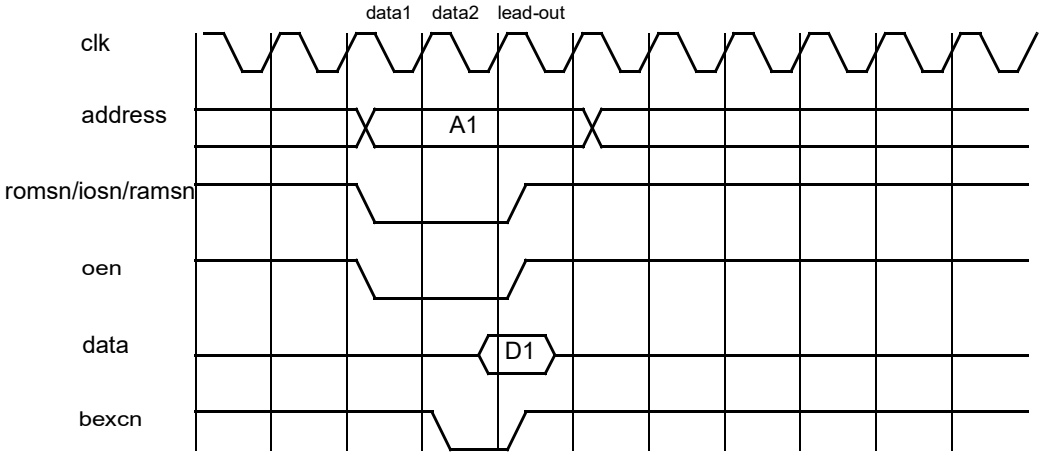


Figure 90. Read cycle with BEXCN.

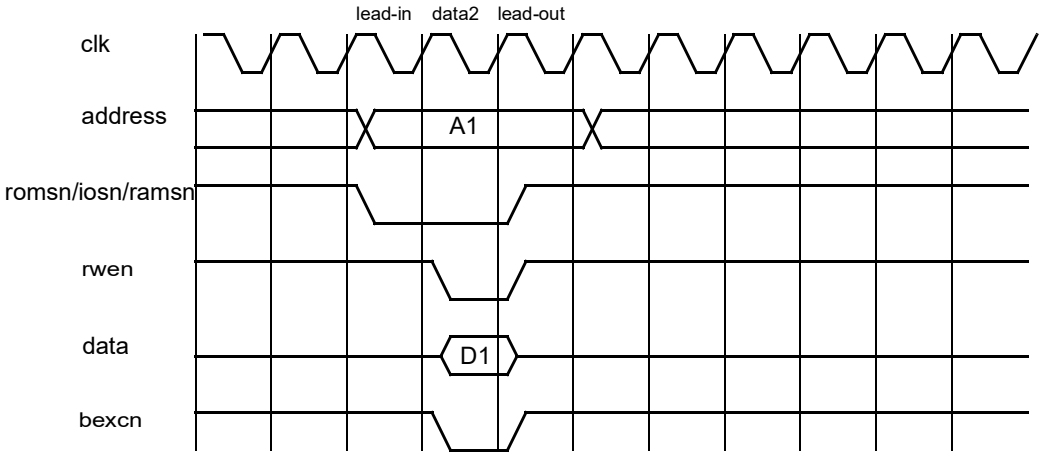


Figure 91. Write cycle with BEXCN. Chip-select (iosn) is not asserted in lead-in cycle for io-accesses.

36.12 Attaching an external DRAM controller

To attach an external DRAM controller, RAMSN[4] should be used since it allows the cycle time to vary through the use of BRDYN. In this way, delays can be inserted as required for opening of banks and refresh.

36.13 Output enable timing

A drive signal vector for the data I/O-pads is provided which has one drive signal for each data bit. It can be used if the synthesis tool does not generate separate registers automatically for the current technology. This can remove timing problems with output delay. An additional vector is used for the separate SDRAM bus.

36.14 Read strobe

The READ signal indicates the direction of the current PROM,SRAM,IO or SDRAM transfer, and it can be used to drive external bi-directional buffers on the data bus. It always is valid at least one cycle before and after the bus is driven, at other times it is held either constant high or low.

36.15 Known bugs and limitations

36.15.1 AHB slave interface non-compliance with AMBA 2.0 specification

The FTMCTRL AHB slave interface has a non-compliance with the AMBA 2.0 specification where the core will only assert HREADY when it is selected (via HSEL). The core functions correctly in GRLIB systems and there are no known compatibility issues caused by this behaviour. The HREADY behaviour has been, and will be, preserved to not compromise the extensive heritage of the memory controller.

36.15.2 Memory corruption from accesses to SDRAM and 8-bit SRAM with EDAC

When SDRAM and 8-bit SRAM with EDAC are simultaneously enabled, there is a condition where a sub-word (byte or half-word, 8- or 16-bit) write to SRAM that is in between two SDRAM accesses will be performed incorrectly. The logic that calculates the least significant address bits for the SRAM access uses the current state of the AHB bus instead of a previously latched state. This results in the read-modify-write sequence to SRAM starting from the wrong address, reading too few bytes, and then EDAC decoding results in an uncorrectable error with high probability (or silent memory corruption with a smaller probability).

This bug and the recommended workarounds are further documented in the technical erratum GRLIB-TN-0023.

36.16 Endianness

The core is designed for big-endian systems.

36.17 Registers

The core is programmed through registers mapped into APB address space.

*Table 418.*FTMCTRL memory controller registers

APB Address offset	Register
0x0	Memory configuration register 1 (MCFG1)
0x4	Memory configuration register 2 (MCFG2)
0x8	Memory configuration register 3 (MCFG3)
0xC	Memory configuration register 4 (MCFG4)
0x10	Memory configuration register 5 (MCFG5)
0x14	Memory configuration register 6 (MCFG6)
0x18	Memory configuration register 7 (MCFG7)

36.17.1 Memory configuration register 1 (MCFG1)

Memory configuration register 1 is used to program the timing of rom and IO accesses.

*Table 419.*0x00 - MCFG1 - Memory configuration register 1

31	30	29	28	27	26	25	24	23 20	19	18	17
	PBRDY	ABRDY	IOBUSW		IBRDY	BEXCN		IO WAITSTATES	IOEN	R	ROMBANKSZ
	0	0	NR		0	0	0	0x00	0	0	0x0
	rw	rw	rw		rw	rw	r	rw	rw	r	rw

14	13 12	11	10	9 8	7 4	3 0
ROMANKS7	RESERVED	PWEN	RES	PROM WIDTH	PROM WRITE WS	PROM READ WS
	0	0	0	*	0xE	0xE
rw	r	rw	r	rw	rw	rw

31 RESERVED

30 PROM area bus ready enable (PBRDY) - Enables bus ready (BRDYN) signalling for the PROM area. Reset to ‘0’.

29 Asynchronous bus ready (ABRDY) - Enables asynchronous bus ready.

28 : 27 I/O bus width (IOBUSW) - Sets the data width of the I/O area (“00”=8, “01”=16, “10” =32).

26 I/O bus ready enable (IBRDY) - Enables bus ready (BRDYN) signalling for the I/O area. Reset to ‘0’.

25 Bus error enable (BEXCN) - Enables bus error signalling for all areas. Reset to ‘0’.

24 RESERVED

23 : 20 I/O waitstates (IO WAITSTATES) - Sets the number of waitstates during I/O accesses (“0000”=0, “0001”=1, “0010”=2,..., “1111”=15).
The values above describe the default configuration The core can be configred at implementation to extend the number of waitstates. The number of wait states inserted will be (IO WAIT-STATES)*$2^{wsshift}$, where *wsshift* can be read from the first user-defined register in the core’s plug&play area (default is wsshift = 0).

19 I/O enable (IOEN) - Enables accesses to the memory bus I/O area.

18 RESERVED

17: 14 PROM bank size (ROMBANKSZ) - Returns current PROM bank size when read. “0000” is a special case and corresponds to a bank size of 256MiB. All other values give the bank size in binary steps: “0001”=16KiB, “0010”=32KiB, “0011”=64KiB,... , “1111”=256MiB (i.e. 8KiB * 2**ROM-BANKSZ). For value “0000” or “1111” only two chip selects are available. For other values, two chip select signals are available for fixed bank sizes. For other values, four chip select signals are available for programmable bank sizes.

Programmable bank sizes can be changed by writing to this register field. The written values correspond to the bank sizes and number of chip-selects as above. Reset to “0000” when programmable.

Programmable ROMBANKSZ is only available when romasel VHDL generic is 0. For other values this is a read-only register field containing the fixed bank size value.

13:12 RESERVED

11 PROM write enable (PWEN) - Enables write cycles to the PROM area.

10 RESERVED

9 : 8 PROM width (PROM WIDTH) - Sets the data width of the PROM area (“00”=8, “01”=16, “10”=32).

Table 419. 0x00 - MCFG1 - Memory configuration register 1

7 : 4 PROM write waitstates (PROM WRITE WS) - Sets the number of wait states for PROM write cycles (“0000”=0, “0001”=2, “0010”=4,..., “1111”=30).

The values above describe the default configuration The core can be configred at implementation to extend the number of waitstates. The number of wait states inserted will be (PROM WRITE WS)*2*$2^{wsshift}$, where *wsshift* can be read from the first user-defined register in the core’s plug&play area (default is wsshift = 0).

3 : 0 PROM read waitstates (PROM READ WS) - Sets the number of wait states for PROM read cycles (“0000”=0, “0001”=2, “0010”=4,...,”1111”=30). Reset to “1111”.

The values above describe the default configuration The core can be configred at implementation to extend the number of waitstates. The number of wait states inserted will be (PROM READ WS)*2*$2^{wsshift}$, where *wsshift* can be read from the first user-defined register in the core’s plug&play area (default is wsshift = 0).

During reset, the prom width (bits [9:8]) are set with value on BWIDTH inputs. The prom waitstates fields are set to 15 (maximum). External bus error and bus ready are disabled. All other fields are undefined.

36.17.2 Memory configuration register 2 (MCFG2)

Memory configuration register 2 is used to control the timing of the SRAM and SDRAM.

Table 420. 0x04 - MCFG2 - Memory configuration register 2

31	30	29 27	26	25 23	22 21	20 19	18	17	16
SDRF	TRP	SDRAM TRFC	TCAS	SDRAM BANKSZ	SDRAM COLSZ	SDRAM CMD	D64	SDPB	R
0	1	0x3	1	0	0x2	0	*	0	0
rw	rw	rw	rw	rw	rw	rw	r	rw	r

15	14	13	12 9	8	7	6	5 4	3 2	1 0
R	SE	SI	RAM BANK SIZE		RBRDY	RMW	RAM WIDTH	RAM WRITE WS	RAM READ WS
0	0	0	NR		NR	NR	NR	0	0
r	rw	rw	rw		rw*	rw	rw	rw	rw

31 SDRAM refresh (SDRF) - Enables SDRAM refresh.

30 SDRAM TRP parameter (TRP) - t_{RP} will be equal to 2 or 3 system clocks (0/1).

29 : 27 SDRAM TRFC parameter (SDRAM TRFC) - t_{RFC} will be equal to 3+field-value system clocks.

26 SDRAM TCAS parameter (TCAS) - Selects 2 or 3 cycle CAS delay (0/1). When changed, a LOAD-COMMAND-REGISTER command must be issued at the same time. Also sets RAS/CAS delay (t_{RCD}).

25 : 23 SDRAM bank size (SDRAM BANKSZ) - Sets the bank size for SDRAM chip selects (“000”=4MiB, “001”=8MiB, “010”=16MiB,...,. “111”=512MiB).

When configured for 64-bit wide SDRAM data bus (sdbits=64), the meaning of this field doubles so that “000”=8 MiB, .., “111”=1024 MiB

22 : 21 SDRAM column size (SDRAM COLSZ) - “00”=256, “01”=512, “10”=1024, “11”=2048 except when bit[25:23]=˜111˜ then ˜11˜=4096

20 : 19 SDRAM command (SDRAM CMD) - Writing a non-zero value will generate a SDRAM command. “01”=PRECHARGE, “10”=AUTO-REFRESH, “11”=LOAD-COMMAND-REGISTER. The field is reset after the command has been executed.

18 64-bit SDRAM data bus (D64) - Reads ‘1’ if the memory controller is configured for 64-bit SDRAM data bus width, ‘0’ otherwise. Read-only.

17 SDRAM Page Burst (SDPB) - SDRAM programmed for page bursts on read when set, else programmed for line burst lengths of 8 on read. Programmable when pageburst VHDL generic is 2, else read-only.

16 : 15 RESERVED

14 SDRAM enable (SE) - Enables the SDRAM controller and disables fifth SRAM bank (RAMSN[4]).

13 SRAM disable (SI) - Disables accesses to SRAM bank if bit 14 (SE) is set to ‘1’.

*Table 420.*0x04 - MCFG2 - Memory configuration register 2

12 : 9	RAM bank size (RAM BANK SIZE) - Sets the size of each RAM bank (“0000”=8KiB, “0001”=16KiB, “0010”=32KiB, “0011”= 64KiB,.., “1111”=256MiB)(i.e. 8KiB * 2**RAM BANK SIZE).
8	RESERVED
7	RAM bus ready enable (RBRDY) - Enables bus ready signaling for the RAM area. Bus read signaling for the RAM area is only available for the fifth chip-select and this field is only available if the memory controller has been implemented with the VHDL generic srbanks set to 5.
6	Read-modify-write enable (RMW) - Enables read-modify-write cycles for sub-word writes to 16- bit 32-bit areas with common write strobe (no byte write strobe).
5 : 4	RAM width (RAM WIDTH) - Sets the data width of the RAM area (“00”=8, “01”=16, “1X”=32). Note: there is an erratum associated with sub-word writes to EDAC protected 8-bit RAM areas when SDRAM is also enabled. Refer to section 36.15.2 for details.
3 : 2	RAM write waitstates (RAM WRITE WS) - Sets the number of wait states for RAM write cycles (“00”=0, “01”=1, “10”=2, “11”=3). The values above describe the default configuration The core can be configred at implementation to extend the number of waitstates. The number of wait states inserted will be (RAM WRITE WS)*$2^{ws\text{-}shift}$, where *wsshift* can be read from the first user-defined register in the core’s plug&play area (default is wsshift = 0).
1 : 0	RAM read waitstates (RAM READ WS) - Sets the number of wait states for RAM read cycles (“00”=0, “01”=1, “10”=2, “11”=3). The values above describe the default configuration The core can be configred at implementation to extend the number of waitstates. The number of wait states inserted will be (RAM READ WS)*$2^{ws\text{-}shift}$, where *wsshift* can be read from the first user-defined register in the core’s plug&play area (default is wsshift = 0).

36.17.3 Memory configuration register 3 (MCFG3)

MCFG3 contains the reload value for the SDRAM refresh counter and to control and monitor the memory EDAC.

*Table 421.*0x08 - MCFG3 - Memory configuration register 3

31 29	28	27	26
RESERVED	RSE	ME	SDRAM REFRESH COUNTER
0	0	1	NR
r	rw	r	rw

12	11	10	9	8	7 0
	WB	RB	RE	PE	TCB
	0	0	NR	0	NR
	rw	rw	rw	rw	rw

31 : 29	RESERVED
28	Reed-Solomon EDAC enable (RSE) - if set, will enable Reed-Solomon protection of SDRAM area when implemented
27	Memory EDAC (ME) - Indicates if memory EDAC is present. (read-only)
26 : 12	SDRAM refresh counter reload value (SDRAM REFRESH COUNTER)
11	EDAC diagnostic write bypass (WB) - Enables EDAC write bypass.
10	EDAC diagnostic read bypass (RB) - Enables EDAC read bypass.
9	RAM EDAC enable (RE) - Enable EDAC checking of the RAM area (including SDRAM). Note: there is an erratum associated with sub-word writes to EDAC protected 8-bit RAM areas when SDRAM is also enabled. Refer to section 36.15.2 for details.
8	PROM EDAC enable (PE) - Enable EDAC checking of the PROM area. Ar reset, this bit is initialized with the value of MEMI.EDAC.
7 : 0	Test checkbits (TCB) - This field replaces the normal checkbits during write cycles when WB is set. It is also loaded with the memory checkbits during read cycles when RB is set.

The period between each AUTO-REFRESH command is calculated as follows:

$t_{REFRESH}$ = ((reload value) + 1) / SYSCLK

36.17.4 Memory configuration register 4 (MCFG4)

MCFG4 is only present if the Reed-Solomon EDAC has been enabled with the *edac* VHDL generic. MCFG4 provides means to insert Reed-Solomon EDAC errors into memory for diagnostic purposes.

Table 422. 0x0C - MCFG4 - Memory configuration register 4

31	16
RESERVED	WB

15	0
TCB[15:0]	

31 : 17	RESERVED
16	EDAC diagnostic write bypass (WB) - Enables EDAC write bypass. Identical to WB in MCFG3.
15 : 0	Test checkbits (TCB) - This field replaces the normal checkbits during write cycles when WB is set. It is also loaded with the memory checkbits during read cycles when RB is set. Note that TCB[7:0] are identical to TCB[7:0] in MCFG3

36.17.5 Memory configuration register 5 (MCFG5)

MCFG5 contains fields to control lead out cycles for the ROM and IO areas.

Table 423. 0x10 - MCFG5 - Memory configuration register 5

31 30	29 23	22 16
RESERVED	IOHWS	RESERVED
	0x00	
	rw	

15 14	13 7	6 0
RESERVED	ROMHWS	RESERVED
	0x00	
	rw	

31 : 30	RESERVED
29:23	IO lead out (IOHWS) - Lead out cycles added to IO accesses are IOHWS(3:0)*$2^{IOHWS(6:4)}$
22 : 14	RESERVED
13:7	ROM lead out (ROMHWS) - Lead out cycles added to ROM accesses are ROMHWS(3:0)*$2^{ROMHWS(6:4)}$
6 : 0	RESERVED

36.17.6 Memory configuration register 6 (MCFG6)

MCFG6 contains fields to control lead out cycles for the (S)RAM area.

Table 424. 0x14 - MCFG6 - Memory configuration register 6

31	16
RESERVED	
0	
r	

15 14	13 7	6 0

*Table 424.*0x14 - MCFG6 - Memory configuration register 6

RESERVED	RAMHWS	RESERVED
r	0x00	r
0	rw	0

31 : 14 RESERVED

13:7 RAM lead out (RAMHWS) - Lead out cycles added to RAM accesses are RAMHWS(3:0)*$2^{RAMHWS(6:4)}$

6 : 0 RESERVED

36.17.7 Memory configuration register 7 (MCFG7)

MCFG7 contains fields to control bus ready timeout.

*Table 425.*0x18 - MCFG7 - Memory configuration register 7

31	16
BRDYNCNT	
0	
rw	

15	0
BRDYNRLD	
0	
rw	

31 : 16 Bus ready count (BRDYNCOUNT) - Counter value. If this register is written then the counter shall be written with the same value as BRDYNRLD.

15: 0 Bus ready reload value (BRDYNRLD) - Reload value for BRDYNCNT

36.18 Vendor and device identifiers

The core has vendor identifier 0x01 (GAISLER) and device identifier 0x054. For description of vendor and device identifiers, see GRLIB IP Library User's Manual.

36.19 Implementation

36.19.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers. See the documentation for the *syncrst* VHDL generic for information on asynchronous reset affecting external signals.

36.19.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

36.20 Configuration options

Table 426 shows the configuration options of the core (VHDL generics).

*Table 426.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0x1FFFFFFF. Also see documentation of romasel VHDL generic below.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.. Also see documentation of romasel VHDL generic below.	0 - 16#FFF#	16#E00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space. Default I/O area is 0x20000000 - 0x2FFFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR1 defining I/O address space.	0 - 16#FFF#	16#E00#
ramaddr	ADDR field of the AHB BAR2 defining RAM address space. Default RAM area is 0x40000000-0x7FFFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR2 defining RAM address space.	0 -16#FFF#	16#C00#
paddr	ADDR field of the APB BAR configuration registers address space.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR configuration registers address space.	0 - 16#FFF#	16#FFF#
wprot	RAM write protection.	0 - 1	0
invclk	unused	N/A	0
fast	Enable fast SDRAM address decoding.	0 - 1	0

*Table 426.*Configuration options

Generic	Function	Allowed range	Default
romasel	Sets the PROM bank size. *romasel 0*: selects a programmable mode where the ROMBANKSZ field in the MCFG1 register sets the bank size. When romasel is 0 and the bank size is configured (MCFG1 register, ROMBANKSZ field, via the core's register interface) to 0b000 or 0b1111 then address bit 28 is used to decode the banks. This means that the core must be mapped at a 512 MiB address boundary (0x0, 0x20000000, 0x40000000, .. see romaddr and rommask VHDL generics) for address decoding to work correctly. *romasel 1 - 14:* Values 1 - 14 sets the size in binary steps (1 = 16KiB, 2 = 32KiB, 3=64KiB, ..., 14=128MiB). Four chip-selects are available for these values. 15 sets the bank size to 256MiB with two chip-selects. *romasel 16 - 28:* Values 16 - 28 sets the bank size in binary steps (16 = 64 KiB, 17 = 128KiB, ... 28 = 256MiB). Two chip-selects are available for this range. The selected bank size is readable from the rombanksz field in the MCFG1 register for the non-programmable modes. The PROM area will wrap back to the first bank after the end of the last decoded bank. As an example, if romasel is set to 14 the following banks will be decoded: bank 0: 0x00000000 - 0x07FFFFFF bank 1: 0x08000000 - 0x0FFFFFFF bank 2: 0x10000000 - 0x17FFFFFF bank 3: 0x18000000 - 0x1FFFFFFF ...bank 0 starting again at 0x20000000 (the same pattern applies for other values less than 14, addresses will wrap after the last decoded bank). If romasel is 15 then the address decoding will result in the following: bank 0: 0x00000000 - 0x0FFFFFFF bank 1: 0x10000000 - 0x1FFFFFFF .. bank 0 starting again at offset 0x20000000 When instantiating the core care must be taken to see how many chip-selects that will be used as a result of the setting of romasel. This affects the base address at which the core can be placed (setting of romaddr and rommask VHDL generics). As an example, placing the PROM area at a 256 MiB address boundary, like the base address 0x10000000 and using romasel = 0, 14, 15 or 28 will NOT result in ROM chip-select 0 getting asserted for an access to the PROM base address as the address decoding requires that the core has been placed on a 512 MiB address boundary.	0 - 28	28
sdrasel	log2(RAM address space size) - 1. E.g if size of the RAM address space is 0x40000000 sdrasel is log2(2^30)-1= 29.	14 - 31	29
srbanks	Number of SRAM banks.	0 - 5	4
ram8	Enable 8-bit PROM, SRAM and I/O access.	0 - 1	0
ram16	Enable 16-bit PROM, SRAM and I/O access.	0 - 1	0
sden	Enable SDRAM controller.	0 - 1	0
sepbus	SDRAM is located on separate bus.	0 - 1	1
sdbits	32 or 64 -bit SDRAM data bus.	32, 64	32
oepol	Select polarity of drive signals for data pads. 0 = active low, 1 = active high.	0 - 1	0

*Table 426.*Configuration options

Generic	Function	Allowed range	Default
edac	Enable EDAC. 0 = No EDAC; 1 = BCH EDAC; 2 = BCH EDAC with pipelining; 3 = BCH + RS EDAC	0 - 3	0
sdlsb	Select least significant bit of the address bus that is connected to SDRAM.	-	2
syncrst	Choose between synchronous and asynchronous reset for chip-select, oen and drive signals.	0 - 1	0
pageburst	Line burst read of length 8 when 0, page burst read when 1, programmable read burst type when 2.	0-2	0
scantest	Enable scan test support	0 - 1	0
netlist	Use technology specific netlist instead of RTL code	0 - 1	0
tech	Technology to use for netlists	0 - NTECH	0
rahold	Unused	0 - 16	0
wsshift	Wait state counter shift. This value defines the number of steps to shift the wait state counter. The number of waitstates that the core can generate is limited by $2^{wsshift}$. See the wait state fields in the core's APB register descriptions to see the effect of this generic. The value of this generic can be read out in the first user-defined register of the core's plug&play area. This means that if wsshift is non-zero then the AHB controller must have full plug&play decoding enabled.	-	0
brdynto	Bus ready timeout conunter enable. If this generic is non-zero then the core will be implemented with a bus ready timeout counter (see MCFG7).	0 - 1	0

36.21 Scan support

Scan support is enabled by setting the SCANTEST generic to 1. When enabled, the asynchronous reset of any flip-flop will be connected to AHBI.testrst during when AHBI.testen = '1'.

36.22 Signal descriptions

Table 427 shows the interface signals of the core (VHDL ports).

*Table 427.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
MEMI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Bus ready strobe	Low
	BEXCN	Input	Bus exception	Low
	CB[15:0]	Input	EDAC checkbits	High
	WRN[3:0]	Input	SRAM write enable feedback signal	Low
	BWIDTH[1:0]	Input	Sets the reset value of the PROM data bus width field in the MCFG1 register	High
	EDAC	Input	The reset value for the PROM EDAC enable bit	High
	SD[31:0]	Input	SDRAM separate data bus	High
	SCB[15:0]	Input	SDRAM separate checkbit bus	High

Table 427. Signal descriptions

Signal name	Field	Type	Function	Active
MEMO	ADDRESS[31:0]	Output	Memory address	High
	CB[15:0]	Output	EDAC Checkbit	
	DATA[31:0]	Output	Memory data	-
	SDDATA[63:0]	Output	Sdram memory data	-
	RAMSN[4:0]	Output	SRAM chip-select	Low
	RAMOEN[4:0]	Output	SRAM output enable	Low
	IOSN	Output	Local I/O select	Low
	ROMSN[3:0]	Output	PROM chip-select	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0]. Any WRN[ ] signal can be used for CB[ ].	Low
	MBEN[3:0]	Output	Read/write byte enable: MBEN[0] corresponds to DATA[31:24], MBEN[1] corresponds to DATA[23:16], MBEN[2] corresponds to DATA[15:8], MBEN[3] corresponds to DATA[7:0]. Any MBEN[ ] signal can be used for CB[ ].	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus. Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0]. Any BDRIVE[ ] signal can be used for CB[ ].	Low/High
	VBDRIVE[31:0]	Output	Vectored I/O-pad drive signals.	Low/High
	SVBDRIVE[63:0]	Output	Vectored I/O-pad drive signals for separate sdram bus.	Low/High
	READ	Output	Read strobe	High
	SA[14:0]	Output	SDRAM separate address bus	High
	CE	Output	Single error detected	High
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
WPROT	WPROTHIT	Input	Unused	-

Table 427. Signal descriptions

Signal name	Field	Type	Function	Active
SDO	SDCASN	Output	SDRAM column address strobe	Low
	SDCKE[1:0]	Output	SDRAM clock enable	High
	SDCSN[1:0]	Output	SDRAM chip select	Low
	SDDQM[7:0]	Output	SDRAM data mask: SDDQM[7] corresponds to SD[63:56], SDDQM[6] corresponds to SD[55:48], SDDQM[5] corresponds to SD[47:40], SDDQM[4] corresponds to SD[39:32], SDDQM[3] corresponds to SD[31:24], SDDQM[2] corresponds to SD[23:16], SDDQM[1] corresponds to SD[15:8], SDDQM[0] corresponds to SD[7:0]. Any SDDQM[ ] signal can be used for CB[ ].	Low
	SDRASN	Output	SDRAM row address strobe	Low
	SDWEN	Output	SDRAM write enable	Low

* see GRLIB IP Library User's Manual

36.23 Signal definitions and reset values

The signals and their reset values are described in table 428.

*Table 428.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
address[27:0]	Output	Memory address	High	Undefined
data[31:0]	Input/Output	Memory data	High	Tri-state
cb[15:0]	Input/Output	Check bits	High	Tri-state
ramsn[4:0]	Output	SRAM chip select	Low	Logical 1
ramoen[4:0]	Output	SRAM output enable	Low	Logical 1
rwen[3:0]	Output,	SRAM write byte enable: rwen[0] corresponds to data[31:24], rwen[1] corresponds to data[23:16], rwen[2] corresponds to data[15:8], rwen[3] corresponds to data[7:0]. Any rwen[ ] signal can be used for cb[ ].	Low	Logical 1
ramben[3:0]	Output	SRAM read/write byte enable: ramben[0] corresponds to data[31:24], ramben[1] corresponds to data[23:16], ramben[2] corresponds to data[15:8], ramben[3] corresponds to data[7:0]. Any ramben[ ] signal can be used for cb[ ].	Low	Logical 1
oen	Output	Output enable	Low	Logical 1
writen	Output	Write strobe	Low	Logical 1
read	Output	Read strobe	High	Logical 1
iosn	Output	IO area chip select	Low	Logical 1
romsn[3:0]	Output	PROM chip select	Low	Logical 1
brdyn	Input	Bus ready. Extends accesses to the IO area.	Low	-
bexcn	Input	Bus exception.	Low	-
sa[15:0]	Output	SDRAM address	High	Undefined
sd[31:0]	Input/Output	SDRAM data	High	Tri-state
scb[15:0]	Input/Output	SDRAM check bits	High	Tri-state
sdcsn[1:0]	Output	SDRAM chip select	Low	Logical 1
sdwen	Output	SDRAM write enable	Low	Logical 1
sdrasn	Output	SDRAM row address strobe	Low	Logical 1
sdcasn	Output	SDRAM column address strobe	Low	Logical 1
sddqm[3:0]	Output	SDRAM data mask: sddqm[5] corresponds to scb[15:8], sddqm[4] corresponds to scb[7:0], sddqm[3] corresponds to sd[31:24], sddqm[2] corresponds to sd[23:16], sddqm[1] corresponds to sd[15:8], sddqm[0] corresponds to sd[7:0]. Any sddqm[ ] signal can be used for scb[ ].	Low	Logical 1

36.24 Timing

The timing waveforms and timing parameters are shown in figure 92 and are defined in table 429.

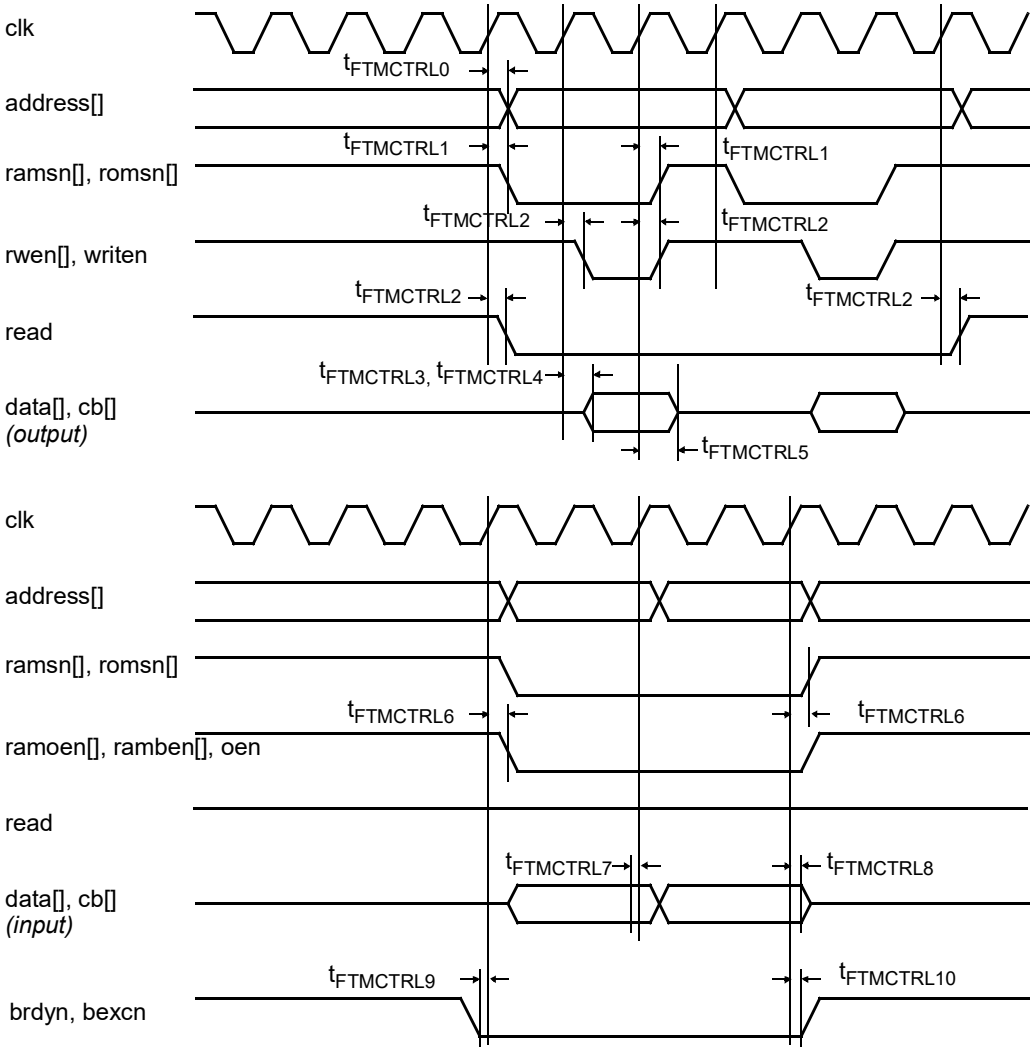


Figure 92. Timing waveforms - SRAM, PROM accesses

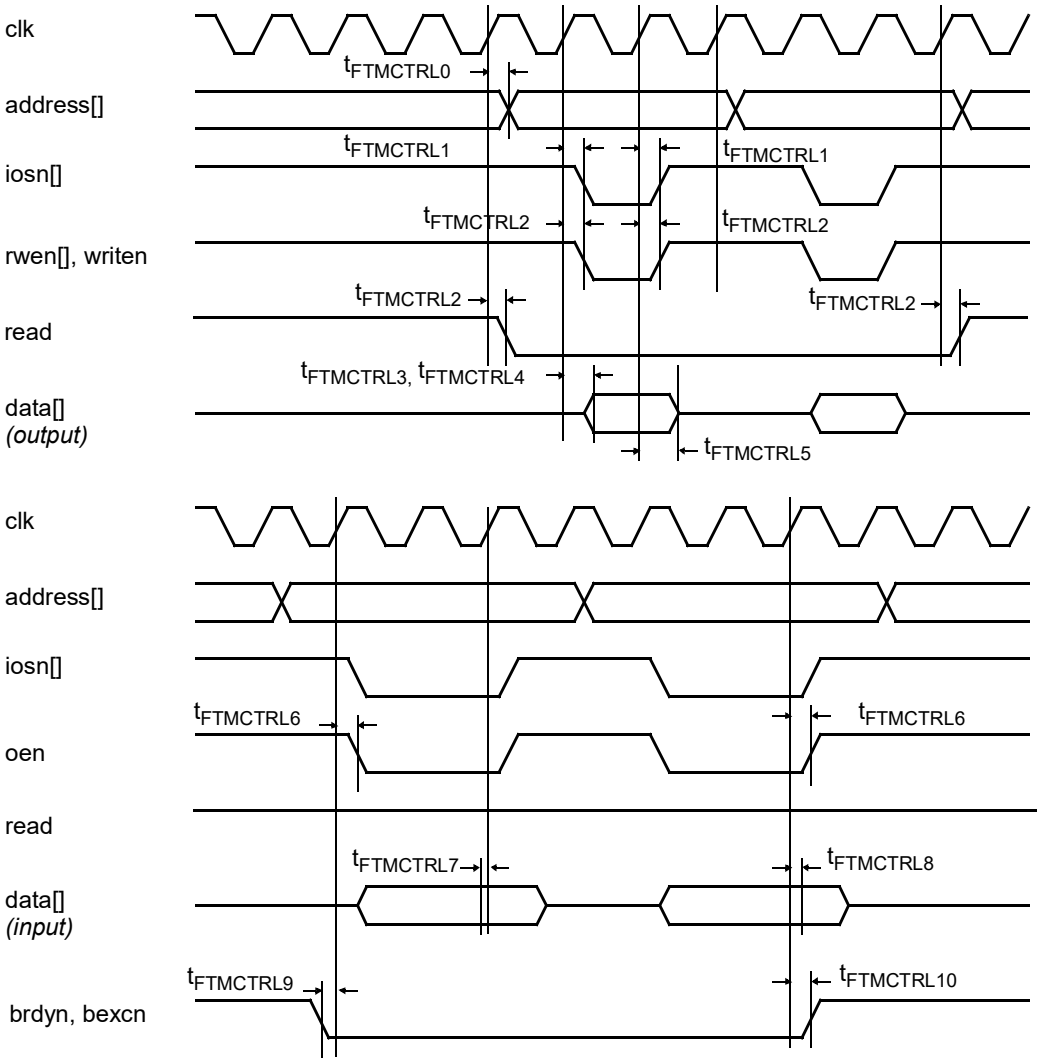


Figure 93. Timing waveforms - I/O accesses

*Table 429.*Timing parameters - SRAM, PROM and I/O accesses

Name	Parameter	Reference edge	Min	Max	Unit
$t_{FTMCTRL0}$	address clock to output delay	rising clk edge	TBD	TBD	ns
$t_{FTMCTRL1}$	clock to output delay	rising clk edge	TBD	TBD	ns
$t_{FTMCTRL2}$	clock to output delay	rising clk edge	TBD	TBD	ns
$t_{FTMCTRL3}$	clock to data output delay	rising clk edge	TBD	TBD	ns
$t_{FTMCTRL4}$	clock to data non-tri-state delay	rising clk edge	TBD	TBD	ns
$t_{FTMCTRL5}$	clock to data tri-state delay	rising clk edge	TBD	TBD	ns
$t_{FTMCTRL6}$	clock to output delay	rising clk edge	TBD	TBD	ns
$t_{FTMCTRL7}$	data input to clock setup	rising clk edge	TBD	-	ns
$t_{FTMCTRL8}$	data input from clock hold	rising clk edge	TBD	-	ns
$t_{FTMCTRL9}$	input to clock setup	rising clk edge	TBD	-	ns
$t_{FTMCTRL10}$	input from clock hold	rising clk edge	TBD	-	ns

The timing waveforms and timing parameters are shown in figure 92 and are defined in table 429.

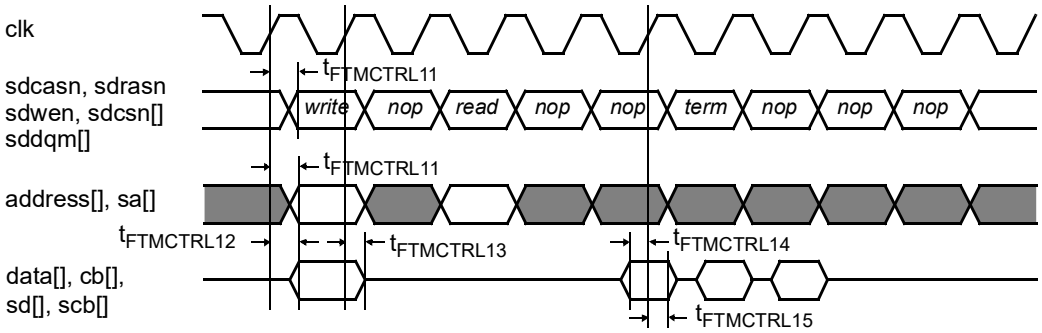


Figure 94. Timing waveforms - SDRAM accesses

*Table 430.*Timing parameters - SDRAM accesses

Name	Parameter	Reference edge	Min	Max	Unit
$t_{FTMCTRL11}$	clock to output delay	rising clk edge	TBD	TBD	ns
$t_{FTMCTRL12}$	clock to data output delay	rising clk edge	TBD	TBD	ns
$t_{FTMCTRL13}$	data clock to data tri-state delay	rising clk edge	TBD	TBD	ns
$t_{FTMCTRL14}$	data input to clock setup	rising clk edge	TBD	-	ns
$t_{FTMCTRL15}$	data input from clock hold	rising clk edge	TBD	-	ns

36.25 Library dependencies

Table 431 shows libraries used when instantiating the core (VHDL libraries).

*Table 431.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals	Memory bus signals definitions
		Components	FTMCTRL component

36.26 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined on the example designs port map and connected to the memory controller. System clock and reset are generated by GR Clock Generator and Reset Generator.

Memory controller decodes default memory areas: PROM area is 0x0 - 0x1FFFFFFF, I/O-area is 0x20000000-0x3FFFFFFF and RAM area is 0x40000000 - 0x7FFFFFFF. SDRAM controller is enabled. SDRAM clock is synchronized with system clock by clock generator.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads

entity mctrl_ex is
```

```
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;

    -- memory bus
    address  : out   std_logic_vector(27 downto 0); -- memory bus
    data     : inout std_logic_vector(31 downto 0);
    ramsn    : out   std_logic_vector(4 downto 0);
    ramoen   : out   std_logic_vector(4 downto 0);
    rwen     : inout std_logic_vector(3 downto 0);
    romsn    : out   std_logic_vector(3 downto 0);
    iosn     : out   std_logic;
    oen      : out   std_logic;
    read     : out   std_logic;
    writen   : inout std_logic;
    brdyn    : in    std_logic;
    bexcn    : in    std_logic;
-- sdram i/f
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                        -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0) -- optional sdram data
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo : sdram_out_type;

  signal wprot : wprot_out_type;  -- dummy signal, not used
  signal clkm, rstn : std_ulogic; -- system clock and reset

-- signals used by clock and reset generators
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;

  signal gnd : std_ulogic;

begin

  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                                tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  -- Memory controller
  ftmctrl0 : ftmctrl generic map (srbanks => 1, sden => 1, edac => 1)
    port map (rstn, clkm, memi, memo, ahbsi, ahbso(0), apbi, apbo(0), wprot, sdo);

  -- memory controller inputs not used in this configuration
  memi.brdyn <= '1'; memi.bexcn <= '1'; memi.wrn <= "1111";
```

```
  memi.sd <= sd;

  -- prom width at reset
  memi.bwidth <= "10";

  -- I/O pads driving data memory bus data signals
  datapads : for i in 0 to 3 generate
      data_pad : iopadv generic map (width => 8)
      port map (pad => memi.data(31-i*8 downto 24-i*8),
                o => memi.data(31-i*8 downto 24-i*8),
                en => memo.bdrive(i),
                i => memo.data(31-i*8 downto 24-i*8));
  end generate;

  -- connect memory controller outputs to entity output signals
  address <= memo.address; ramsn <= memo.ramsn; romsn <= memo.romsn;
  oen <= memo.oen; rwen <= memo.wrn; ramoen <= "1111" & memo.ramoen(0);
  sa <= memo.sa;
  writen <= memo.writen; read <= memo.read; iosn <= memo.iosn;
  sdcke <= sdo.sdcke; sdwen <= sdo.sdwen; sdcsn <= sdo.sdcsn;
  sdrasn <= sdo.rasn; sdcasn <= sdo.casn; sddqm <= sdo.dqm;
end;
```

37 FTSDCTRL - 32/64-bit PC133 SDRAM Controller with EDAC

37.1 Overview

The fault tolerant SDRAM memory interface handles PC133 SDRAM compatible memory devices attached to a 32- or 64-bit wide data bus. The interface acts as a slave on the AHB bus where it occupies configurable amount of address space for SDRAM access. An optional Error Detection And Correction Unit (EDAC) logic (only for the 32 - bit bus) corrects one bit error and detects two bit errors.

The SDRAM controller function is programmed by means of register(s) mapped into AHB I/O address space. Chip-select decoding is done for two SDRAM banks.

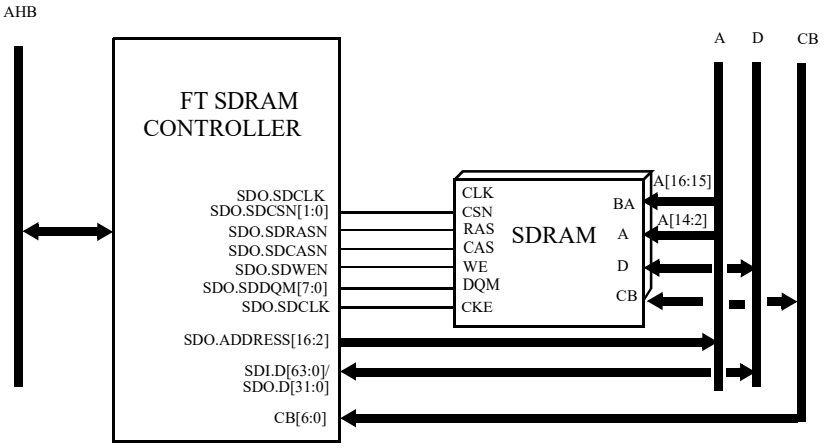


Figure 95. FT SDRAM memory controller connected to AMBA bus and SDRAM

37.2 Operation

37.2.1 General

Synchronous Dynamic RAM (SDRAM) access is supported to two banks of PC100/PC133 compatible devices. The controller supports 64, 256 and 512 Mbyte devices with 8 - 12 column-address bits, up to 13 row-address bits, and 4 banks. The size of each of the two banks can be programmed in binary steps between 4 Mbyte and 512 Mbyte. The operation of the SDRAM controller is controlled through the configuration register SDCFG. A second register, ECFG, is available for configuring the EDAC functions. SDRAM banks data bus width is configurable between 32 and 64 bits.

37.2.2 Initialization

When the SDRAM controller is enabled, it automatically performs the SDRAM initialization sequence of PRECHARGE, 8x AUTO-REFRESH and LOAD-MODE-REG on both banks simultaneously. When mobile SDRAM functionality is enabled, the initialization sequence is appended with a LOAD-EXTMODE-REG command. The controller programs the SDRAM to use page burst on read accesses and single location access on write accesses. If the *pwron* VHDL generic is 1, the initialization sequence is also sent automatically when reset is released. Note that some SDRAM devices require a stable clock of 100 us before any commands might be sent. When using on-chip PLL, this might not always be the case and the *pwron* VHDL generic should be set to 0 in such cases.

37.2.3 Configurable SDRAM timing parameters

To provide optimum access cycles for different SDRAM devices (and at different frequencies), three SDRAM parameters can be programmed through memory configuration register 2 (MCFG2): TCAS, TRP and TRFCD. The value of these fields affect the SDRAM timing as described in table 432.

*Table 432.*SDRAM programmable minimum timing parameters

SDRAM timing parameter	Minimum timing (clocks)
CAS latency, RAS/CAS delay (t_{CAS}, t_{RCD})	TCAS + 2
Precharge to activate (t_{RP})	TRP + 2
Auto-refresh command period (t_{RFC})	TRFC + 3
Activate to precharge (t_{RAS})	TRFC + 1
Activate to Activate (t_{RC})	TRP + TRFC + 4

If the TCAS, TRP and TRFC are programmed such that the PC100/133 specifications are fulfilled, the remaining SDRAM timing parameters will also be met. The table below shows typical settings for 100 and 133 MHz operation and the resulting SDRAM timing (in ns):

*Table 433.*SDRAM example programming

SDRAM settings	t_{CAS}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}
100 MHz, CL=2; TRP=0, TCAS=0, TRFC=4	20	80	20	70	50
100 MHz, CL=3; TRP=0, TCAS=1, TRFC=4	30	80	20	70	50
133 MHz, CL=2; TRP=1, TCAS=0, TRFC=6	15	82	22	67	52
133 MHz, CL=3; TRP=1, TCAS=1, TRFC=6	22	82	22	67	52

When mobile SDRAM support is enabled, one additional timing parameter (TXSR) can be programmed though the Power-Saving configuration register.

*Table 434.*Mobile SDRAM programmable minimum timing parameters

SDRAM timing parameter	Minimum timing (clocks)
Exit Self Refresh mode to first valid command (t_{XSR})	tXSR

37.2.4 Refresh

The SDRAM controller contains a refresh function that periodically issues an AUTO-REFRESH command to both SDRAM banks. The period between the commands (in clock periods) is programmed in the refresh counter reload field in the SDCFG register. Depending on SDRAM type, the required period is typically 7.8 or 15.6 µs (corresponding to 780 or 1560 clocks at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by setting bit 31 in SDCFG register.

37.2.5 Self Refresh

The self refresh mode can be used to retain data in the SDRAM even when the rest of the system is powered down. When in the self refresh mode, the SDRAM retains data without external clocking and refresh are handled internally. The memory array that is refreshed during the self refresh operation is defined in the extended mode register. These settings can be changed by setting the PASR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the PASR bits are changed. The supported “Partial Array Self Refresh” modes are: Full, Half, Quarter, Eighth, and Sixteenth array. “Partial Array Self Refresh” is only supported when mobile SDRAM functionality is enabled. To enable the self refresh mode, set the PMODE bits in the Power-Saving configuration register to “010” (Self Refresh). The controller will enter self refresh mode after

every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits are cleared. When exiting this mode the controller introduce a delay defined by tXSR in the Power-Saving configuration register and a AUTO REFRESH command before any other memory access is allowed. The minimum duration of this mode is defined by tRAS. This mode is only available when the VHDL generic *mobile* is >= 1.

37.2.6 Power-Down

When entering the power-down mode all input and output buffers, excluding SDCKE, are deactivated. All data in the SDRAM is retained during this operation. To enable the power-down mode, set the PMODE bits in the Power-Saving configuration register to “001” (Power-Down). The controller will enter power-down mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits is cleared. The REFRESH command will still be issued by the controller in this mode. When exiting this mode a delay of one clock cycles are added before issue any command to the memory. This mode is only available when the VHDL generic *mobile* is >= 1.

37.2.7 Deep Power-Down

The deep power-down operating mode is used to achieve maximum power reduction by eliminating the power of the memory array. Data will not be retained after the device enters deep power-down mode. To enable the deep power-down mode, set the PMODE bits in the Power-Saving configuration register to “101” (Deep Power-Down). To exit the deep power-down mode the PMODE bits in the Power-Saving configuration register must be cleared. The controller will respond with an AMBA ERROR response to an AMBA access, that will result in a memory access, during Deep Power-Down mode. This mode is only available when the VHDL generic *mobile* is >= 1 and mobile SDRAM functionality is enabled.

37.2.8 Temperature-Compensated Self Refresh

The settings for the temperature-compensation of the Self Refresh rate can be controlled by setting the TCSR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the TCSR bits are changed. Note that some vendors implements a Internal Temperature-Compensated Self Refresh feature, which makes the memory ignore the TCSR bits. This functionality is only available when the VHDL generic *mobile* is >= 1 and mobile SDRAM functionality is enabled.

37.2.9 Drive Strength

The drive strength of the output buffers can be controlled by setting the DS bits in the Power-Saving configuration register. The extended mode register is automatically updated when the DS bits are changed. The available options are: full, three-quarter, one-half, and one-quarter drive strengths. This functionality is only available when the VHDL generic *mobile* is >= 1 and mobile SDRAM functionality is enabled.

37.2.10 SDRAM commands

The controller can issue four SDRAM commands by writing to the SDRAM command field in the SDRAM Configuration register: PRE-CHARGE, AUTO-REFRESH, LOAD-MODE-REG (LMR) and LOAD-EXTMODE-REG (EMR). If the LMR command is issued, the CAS delay as programmed in SDCFG will be used. Line burst of length 8 will be set for read when pageburst VHDL generic is 0. Page burst will be set for read when pageburst VHDL generic is 1. Page burst or line burst of length 8, selectable via the SDCFG register will be set, when pageburst VHDL generic is 2. Remaining fields are fixed: page read burst, single location write, sequential burst. If the EMR command is issued, the DS, TCSR and PASR as programmed in Power-Saving configuration register will be used. The command field will be cleared after a command has been executed. Note that when changing the value of the CAS delay, a LOAD-MODE-REGISTER command should be generated at the same time.

37.2.11 Read cycles

A read cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a READ command after the programmed CAS delay. A read burst is performed if a burst access has been requested on the AHB bus. The read cycle is terminated with a PRE-CHARGE command, no banks are left open between two accesses. Note that only word bursts are supported by the SDRAM controller. The AHB bus supports bursts of different sizes such as bytes and halfwords but they cannot be used.

37.2.12 Write cycles

Write cycles are performed similarly to read cycles, with the difference that WRITE commands are issued after activation. A write burst on the AHB bus will generate a burst of write commands without idle cycles in-between. As in the read case, only word bursts are supported.

37.2.13 Address bus connection

The SDRAM address bus should be connected to SA[12:0], the bank address to SA[14:13], and the data bus to SD[31:0] or SD[63:0] if 64-bit data bus is used.

37.2.14 Data bus

Data bus width is configurable to 32 or 64 bits. 64-bit data bus allows the 64-bit SDRAM devices to be connected using the full data capacity of the devices. 64-bit SDRAM devices can be connected to 32-bit data bus if 64-bit data bus is not available but in this case only half the full data capacity will be used.

37.2.15 EDAC

The controller optionally contains Error Detection And Correction (EDAC) logic, using a BCH(32, 7) code. It is capable of correcting one bit error and detecting two bit errors. The EDAC logic does not add any additional waitstates during normal operation. Detected errors will cause additional waitstates for correction (single errors) or error reporting (multiple errors). Single errors are automatically corrected and generally not visible externally unless explicitly checked.

This checking is done by monitoring the ce signal and single error counter. This counter holds the number of detected single errors. The ce signal is asserted one clock cycle when a single error is detected and should be connected to the AHB status register. This module stores the AHB status of the instruction causing the single error and generates interrupts (see the AHB status register documentation for more information).

The EDAC functionality can be enabled/disabled during run-time from the ECFG register (and the logic can also be completely removed during synthesis with VHDL generics. The ECFG register also contains control bits and checkbit fields for diagnostic reads. These diagnostic functions are used for testing the EDAC functions on-chip and allows one to store arbitrary checkbits with each written word. Checkbits read from memory can also be controlled.

64-bit bus support is not provided when EDAC is enabled. Thus, the and edacen VHDL generic should never be set to one when the sdbits VHDL generic is set to 64.

The equations below show how the EDAC checkbits are generated:

```
CB0 = D0 ^ D4 ^ D6 ^ D7 ^ D8 ^ D9 ^ D11 ^ D14 ^ D17 ^ D18 ^ D19 ^ D21 ^ D26 ^ D28 ^ D29 ^ D31
CB1 = D0 ^ D1 ^ D2 ^ D4 ^ D6 ^ D8 ^ D10 ^ D12 ^ D16 ^ D17 ^ D18 ^ D20 ^ D22 ^ D24 ^ D26 ^ D28
CB2 = D0 ^ D3 ^ D4 ^ D7 ^ D9 ^ D10 ^ D13 ^ D15 ^ D16 ^ D19 ^ D20 ^ D23 ^ D25 ^ D26 ^ D29 ^ D31
CB3 = D0 ^ D1 ^ D5 ^ D6 ^ D7 ^ D11 ^ D12 ^ D13 ^ D16 ^ D17 ^ D21 ^ D22 ^ D23 ^ D27 ^ D28 ^ D29
CB4 = D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D14 ^ D15 ^ D18 ^ D19 ^ D20 ^ D21 ^ D22 ^ D23 ^ D30 ^ D31
CB5 = D8 ^ D9 ^ D10 ^ D11 ^ D12 ^ D13 ^ D14 ^ D15 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
CB6 = D0 ^ D1 ^ D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
```

37.2.16 Clocking

The SDRAM controller is designed for an external SDRAM clock that is in phase or slightly earlier than the internal AHB clock. This provides the maximum margin for setup and hold on the external signals, and allows highest possible frequency. For Xilinx and Altera devices, the GRLIB Clock Generator (CLKGEN) can be configured to produce a properly synchronized SDRAM clock. For other FPGA targets, the custom clock synchronization must be designed, or the inverted clock option can be used (see below). For ASIC targets, the SDRAM clock can be derived from the AHB clock with proper delay adjustments during place&route.

If the VHDL generic INVCLK is set, then all outputs from the SDRAM controller are delayed for 1/2 clock. This is done by clocking all output registers on the falling clock edge. This option can be used on FPGA targets where proper SDRAM clock synchronization cannot be achieved. The SDRAM clock can be the internal AHB clock without further phase adjustments. Since the SDRAM signals will only have 1/2 clock period to propagate, this option typically limits the maximum SDRAM frequency to 40 - 50 MHz.

37.2.17 Endianness

The core is designed for big-endian systems.

37.3 Registers

The memory controller is programmed through register(s) mapped into the AHB I/O space defined by the controllers AHB BAR1.

If EDAC is enabled through the use of the edacen VHDL generic, an EDAC configuration register will be available.

Only 32-bit single-accesses to the registers are supported.

*Table 435.*FT SDRAM controller registers

AHB address offset	Register
0x0	SDRAM Configuration register
0x4	EDAC Configuration register
0x8	SDRAM Power-Saving configuration register

37.3.1 SDRAM configuration register (SDCFG)

The SDRAM configuration register is used to control the timing of the SDRAM.

Table 436. 0x00 - SDCFG - SDRAM configuration register

31	30	29 27	26	25 23	22 21	20 18	17	16	15	14 0
Refresh	tRP	tRFC	tCD	SDRAM bank size	SDRAM col. size	SDRAM command	Page-Burst	MS	D64	SDRAM refresh load value
0	1	0b111	1	0	0b10	0	*	*	*	NR
rw	rw	rw	rw	rw	rw	rw	rw*	r	r	rw

31 SDRAM refresh. If set, the SDRAM refresh will be enabled.

30 SDRAM t_{RP} timing. t_{RP} will be equal to 2 or 3 system clocks (0/1).

29: 27 SDRAM t_{RFC} timing. t_{RFC} will be equal to 3 + field-value system clocks.

26 SDRAM CAS delay. Selects 2 or 3 cycle CAS delay (0/1). When changed, a LOAD-MODE-REGISTER command must be issued at the same time. Also sets RAS/CAS delay (tRCD).

Table 436. 0x00 - SDCFG - SDRAM configuration register

25: 23	SDRAM banks size. Defines the decoded memory size for each SDRAM chip select: “000”= 4 Mbyte, “001”= 8 Mbyte, “010”= 16 Mbyte .... “111”= 512 Mbyte. When configured for 64-bit wide SDRAM data bus (sdbits=64), the meaning of this field doubles so that “000”=8 Mbyte, .., “111”=1024 Mbyte
22: 21	SDRAM column size. “00”=256, “01”=512, “10”=1024, “11”=2048 except when bit[25:23]=˜111˜ then ˜11˜=4096
20: 18	SDRAM command. Writing a non-zero value will generate an SDRAM command: “010”=PRE-CHARGE, “100”=AUTO-REFRESH, “110”=LOAD-MODE-REGISTER, “111”=LOAD-EXT-MODE-REGISTER. The field is reset after command has been executed.
17	1 = pageburst is used for read operations, 0 = line burst of length 8 is used for read operations. (Only available when VHDL generic pageburst i set to 2)
16	Mobile SDR support enabled. '1' = Enabled, '0' = Disabled (read-only).
15	64-bit data bus (D64) - Reads ‘1’ if memory controller is configured for 64-bit data bus, otherwise ‘0’. Read-only.
14: 0	The period between each AUTO-REFRESH command - Calculated as follows: tREFRESH = ((reload value) + 1) / SYSCLK

37.3.2 EDAC configuration register (ECFG)

The EDAC configuration register controls the EDAC functions of the SDRAM controller during run time.

*Table 437.*0x04 - ECFG - EDAC configuration register

31	30 ... cntbits + 10	cnbits + 9 ... 10	9	8	7	6 ... 0
EAV	RESERVED	SEC	WB	RB	EN	TCB
*	0	NR	NR	NR	0	NR
r	r	wc	rw	rw	rw	rw

6: 0	TCB : Test checkbits. These bits are written as checkbits into memory during a write operation when the WB bit in the ECFG register is set. Checkbits read from memory during a read operation are written to this field when the RB bit is set.
7:	EN : EDAC enable. Run time enable/disable of the EDAC functions. If EDAC is disabled no error detection will be done during reads and subword writes. Checkbits will still be written to memory during write operations.
8:	RB : Read bypass. Store the checkbits read from memory during a read operation into the TCB field.
9:	WB : Write bypass. Write the TCB field as checkbits into memory for all write operations.
cntbits + 9: 10	SEC : Single error counter. This field is available when the errcnt VHDL generic is set to one during synthesis. It increments each time a single error is detected. It saturates when the maximum value is reached. The maximum value is the largest number representable in the number of bits used, which in turn is determined by the cntbits VHDL generic. Each bit in the counter can be reset by writing a one to it.
30:cntbits + 10	Reserved.
31:	EAV : EDAC available. This bit is always one if the SDRAM controller contains EDAC.

37.3.3 SDRAM Power-Saving configuration register (SDCFG2)

The SDRAM Power-Saving register is used to control the power settings of the SDRAM.

*Table 438.*0x08 - SDCFG2 - SDRAM Power-Saving configuration register

31	30	29 ... 24	23 ... 20	19	18 ... 16	15 ... 7	6	5	4	3	2 ... 0
ME	CE	RESERVED	tXSR	R	PMODE	RESERVED	DS		TCSR		PASR
*	*	0	*	0	0	0	0		0		0

*Table 438.*0x08 - SDCFG2 - SDRAM Power-Saving configuration register

rw*	rw*	r	rw*	r	rw	r	rw	rw	rw

31	Mobile SDRAM functionality enabled. '1' = Enabled (support for Mobile SDRAM), '0' = disabled (support for standard SDRAM)
30	Clock enable (CE). This value is driven on the CKE inputs of the SDRAM. Should be set to '1' for correct operation. This register bit is read only when Power-Saving mode is other then none.
29: 24	Reserved
23: 20	SDRAM tXSR timing. tXSR will be equal to field-value system clocks. (Read only when Mobile SDR support is disabled).
19	Reserved
18: 16	Power-Saving mode (Read only when Mobile SDR support is disabled). "000": none "001": Power-Down (PD) "010": Self-Refresh (SR) "101": Deep Power-Down (DPD)
15: 7	Reserved
6: 5	Selectable output drive strength (Read only when Mobile SDR support is disabled). "00": Full "01": One-half "10": One-quarter "11": Three-quarter
4: 3	Reserved for Temperature-Compensated Self Refresh (Read only when Mobile SDR support is disabled). "00": 70ªC "01": 45ªC "10": 15ªC "11": 85ªC
2: 0	Partial Array Self Refresh (Read only when Mobile SDR support is disabled). "000": Full array (Banks 0, 1, 2 and 3) "001": Half array (Banks 0 and 1) "010": Quarter array (Bank 0) "101": One-eighth array (Bank 0 with row MSB = 0) "110": One-sixteenth array (Bank 0 with row MSB = 00)

37.4 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x055. For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

37.5 Implementation

37.5.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). By default, the core makes use of synchronous reset and resets a subset of its internal registers.

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_async_reset_enable* is set.

The registers driving SDRAM chip select and output enables for the SDRAM data bus have asynchronous reset.

37.5.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

37.6 Configuration options

Table 439 shows the configuration options of the core (VHDL generics).

*Table 439.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR0 defining SDRAM area. Default is 0xF0000000 - 0xFFFFFFFF.	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0 defining SDRAM area.	0 - 16#FFF#	16#F00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space where SDCFG register is mapped.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR1 defining I/O address space.	0 - 16#FFF#	16#FFF#
wprot	Write protection.	0 - 1	0
invclk	Inverted clock is used for the SDRAM.	0 - 1	0
fast	Enable fast SDRAM address decoding.	0 - 1	0
pwron	Enable SDRAM at power-on.	0 - 1	0
sdbits	32 or 64 -bit data bus width.	32, 64	32
edacen	EDAC enable. If set to one, EDAC logic will be included in the synthesized design. An EDAC configuration register will also be available.	0 - 1	0
errcnt	Include an single error counter which is accessible from the EDAC configuration register.	0 - 1	0
cntbits	Number of bits used in the single error counter	1 - 8	1
pageburst	Enable SDRAM page burst operation. 0: Controller uses line burst of length 8 for read operations. 1: Controller uses pageburst for read operations. 2: Controller uses pageburst/line burst depending on PageBurst bit in SDRAM configuration register.	0 - 2	0
mobile	Enable Mobile SDRAM support 0: Mobile SDRAM support disabled 1: Mobile SDRAM support enabled but not default 2: Mobile SDRAM support enabled by default 3: Mobile SDRAM support only (no regular SDR support)	0 - 3	0

37.7 Signal descriptions

Table 440 shows the interface signals of the core (VHDL ports).

*Table 440.*Signals declarations

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
SDI	WPROT	Input	Not used	-
	DATA[63:0]	Input	Data	-
	CB[7:0]	Input	Checkbits	-
SDO	SDCKE[1:0]	Output	SDRAM clock enable	High
	SDCSN[1:0]	Output	SDRAM chip select	Low
	SDWEN	Output	SDRAM write enable	Low
	RASN	Output	SDRAM row address strobe	Low
	CASN	Output	SDRAM column address strobe	Low
	DQM[7:0]	Output	SDRAM data mask: DQM[7] corresponds to DATA[63:56], DQM[6] corresponds to DATA[55:48], DQM[5] corresponds to DATA[47:40], DQM[4] corresponds to DATA[39:32], DQM[3] corresponds to DATA[31:24], DQM[2] corresponds to DATA[23:16], DQM[1] corresponds to DATA[15:8], DQM[0] corresponds to DATA[7:0]. Any DQM[ ] signal can be used for CB[ ].	Low
	BDRIVE	Output	Drive SDRAM data bus	Low
	ADDRESS[16:2]	Output	SDRAM address	-
	DATA[31:0]	Output	SDRAM data	-
	CB[7:0]	Output	Checkbits	-
	CE	Output	Correctable Error	High

* see GRLIB IP Library User's Manual

37.8 Signal definitions and reset values

The signals and their reset values are described in table 441.

*Table 441.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
sa[14:0]	Output	SDRAM address	High	Undefined
sd[31:0]	Input/Output	SDRAM data	High	Tri-state
scb[15:0]	Input/Output	SDRAM check bits	High	Tri-state
sdcsn[1:0]	Output	SDRAM chip select	Low	Logical 1
sdwen	Output	SDRAM write enable	Low	Logical 1
sdrasn	Output	SDRAM row address strobe	Low	Logical 1
sdcasn	Output	SDRAM column address strobe	Low	Logical 1
sddqm[3:0]	Output	SDRAM data mask: sddqm[3] corresponds to sd[31:24], sddqm[2] corresponds to sd[23:16], sddqm[1] corresponds to sd[15:8], sddqm[0] corresponds to sd[7:0]. Any sddqm[] signal can be used for scb[ ].	Low	Logical 1

37.9 Timing

The timing waveforms and timing parameters are shown in figure 96 and are defined in table 442.

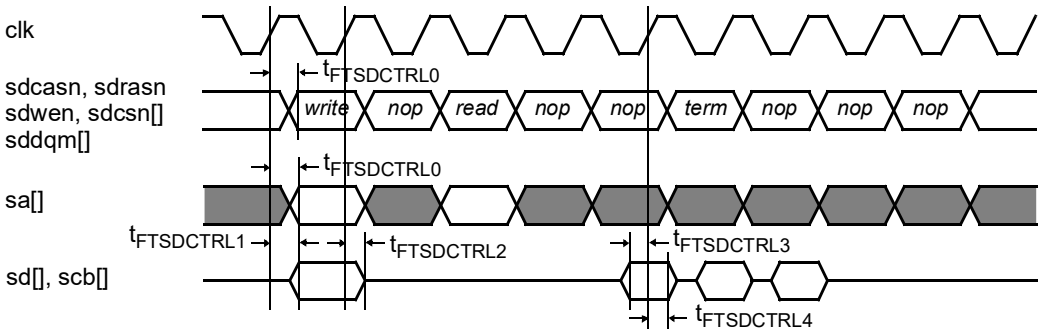


Figure 96. Timing waveforms

*Table 442.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{FTSDCTRL0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns
$t_{FTSDCTRL1}$	clock to data output delay	rising *clk* edge	TBD	TBD	ns
$t_{FTSDCTRL2}$	data clock to data tri-state delay	rising *clk* edge	TBD	TBD	ns
$t_{FTSDCTRL3}$	data input to clock setup	rising *clk* edge	TBD	-	ns
$t_{FTSDCTRL4}$	data input from clock hold	rising *clk* edge	TBD	-	ns

37.10 Library dependencies

Table 5 shows libraries used when instantiating the core (VHDL libraries).

*Table 443.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

37.11 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the FT SDRAM controller. The external SDRAM bus is defined in the example designs port map and connected to the SDRAM controller. System clock and reset are generated by GR Clock Generator and Reset Generator. It is also shown how the correctable error (CE) signal is connected to the ahb status register. It is not mandatory to connect this signal. In this example, 3 units can be connected to the status register.

The SDRAM controller decodes SDRAM area: 0x60000000 - 0x6FFFFFFF. SDRAM Configuration and EDAC configuration registers are mapped into AHB I/O space on address (AHB I/O base address + 0x100).

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;
    ... -- other signals

-- sdram memory bus
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0); -- optional sdram data
    cb       : inout std_logic_vector(7 downto 0) --EDAC checkbits
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
```

```
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect SDRAM controller and SDRAM memory bus
  signal sdi   : sdctrl_in_type;
  signal sdo   : sdctrl_out_type;

  signal clkm, rstn : std_ulogic; -- system clock and reset
signal ce : std_logic_vector(0 to 2); --correctable error signal vector

-- signals used by clock and reset generators
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;

  signal gnd : std_ulogic;

begin

  -- AMBA Components are defined here ...
  ...

  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                                tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  rst0 : rstgen
  port map (resetn, clkm, cgo.clklock, rstn);

  -- AHB Status Register
  astat0 : ahbstat generic map(pindex => 13, paddr => 13, pirq => 11,
    nftslv => 3)
    port map(rstn, clkm, ahbmi, ahbsi, ce, apbi, apbo(13));

  -- SDRAM controller
  sdc : ftsdctrl generic map (hindex => 3, haddr => 16#600#, hmask => 16#F00#,
    ioaddr => 1, fast => 0, pwron => 1, invclk => 0, edacen => 1, errcnt => 1,
    cntbits => 4)
    port map (rstn, clkm, ahbsi, ahbso(3), sdi, sdo, ce(0));

  -- input signals
  sdi.data(31 downto 0) <= sd(31 downto 0);

  -- connect SDRAM controller outputs to entity output signals
  sa <= sdo.address; sdcke <= sdo.sdcke; sdwen <= sdo.sdwen;
  sdcsn <= sdo.sdcsn; sdrasn <= sdo.rasn; sdcasn <= sdo.casn;
  sddqm <= sdo.dqm;

  -- I/O pads driving data bus signals
  sd_pad : iopadv generic map (width => 32)
      port map (sd(31 downto 0), sdo.data, sdo.bdrive, sdi.data(31 downto 0));

  -- I/O pads driving checkbit signals
  cb_pad : iopadv generic map (width => 8)
      port map (cb, sdo.cb, sdo.bdrive, sdi.cb);


end;
```

37.12 Constraints

This section contains example constraints for the SDRAM controller.

```
###### SDRAM interface
###

set sdram_freq 100.0
set sdram_clkper [ expr { 1000.0 / $sdram_freq } ]
```

```
create_clock -name "c_memclk" -period $sdram_clkper [get_ports "mem_clk"]

set sdram_cmd_ports [get_ports {mem_wen mem_rasn mem_cke mem_casn mem_ba mem_addr[12]
mem_addr[11] mem_addr[10] mem_addr[9] mem_addr[8] mem_addr[7] mem_addr[6] mem_addr[5]
mem_addr[4] mem_addr[3] mem_addr[2] mem_addr[1] mem_addr[0]}]
set sdram_cs_ports [get_ports {mem_sn*}]
set sdram_dq_ports [get_ports mem_dq]
set sdram_dqm_ports [get_ports mem_dqm]

# Use Micron datasheet values for SDRAM plus 1 ns margin for PCB propagation and other un-
modeled effects
set sdram_tAC 6.0
set sdram_tOHN 1.8
set sdram_ts_cmd 1.5
set sdram_th_cmd 0.8
set sdram_ts_cs 1.5
set sdram_th_cs 0.8
set sdram_ts_dq 1.5
set sdram_th_dq 0.8
set sdram_ts_dqm 1.5
set sdram_th_dqm 0.8

set_input_delay -clock "c_memclk" -min $sdram_tOHN $sdram_dq_ports
set_input_delay -clock "c_memclk" -max [expr { $sdram_tAC + 1.0 }] $sdram_dq_ports

set_output_delay -clock "c_memclk" -max [expr {$sdram_ts_cmd+1.0}] $sdram_cmd_ports
set_output_delay -clock "c_memclk" -min -$sdram_th_cmd $sdram_cmd_ports
set_output_delay -clock "c_memclk" -max [expr {$sdram_ts_cs+1.0}] $sdram_cs_ports
set_output_delay -clock "c_memclk" -min -$sdram_th_cs $sdram_cs_ports
set_output_delay -clock "c_memclk" -max [expr {$sdram_ts_dq+1.0}] $sdram_dq_ports
set_output_delay -clock "c_memclk" -min -$sdram_th_dq $sdram_dq_ports
set_output_delay -clock "c_memclk" -max [expr {$sdram_ts_dqm+1.0}] $sdram_dqm_ports
set_output_delay -clock "c_memclk" -min -$sdram_th_dqm $sdram_dqm_ports
```

38 FTSRCTRL - Fault Tolerant 32-bit PROM/SRAM/IO Controller

38.1 Overview

The fault tolerant 32-bit PROM/SRAM memory interface uses a common 32-bit memory bus to interface PROM, SRAM and I/O devices. Support for 8-bit PROM banks can also be separately enabled. In addition it also provides an Error Detection And Correction Unit (EDAC), correcting one and detecting two errors. Configuration of the memory controller functions is performed through the APB bus interface.

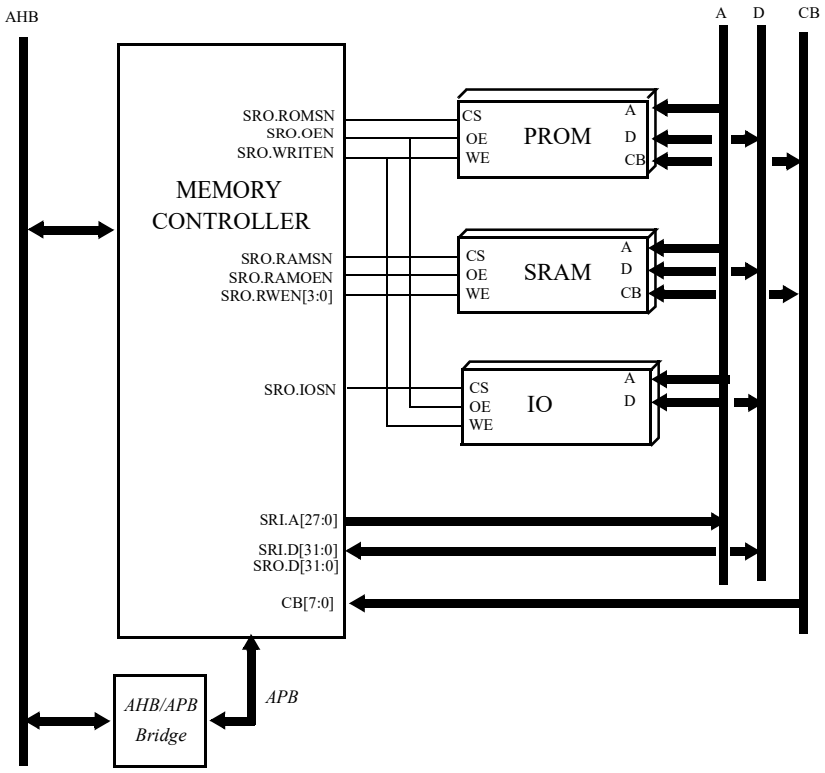


Figure 97. 32-bit FT PROM/SRAM/IO controller

38.2 Operation

The controller is configured through VHDL generics to decode three address ranges: PROM, SRAM and I/O area. By default the PROM area is mapped into address range 0x0 - 0x00FFFFFF, the SRAM area is mapped into address range 0x40000000 - 0x40FFFFFF, and the I/O area is mapped to 0x20000000 - 0x20FFFFFF.

One chip select is decoded for the I/O area, while SRAM and PROM can have up to 8 chip select signals. The controller generates both a common write-enable signal (WRITEN) as well as four byte-write enable signals (WREN). If the SRAM uses a common write enable signal the controller can be configured to perform read-modify-write cycles for byte and half-word write accesses. Number of waitstates is separately configurable for the three address ranges.

The EDAC function is optional, and can be enabled with the *edacen* VHDL generic. The configuration of the EDAC is done through a configuration register accessed from the APB bus. During nominal operation, the EDAC checksum is generated and checked automatically. Single errors are corrected without generating any indication of this condition in the bus response. If a multiple error is detected, a two cycle error response is given on the AHB bus.

Single errors can be monitored in two ways:

- by monitoring the CE signal which is asserted for one cycle each time a single error is detected.
- by checking the single error counter which is accessed from the MCFG3 configuration register.

The CE signal can be connected to the AHB status register which stores information of the AHB instruction causing the error and also generates interrupts. See the AHB status register documentation for more information. When EDAC is enabled, one extra latency cycle is generated during reads and subword writes.

The EDAC function can be enabled for SRAM and PROM area accesses, but not for I/O area accesses. For the SRAM area, the EDAC functionality is only supported for accessing 32-bit wide SRAM banks. For the PROM area, the EDAC functionality is supported for accessing 32-bit wide PROM banks, as well as for read accesses to 8-bit wide PROM banks.

The equations below show how the EDAC checkbits are generated:

```
CB0 = D0 ^ D4 ^ D6 ^ D7 ^ D8 ^ D9 ^ D11 ^ D14 ^ D17 ^ D18 ^ D19 ^ D21 ^ D26 ^ D28 ^ D29 ^ D31
CB1 = D0 ^ D1 ^ D2 ^ D4 ^ D6 ^ D8 ^ D10 ^ D12 ^ D16 ^ D17 ^ D18 ^ D20 ^ D22 ^ D24 ^ D26 ^ D28
CB2 = D0 ^ D3 ^ D4 ^ D7 ^ D9 ^ D10 ^ D13 ^ D15 ^ D16 ^ D19 ^ D20 ^ D23 ^ D25 ^ D26 ^ D29 ^ D31
CB3 = D0 ^ D1 ^ D5 ^ D6 ^ D7 ^ D11 ^ D12 ^ D13 ^ D16 ^ D17 ^ D21 ^ D22 ^ D23 ^ D27 ^ D28 ^ D29
CB4 = D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D14 ^ D15 ^ D18 ^ D19 ^ D20 ^ D21 ^ D22 ^ D23 ^ D30 ^ D31
CB5 = D8 ^ D9 ^ D10 ^ D11 ^ D12 ^ D13 ^ D14 ^ D15 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
CB6 = D0 ^ D1 ^ D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
```

38.2.1 8-bit PROM access

The FTSRCTRL controller can be configured to access an 8-bit wide PROM. The data bus of the external PROM should be connected to the upper byte of the 32-bit data bus, i.e. D[31:24]. The 8-bit mode is enabled with the prom8en VHDL generic. When enabled, read accesses to the PROM area will be done in four-byte bursts for all 32-, 16- and 8-bit AMBA AHB accesses. The whole 32-bit word is then output on the AHB data bus, allowing the master to chose the bytes needed (big-endian).

Writes should be done one byte at a time. For correct word aligned 32-bit word write accesses, the byte should always be driven on bits 31 to 24 on the AHB data bus. For non-aligned 32-bit word write accesses, the byte should be driven on the bits of the AHB data bus that correspond to the byte address (big-endian). For correct half-word aligned 16-bit half-word write accesses, the byte should always be driven on bits 31 to 24, or 15 to 8, on the AHB data bus. For non-aligned 16-bit half-word write accesses, the byte should be driven on the bits of the AHB data bus that correspond to the byte address (big-endian). For 8-bit word write accesses the byte should always be driven on the AHB data bus bits that corresponds to the byte address (big-endian). To summarize, all legal AMBA AHB write accesses are supported according to the AMBA standard, additional illegal accesses are supported as described above, and it is always the addressed byte that is output.

It is possible to dynamically switch between 8- and 32-bit PROM mode by writing to the RBW field of the MCFG1 register. The BWIDTH[1:0] input signal determines the reset value of this RBW register field. When RBW is “00” then 8-bit mode is selected. If RBW is “10” then 32-bit mode is selected. Other RBW values are reserved for future use. SRAM access is not affected by the 8-bit PROM mode.

It is also possible to use the EDAC in the 8-bit PROM mode, configured by the edacen VHDL generic, and enabled via the MCFG3 register. Read accesses to the 8-bit PROM area will be done in five-byte bursts for all 32-, 16- and 8-bit AMBA AHB accesses. After a potential correction, the whole 32-bit word is output on the AHB data bus, allowing the master to chose the bytes needed (big-endian). EDAC support is not provided for write accesses, they are instead performed in the same way as without the EDAC enabled. The checksum byte must be written by the user into the correct byte address location.

The fifth byte corresponds to the EDAC checksum and is located in the upper part of the effective memory area, as explained in detail in the definition of the MCFG1 memory configuration register. The EDAC checksums are located in the upper quarter of what is defined as available EDAC area by means of the EBSZ field and the ROMBSZ field or rombanksz VHDL generic. When set to 0, the size

of the available EDAC area is defined as the PROM bank size. When set to 1, as twice the PROM bank size. When set to 2, as four times the PROM bank size. And when set to 3, as eight times the PROM bank size. For any other value than 0, the use of multiple PROM banks is required.

Example, if ROMBSZ=10 and EBSZ=1, the EDAC area is 8KiB*2^ROMBSZ*2^EBSZ= 16MiB=0x01000000. The checksum byte for the first word located at address 0x00000000 to 0x00000003 is located at 0x00C00000. The checksum byte for the second word located at address 0x00000004 to 0x00000007 is located at 0x00C00001, and so on. Since EBSZ=1, two PROM banks are required for implementing the EDAC area, each bank with size 8MiB=0x00800000.

38.2.2 Access errors

The active low Bus Exception signal (BEXCN) can be used to signal access errors. It is enabled by setting the BEXCEN bit in MCFG1 and is active for all types of accesses to all areas (PROM, SRAM and I/O). The BEXCN signal is sampled on the same cycle as read data is sampled. For writes it is sampled on the last rising edge before writen/rwen is de-asserted (writen and rwen are clocked on the falling edge). When a bus exception is detected an error response will be generated for the access.

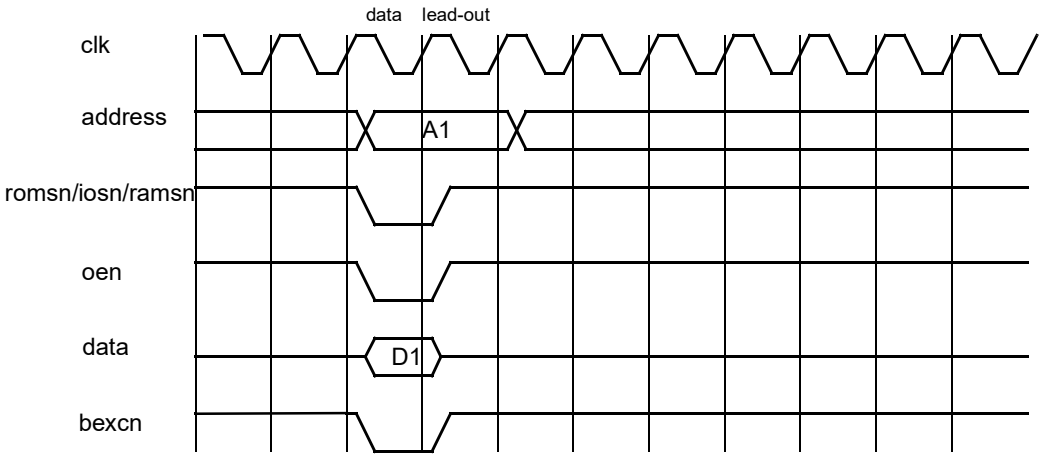


Figure 98. Read cycle with BEXCN.

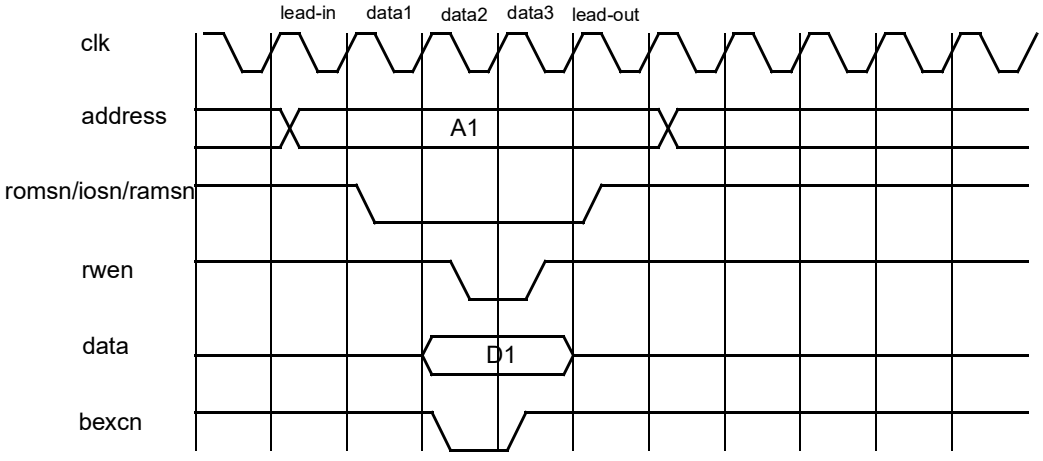


Figure 99. Write cycle with BEXCN.

38.2.3 Using bus ready signalling

The Bus Ready (BRDYN) signal can be used to add waitstates to I/O-area accesses, covering the complete memory area and both read and write accesses. It is enabled by setting the Bus Ready

Enable (BRDYEN) bit in the MCFG1 register. An access will have at least the amount of waitstates set with the VHDL generic or through the register, but will be further stretched until BRDYN is asserted. Additional waitstates can thus be inserted after the pre-set number of waitstates by de-asserting the BRDYN signal. BRDYN should be asserted in the cycle preceding the last one. It is recommended that BRDYN remains asserted until the IOSN signal is de-asserted, to ensure that the access has been properly completed and avoiding the system to stall. Read accesses will have the same timing as when EDAC is enabled while write accesses will have the timing as for single accesses even if bursts are performed.

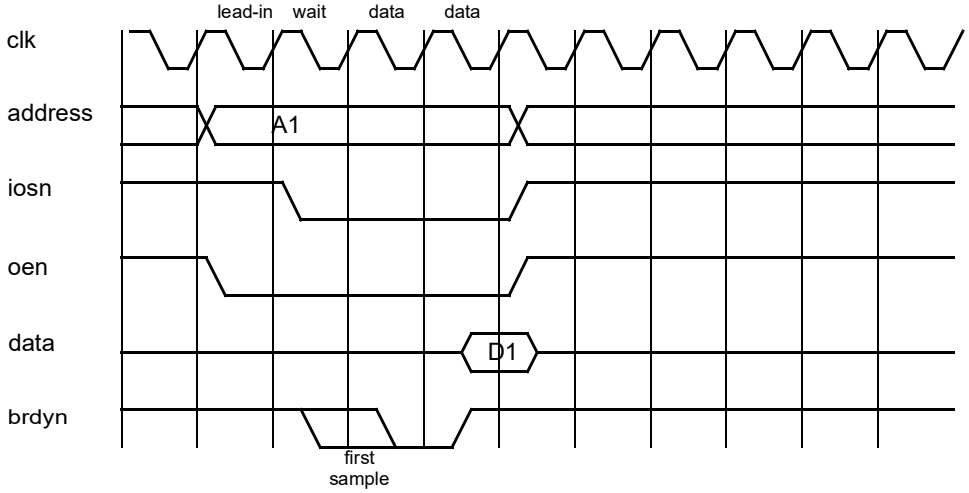


Figure 100. I/O READ cycle, programmed with 1 wait state, and with an extra data cycle added with BRDYN.

38.3 PROM/SRAM/IO waveforms

The internal and external waveforms of the interface are presented in the figures hereafter.

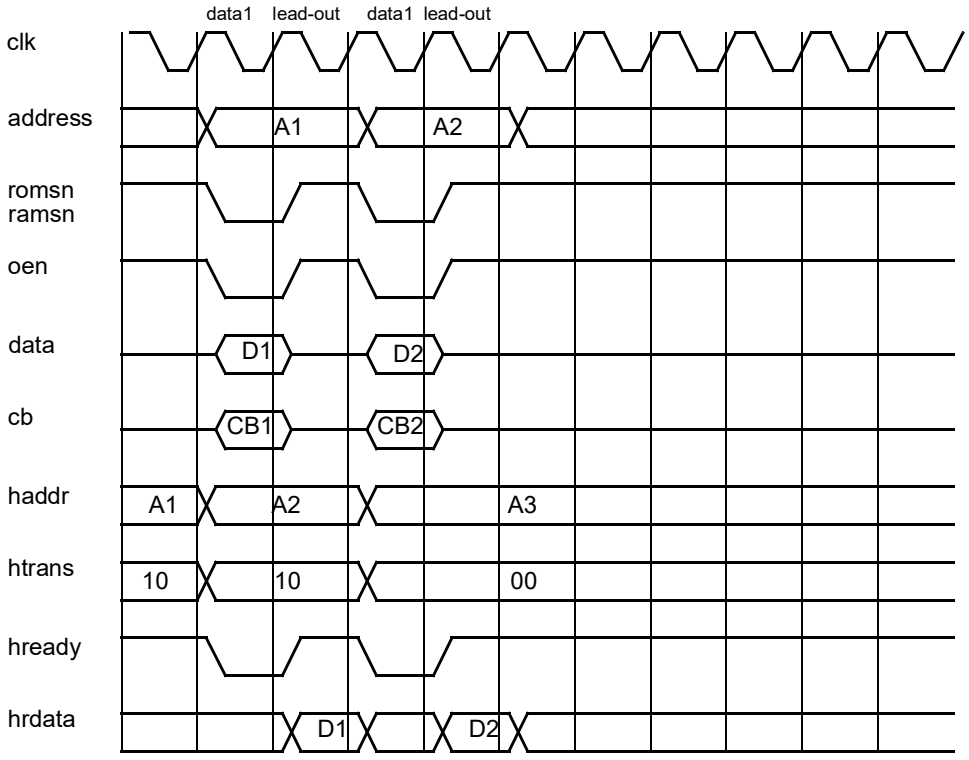


Figure 101. PROM/SRAM non-consecutive read cyclecs.

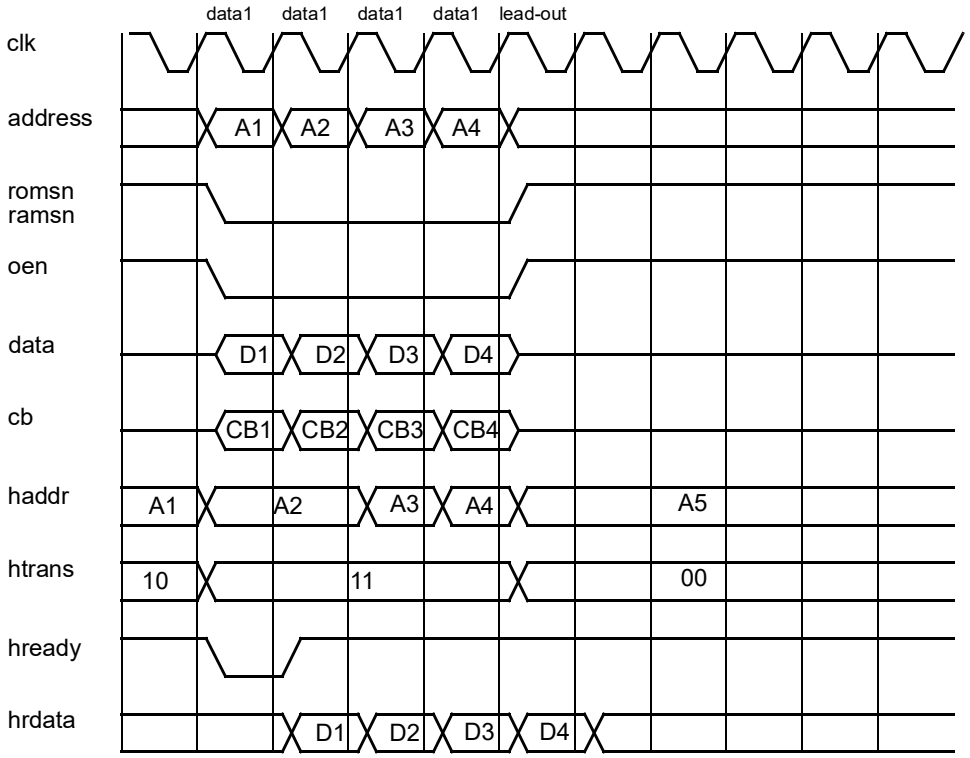


Figure 102. 32-bit PROM/SRAM sequential read access with 0 wait-states and EDAC disabled.

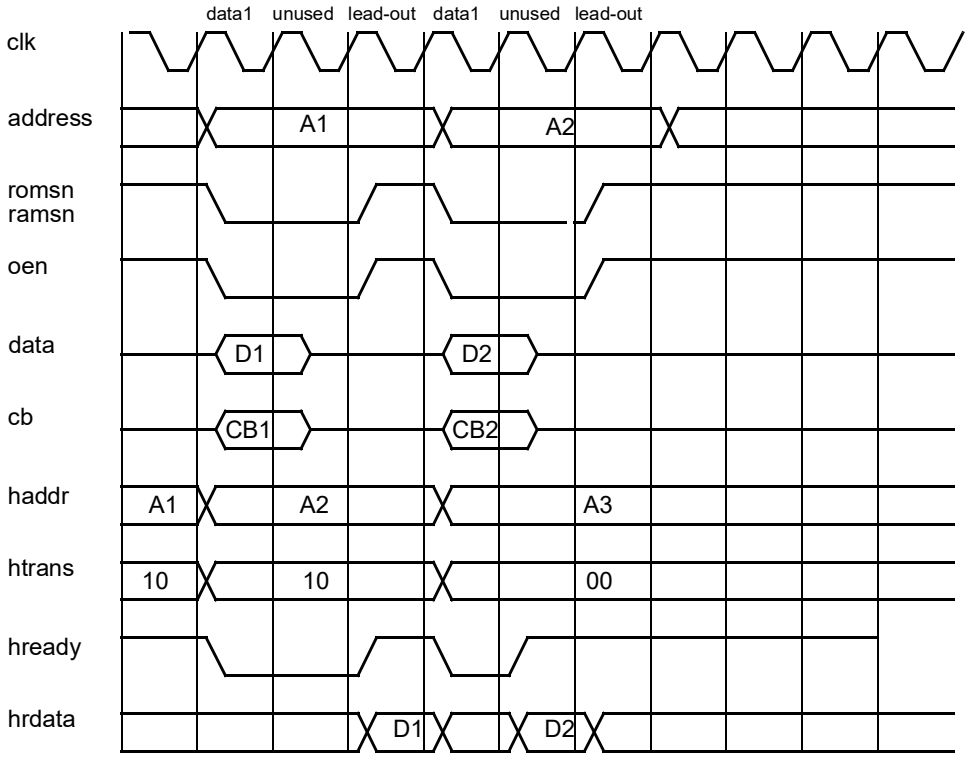


Figure 103. 32-bit PROM/SRAM non-sequential read access with 0 wait-states and EDAC enabled.

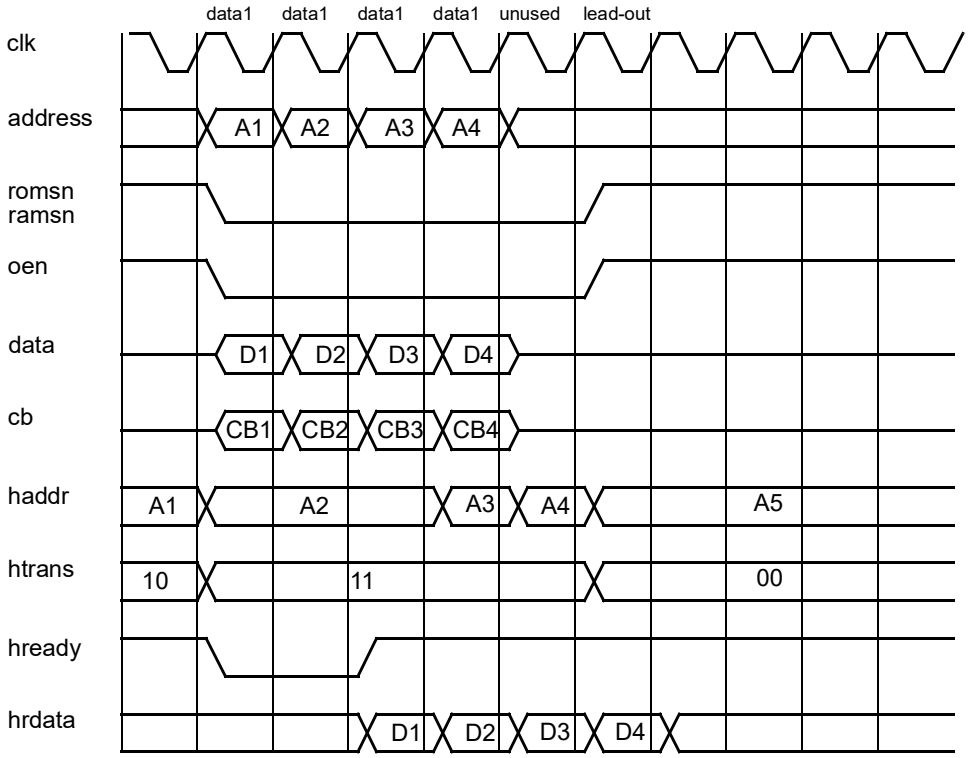


Figure 104. 32-bit PROM/SRAM sequential read access with 0 wait-states and EDAC enabled..

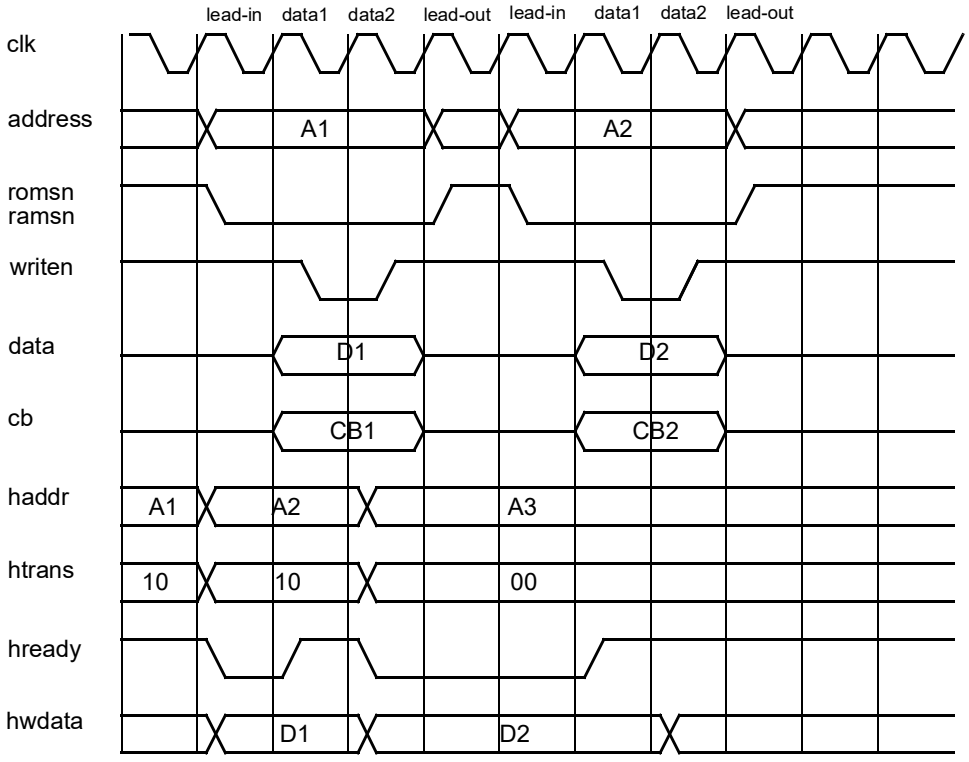


Figure 105. 32-bit PROM/SRAM non-sequential write access with 0 wait-states and EDAC disabled.

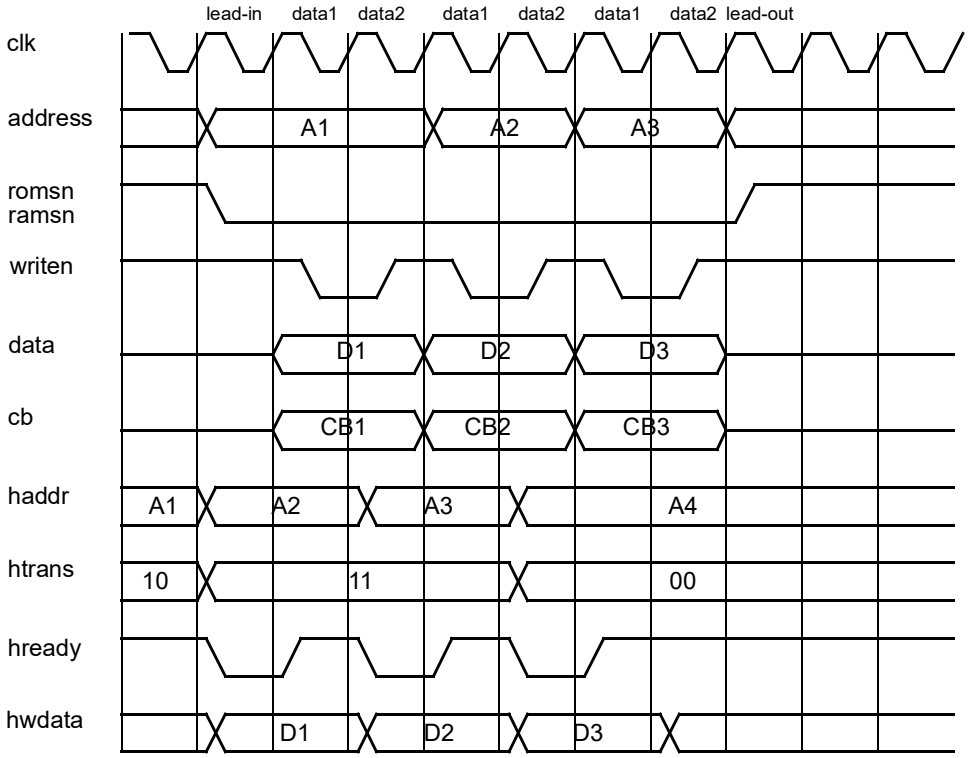


Figure 106. 32-bit PROM/SRAM sequential write access with 0 wait-states and EDAC disabled.

If waitstates are configured through the VHDL generics or registers, one extra data cycle will be inserted for each waitstate in both read and write cycles. The timing for write accesses is not affected when EDAC is enabled while one extra latency cycle is introduced for single access reads and at the beginning of read bursts.

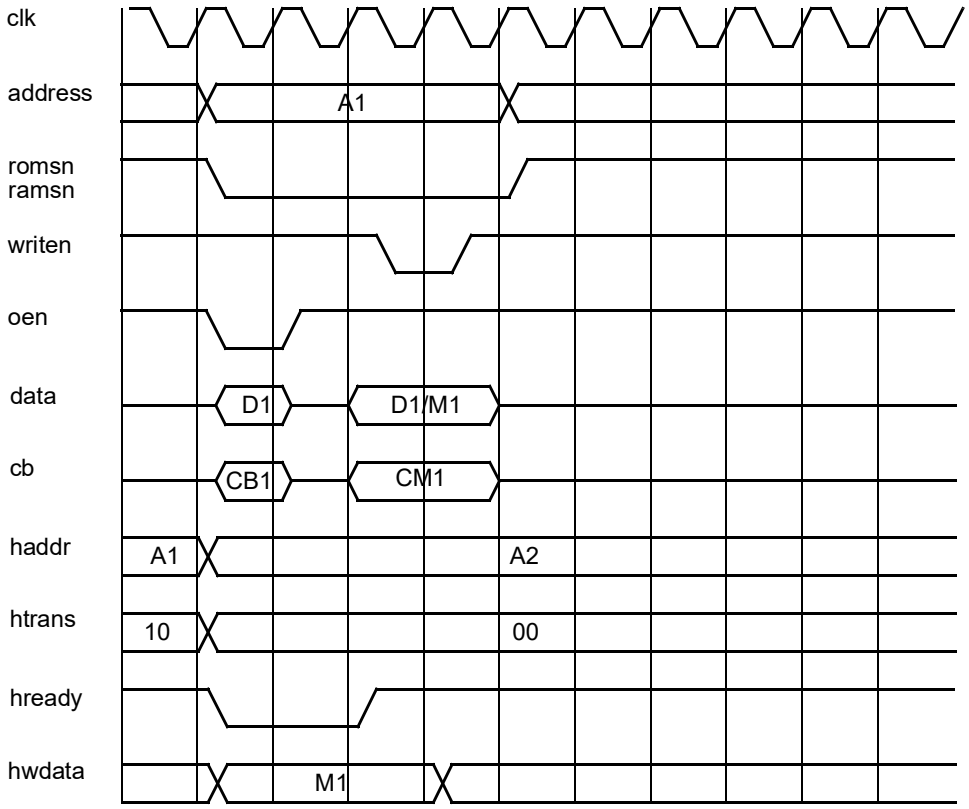


Figure 107. 32-bit PROM/SRAM rmw access with 0 wait-states and EDAC disabled.

Read-Modify-Write (RMW) accesses will have an additional waitstate inserted to accommodate decoding when EDAC is enabled.

I/O accesses are similar to PROM and SRAM accesses but a lead-in and lead-out cycle is always present.

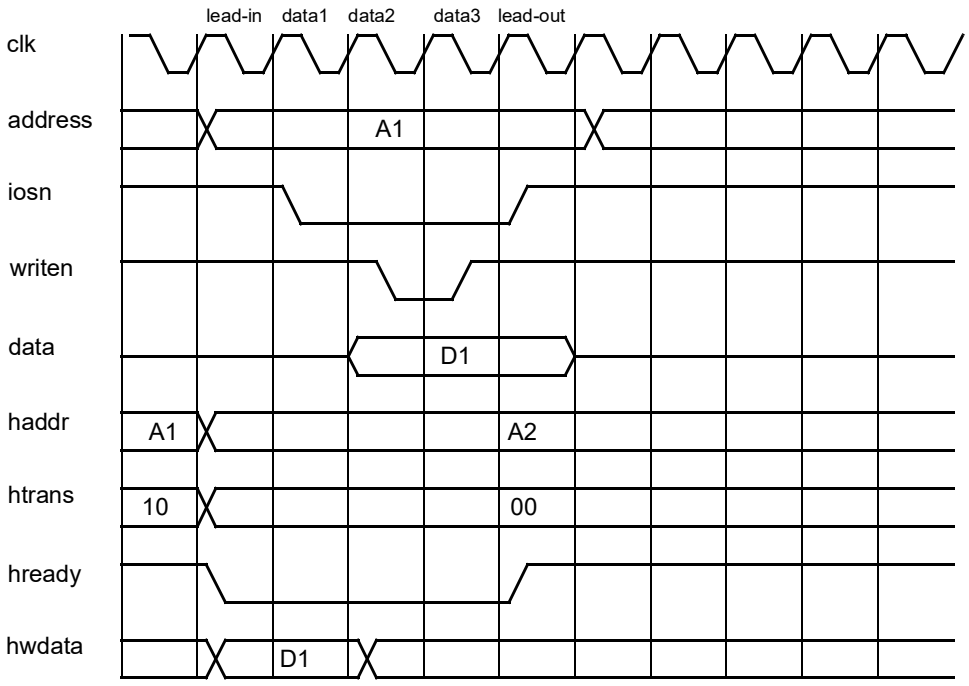


Figure 108. I/O write access with 0 wait-states.

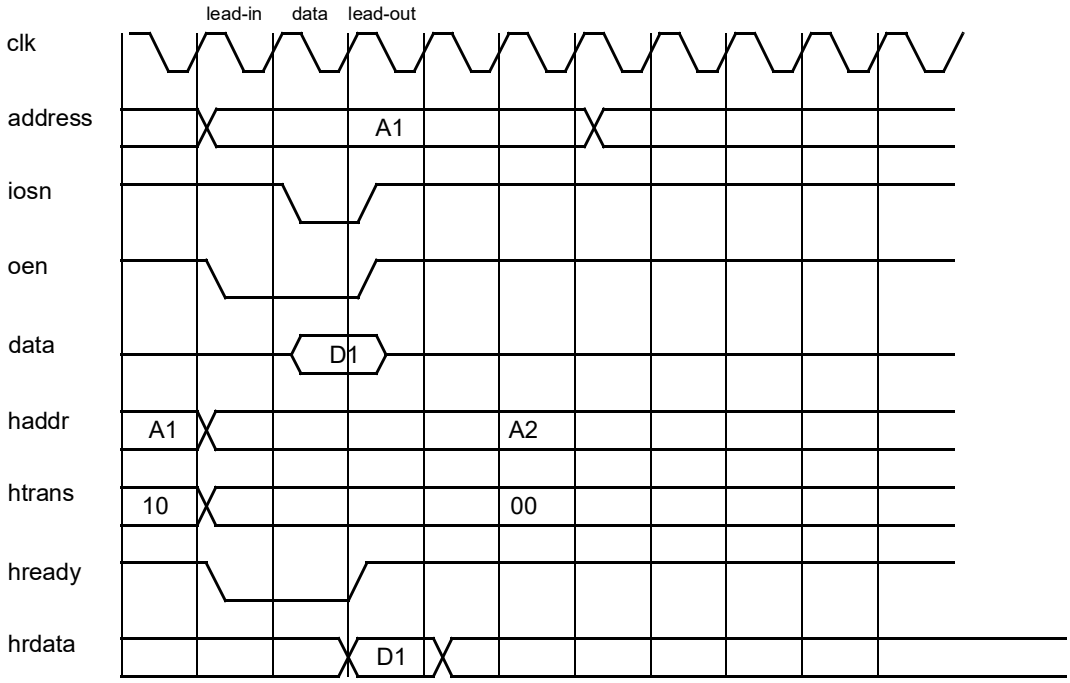


Figure 109. I/O read access with 0 wait-states

38.4 Endianness

The core is designed for big-endian systems.

38.5 Registers

The core is programmed through registers mapped into APB address space.

Table 444. FT PROM/SRAM/IO controller registers

APB Address offset	Register
0x0	Memory configuration register 1
0x4	Memory configuration register 2
0x8	Memory configuration register 3

38.5.1 Memory Configuration Register 1

*Table 445.*0x00 - MCFG1 - Memory configuration register 1.

31 27	26	25	24	23 20	19 18	17 14	13 12	11	10	9 8	7 4	3 0
RESERVED	BR	BE	R	IOWS	R	ROMBSZ	EBSZ	RW	R	RBW	RESERVED	ROMWS
0	0	0	0	0	0	*	*	0	0	*	0	0xF
r	rw	rw	r	rw	r	rw*	rw*	rw	r	rw	r	rw

31: 27 RESERVED

26 Bus ready enable (BR) - Enables the bus ready signal (BRDYN) for I/O-area.

25 Bus exception enable (BE) - Enables the bus exception signal (BEXCEN) for PROM, SRAM and I/O areas

24 RESERVED

23: 20 I/O wait states (IOWS) - Sets the number of waitstates for accesses to the I/O-area. Only available if the wsreg VHDL generic is set to one.

19: 18 RESERVED

17: 14 ROM bank size (ROMBSZ) - Sets the PROM bank size. Only available if the rombanksz VHDL generic is set to zero. Otherwise, the rombanksz VHDL generic sets the bank size and the value can be read from this field. 0 = 8KiB, 1 = 16KiB, 2 = 32KiB, 3 = 64KiB, ..., 15=256 MiB (i.e. 8 KiB * 2**ROMBSZ).

13: 12 EDAC bank size (EBSZ) - Sets the EDAC bank size for 8-bit PROM support. Only available if the rombanksz VHDL generic is zero, and edacen and prom8en VHDL generics are one. Otherwise, the value is fixed to 0. The resulting EDAC bank size is 2^EBSZ * 2^ROMBSZ * 8KiB. Note that only the three lower quarters of the bank can be used for user data. The EDAC checksums are placed in the upper quarter of the bank.

11 ROM write enable (RW) - Enables writes to the PROM memory area. When disabled, writes to the PROM area will generate an ERROR response on the AHB bus.

10 RESERVED

9: 8 ROM data bus width (RBW) - Sets the PROM data bus width. “00” = 8-bit, “10” = 32-bit, others reserved.

7: 4 RESERVED

3: 0 ROM waitstates (ROMWS) - Sets the number of waitstates for accesses to the PROM area. Reset to all-ones. Only available if the wsreg generic is set to one.

38.5.2 Memory Configuration Register 2

*Table 446.*0x04 - MCFG2 - Memory configuration register 2.

31 13	12 9	8 7	6	5 2	1 0
RESERVED	RAMBSZ	R	RW	RESERVED	RAMW
0	*	0	*	0	0
r	rw*	r	rw*	r	rw*

31: 13 RESERVED

12: 9 RAM bank size (RAMBSZ) - Sets the RAM bank size. Only available if the banksz VHDL generic is set to zero. Otherwise, the banksz VHDL generic sets the bank size and the value can be read from this field. 0 = 8KiB, 1 = 16KiB, 2 = 32KiB, 3 = 64KiB, ..., 15=256 MiB (i.e. 8 KiB * 2**RAMBSZ)

8: 7 RESERVED

6 Read-modify-write enable (RW) - Enables read-modify-write cycles for write accesses. Only available if the rmw VHDL generic is set to one.

5: 2 RESERVED

1: 0 RAM waitstates (RAMW) - Sets the number of waitstates for accesses to the RAM area. Only available if the wsreg VHDL generic is set to one.

38.5.3 Memory Configuration Register 3

*Table 447.*0x08 - MCFG3 - Memory configuration register 3.

31 20	19 12	11	10	9	8	7 0
RESERVED	SEC	WB	RB	SE	PE	TCB
0	0	0	0	0	0	NR
r	wc	rw	rw	rw	rw	rw*

31: 20	RESERVED
19: 12	Single error counter.(SEC) - This field increments each time a single error is detected until the maximum value that can be stored in the field is reached. Each bit can be reset by writing a one to it.
11	Write bypass (WB) - Enables EDAC write bypass. When enabled the TCB field will be used as checkbits in all write operations.
10	Read bypass (RB) - Enables EDAC read bypass. When enabled checkbits read from memory in all read operations will be stored in the TCB field.
9	SRAM EDAC enable (SE) - Enables EDAC for the SRAM area.
8	PROM EDAC enable (PE) - Enables EDAC for the PROM area. Reset value is taken from the input signal sri.edac.
7: 0	Test checkbits (TCB) - Used as checkbits in write operations when WB is activated and checkbits from read operations are stored here when RB is activated.

All the fields in MCFG3 register are available if the edacen VHDL generic is set to one except SEC field which also requires that the errcnt VHDL generic is set to one. The exact breakpoint between the SEC and RESERVED field depends on the cntbits generic. The breakpoint is 11+cntbits. The values shown in the table is for maximum cntbits value 8.

38.6 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x051. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

38.7 Implementation

38.7.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers. The registers driving external chip select, output enable and output enables for the data bus have asynchronous reset.

38.7.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

38.8 Configuration options

Table 444 shows the configuration options of the core (VHDL generics).

Table 448. Controller configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index.	1 - NAHBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0xFFFFFF.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.	0 - 16#FFF#	16#FF0#
ramaddr	ADDR field of the AHB BAR1 defining RAM address space. Default RAM area is 0x40000000-0x40FFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR1 defining RAM address space.	0 -16#FFF#	16#FF0#
ioaddr	ADDR field of the AHB BAR2 defining IO address space. Default RAM area is 0x20000000-0x20FFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR2 defining IO address space.	0 - 16#FFF#	16#FF0#
ramws	Number of waitstates during access to SRAM area.	0 - 15	0
romws	Number of waitstates during access to PROM area.	0 - 15	2
iows	Number of waitstates during access to IO area.	0 - 15	2
rmw	Enable read-modify-write cycles.	0 - 1	0
srbanks	Set the number of RAM banks.	1 - 8	1
banksz	Set the size of bank 1 - 4. 1 = 16KiB, 2 = 32KiB, 3 = 64KiB, ... , 15 = 256 MiB (i.e. 8 KiB * 2**banksz). If set to zero, the bank size is set with the rambsz field in the MCFG2 register.	0 - 15	15
rombanks	Sets the number of PROM banks available.	1 - 8	1
rombanksz	Sets the size of one PROM bank. 1 = 16KiB, 2 = 32KiB, 3 = 64KiB, ... , 15 = 256 MiB (i.e. 8 KiB * 2**rombanksz). If set to zero, the bank size is set with the rombsz field in the MCFG1 register.	0 - 15	15
rombankszdef	Sets the reset value of the rombsz register field in MCFG1 if available.	0 - 15	15
pindex	APB slave index.	1 - NAPBSLV-1	0
paddr	APB address.	1 - 16#FFF#	0
pmask	APB address mask.	1 - 16#FFF#	16#FFF#
edacen	EDAC enable. If set to one, EDAC logic is synthesized.	0 - 1	0
errcnt	If one, a single error counter is added.	0 - 1	0
cntbits	Number of bits in the single error counter.	1 - 8	1
wsreg	Enable programmable waitstate generation.	0 - 1	0
prom8en	Enable 8-bit PROM mode.	0 - 1	0
oepol	Select polarity of output enable signals. 0 = active low, 1 = active high.	0 - 1	0

38.9 Signal descriptions

Table 449 shows the interface signals of the core (VHDL ports).

*Table 449.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low

Table 449. Signal descriptions

Signal name	Field	Type	Function	Active
SRI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Bus ready strobe	Low
	BEXCN	Input	Bus exception	Low
	WRN[3:0]	Input	Not used	-
	BWIDTH[1:0]	Input	Sets the reset value of the PROM data bus width field in the MCFG1 register	-
	SD[31:0]	Input	Not used	-
	CB[7:0]	Input	Checkbits	-
	PROMDATA[31:0]	Input	Not used	-
	EDAC	Input	The reset value for the PROM EDAC enable bit	High

Table 449. Signal descriptions

Signal name	Field	Type	Function	Active
SRO	ADDRESS[31:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data	High
	RAMSN[7:0]	Output	SRAM chip-select	Low
	RAMOEN[7:0]	Output	SRAM output enable	Low
	IOSN	Output	IO area chip select	Low
	ROMSN[7:0]	Output	PROM chip-select	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0]. Any WRN[ ] signal can be used for CB[ ].	Low
	MBEN[3:0]	Output	Byte enable: MBEN[0] corresponds to DATA[31:24], MBEN[1] corresponds to DATA[23:16], MBEN[2] corresponds to DATA[15:8], MBEN[3] corresponds to DATA[7:0]. Any MBEN[ ] signal can be used for CB[ ].	
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus.Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0]. Any BDRIVE[ ] signal can be used for CB[ ].	Low
	READ	Output	Read strobe	High
	RAMN	Output	Common SRAM Chip Select. Always asserted when one of the 8 RAMSN signals is asserted.	Low
	ROMN	Output	Common PROM Chip Select. Always asserted when one of the 8 ROMSN signals is asserted.	Low
	SA[14:0]	Output	Not used	-
	CB[7:0]	Output	Checkbits	-
	PSEL	Output	Not used	-
	CE	Output	Single error detected.	High
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
SDO	SDCASN	Output	Not used. All signals are drive to inactive state.	Low

* see GRLIB IP Library User's Manual

38.10 Signal definitions and reset values

The signals and their reset values are described in table 450.

*Table 450.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
address[27:0]	Output	Memory address	High	Undefined
data[31:0]	Input/Output	Memory data	High	Tri-state
cb[7:0]	Input/Output	Check bits	High	Tri-state
ramsn[3:0]	Output	SRAM chip select	Low	Logical 1
ramoen[3:0]	Output	SRAM output enable	Low	Logical 1
rwen[3:0]	Output,	SRAM write byte enable: rwen[0] corresponds to data[31:24], rwen[1] corresponds to data[23:16], rwen[2] corresponds to data[15:8], rwen[3] corresponds to data[7:0]. Any rwen[ ] signal can be used for cb[ ].	Low	Logical 1
ramben[3:0]	Output	SRAM read/write byte enable: ramben[0] corresponds to data[31:24], ramben[1] corresponds to data[23:16], ramben[2] corresponds to data[15:8], ramben[3] corresponds to data[7:0]. Any ramben[ ] signal can be used for cb[ ].	Low	Logical 1
oen	Output	Output enable	Low	Logical 1
writen	Output	Write strobe	Low	Logical 1
read	Output	Read strobe	High	Logical 1
iosn	Output	IO area chip select	Low	Logical 1
romsn[1:0]	Output	PROM chip select	Low	Logical 1
brdyn	Input	Bus ready. Extends accesses to the IO area.	Low	-
bexcn	Input	Bus exception.	Low	-

38.11 Timing

The timing waveforms and timing parameters are shown in figure 110 and are defined in table 451.

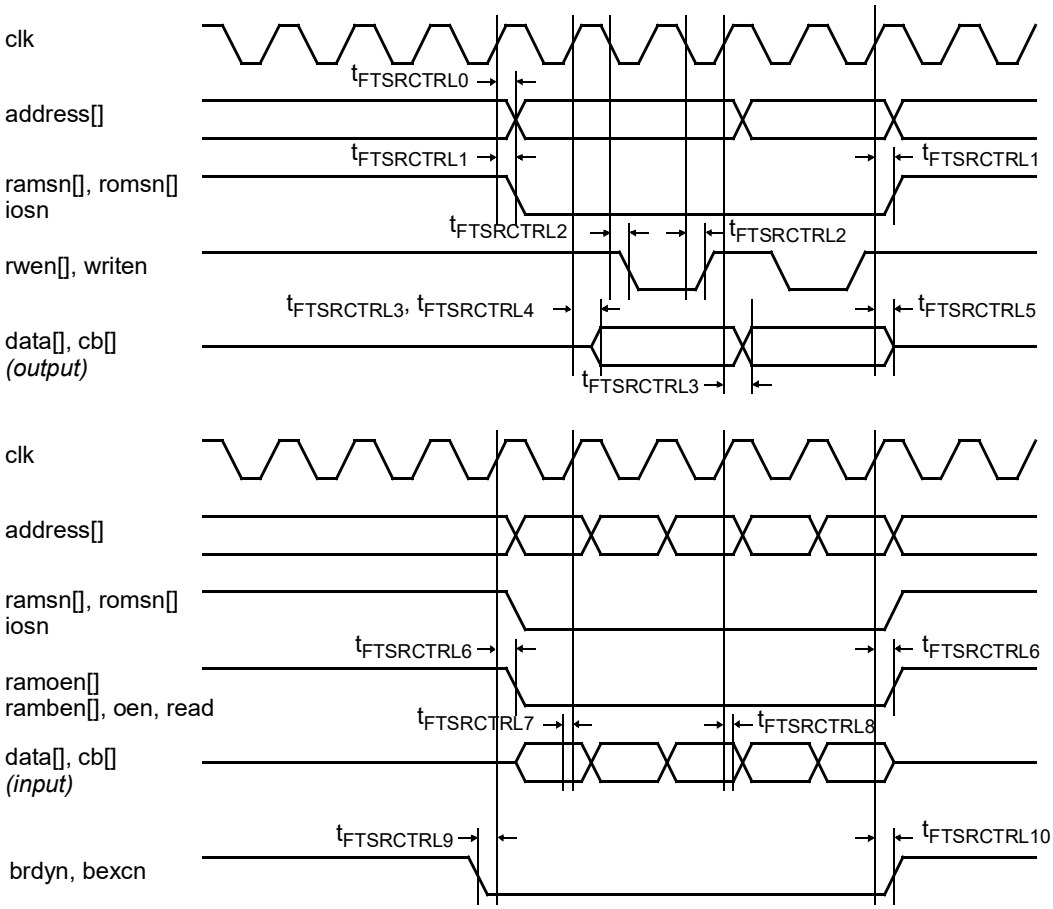


Figure 110. Timing waveforms

*Table 451.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{FTSRCTRL0}$	address clock to output delay	rising *clk* edge	TBD	TBD	ns
$t_{FTSRCTRL1}$	clock to output delay	rising *clk* edge	TBD	TBD	ns
$t_{FTSRCTRL2}$	clock to output delay	rising *clk* edge	TBD	TBD	ns
$t_{FTSRCTRL3}$	clock to data output delay	falling *clk* edge	TBD	TBD	ns
$t_{FTSRCTRL4}$	clock to data non-tri-state delay	rising *clk* edge	TBD	TBD	ns
$t_{FTSRCTRL5}$	clock to data tri-state delay	rising *clk* edge	TBD	TBD	ns
$t_{FTSRCTRL6}$	clock to output delay	rising *clk* edge	TBD	TBD	ns
$t_{FTSRCTRL7}$	data input to clock setup	rising *clk* edge	TBD	-	ns
$t_{FTSRCTRL8}$	data input from clock hold	rising *clk* edge	TBD	-	ns
$t_{FTSRCTRL9}$	input to clock setup	rising *clk* edge	TBD	-	ns
$t_{FTSRCTRL10}$	input from clock hold	rising *clk* edge	TBD	-	ns

38.12 Library dependencies

Table 452 shows libraries used when instantiating the core (VHDL libraries).

*Table 452.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

38.13 Component declaration

The core has the following component declaration.

```
component ftsrctrl is
  generic (
    hindex       : integer := 0;
    romaddr      : integer := 0;
    rommask      : integer := 16#ff0#;
    ramaddr      : integer := 16#400#;
    rammask      : integer := 16#ff0#;
    ioaddr       : integer := 16#200#;
    iomask       : integer := 16#ff0#;
    ramws        : integer := 0;
    romws        : integer := 2;
    iows         : integer := 2;
    rmw          : integer := 0;
    srbanks      : integer range 1 to 8  := 1;
    banksz       : integer range 0 to 15 := 15;
    rombanks     : integer range 1 to 8  := 1;
    rombanksz    : integer range 0 to 15 := 15;
    rombankszdef : integer range 0 to 15 := 15;
    pindex       : integer := 0;
    paddr        : integer := 0;
    pmask        : integer := 16#fff#;
    edacen       : integer range 0 to 1 := 1;
    errcnt       : integer range 0 to 1 := 0;
    cntbits      : integer range 1 to 8 := 1;
    wsreg        : integer := 0;
    oepol        : integer := 0;
    prom8en      : integer := 0
  );
  port (
    rst          : in  std_ulogic;
    clk          : in  std_ulogic;
    ahbsi        : in  ahb_slv_in_type;
    ahbso        : out ahb_slv_out_type;
    apbi         : in  apb_slv_in_type;
    apbo         : out apb_slv_out_type;
    sri          : in  memory_in_type;
    sro          : out memory_out_type;
    sdo          : out sdctrl_out_type
  );
end component;
```

38.14 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined in the example design's port map and connected to the memory controller. System clock and reset are generated by GR Clock Generator and Reset Generator. The CE signal of the memory controller is also connected to the AHB status register.

Memory controller decodes default memory areas: PROM area is 0x0 - 0xFFFFFF and RAM area is 0x40000000 - 0x40FFFFF.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;

    -- memory bus
    address  : out   std_logic_vector(27 downto 0); -- memory bus
    data     : inout std_logic_vector(31 downto 0);
    ramsn    : out   std_logic_vector(4 downto 0);
    ramoen   : out   std_logic_vector(4 downto 0);
    rwen     : inout std_logic_vector(3 downto 0);
    romsn    : out   std_logic_vector(1 downto 0);
    iosn     : out   std_logic;
    oen      : out   std_logic;
    read     : out   std_logic;
    writen   : inout std_logic;
    brdyn    : in    std_logic;
    bexcn    : in    std_logic;
-- sdram i/f
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0); -- optional sdram data
    cb       : inout std_logic_vector(7 downto 0); --checkbits
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo : sdctrl_out_type;

  signal wprot : wprot_out_type;  -- dummy signal, not used
  signal clkm, rstn : std_ulogic; -- system clock and reset

-- signals used by clock and reset generators
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;
```

```
  signal gnd : std_ulogic;

  signal stati : ahbstat_in_type; --correctable error vector

begin

  -- AMBA Components are defined here ...


  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                                tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  rst0 : rstgen
  port map (resetn, clkm, cgo.clklock, rstn);

  -- AHB Status Register
  astat0 : ahbstat generic map(pindex => 13, paddr => 13, pirq => 11,
    nftslv => 1)
    port map(rstn, clkm, ahbmi, ahbsi, stati, apbi, apbo(13));

  stati.cerror(0) <= memo.ce;

  -- Memory controller
  mctrl0 : ftsrctrl generic map (rmw => 1, pindex => 10, paddr => 10,
    edacen => 1, errcnt => 1, cntbits => 4)
    port map (rstn, clkm, ahbsi, ahbso(0), apbi, apbo(10), memi, memo,
  sdo);


  -- I/O pads driving data memory bus data signals
  datapads : for i in 0 to 3 generate
      data_pad : iopadv generic map (width => 8)
      port map (pad => data(31-i*8 downto 24-i*8),
                o => memi.data(31-i*8 downto 24-i*8),
                en => memo.bdrive(i),
                i => memo.data(31-i*8 downto 24-i*8));
  end generate;

  --I/O pads driving checkbit signals
  cb_pad : iopadv generic map (width => 8)
      port map (pad => cb,
                o => memi.cb,
                en => memo.bdrive(0),
                i => memo.cb;

  -- connect memory controller outputs to entity output signals
  address <= memo.address; ramsn <= memo.ramsn; romsn <= memo.romsn;
  oen <= memo.oen; rwen <= memo.wrn; ramoen <= memo.ramoen;
  writen <= memo.writen; read <= memo.read; iosn <= memo.iosn;
  sdcke <= sdo.sdcke; sdwen <= sdo.sdwen; sdcsn <= sdo.sdcsn;
  sdrasn <= sdo.rasn; sdcasn <= sdo.casn; sddqm <= sdo.dqm;

end;
```

39 FTSRCTRL8 - 8-bit SRAM/16-bit IO Memory Controller with EDAC

39.1 Overview

The fault tolerant 8-bit SRAM/16-bit I/O memory interface uses a common 16-bit data bus to interface 8-bit SRAM and 16-bit I/O devices. It provides an Error Detection And Correction unit (EDAC), correcting up to two errors and detecting up to four errors in a data byte. The EDAC eight checkbits are stored in parallel with the 8-bit data in SRAM memory. Configuration of the memory controller functions is performed through the APB bus interface.

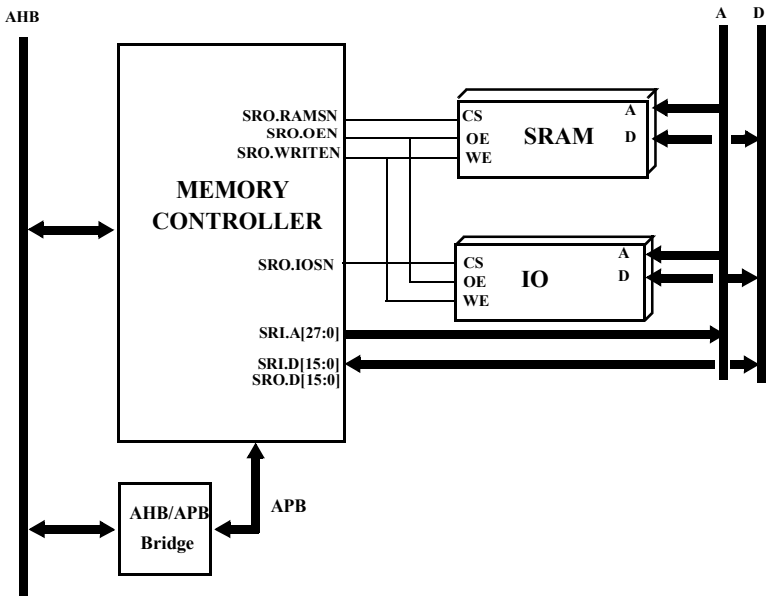


Figure 111. Block diagram

39.2 Operation

The controller is configured through VHDL generics to decode two address ranges: SRAM and I/O area. By default the SRAM area is mapped into address range 0x40000000 - 0x40FFFFFF, and the I/O area is mapped to 0x20000000 - 0x20FFFFFF.

One chip select is decoded for the I/O area, while SRAM can have up to 8 chip select signals. The controller generates a common write-enable signal (WRITEN) for both SRAM and I/O. The number of waitstates may be separately configured for the two address ranges.

The EDAC function is optional, and can be enabled with the edacen VHDL generic. The configuration of the EDAC is done through a configuration register accessed from the APB bus. During nominal operation, the EDAC checksum is generated and checked automatically. The 8-bit input to the EDAC function is split into two 4-bit nibbles. A modified hamming(8,4,4) coding featuring a single error correction and double error detection is applied to each 4-bit nibble. This makes the EDAC capable of correcting up to two errors and detecting up to four errors per 8-bit data. Single errors (correctable errors) are corrected without generating any indication of this condition in the bus response. If a multiple error (uncorrectable errors) is detected, a two cycle error response is given on the AHB bus.

Single errors may be monitored in two ways:

- by monitoring the CE signal which is asserted for one cycle each time a correctable error is detected.
- by checking the single error counter which is accessed from the MCFG3 configuration register.

The CE signal can be connected to the AHB status register which stores information of the AHB instruction causing the error and also generates interrupts. See the AHB status register documentation for more information.

The EDAC function can only be enabled for SRAM area accesses. If a 16-bit or 32-bit bus access is performed, the memory controller calculates the EDAC checksum for each byte read from the memory but the indication of single error is only signaled when the access is done. (I.e. if more than one byte in a 32-bit access has a single error, only one error is indicated for the hole 32-bit access.)

The equations below show how the EDAC checkbits are generated:

```
CB7 = Data[15] ^ Data[14] ^ Data[13]     // i.e. Data[7]
CB6 = Data[15] ^ Data[14] ^ Data[12]     // i.e. Data[6]
CB5 = Data[15] ^ Data[13] ^ Data[12]     // i.e. Data[5]
CB4 = Data[14] ^ Data[13] ^ Data[12]     // i.e. Data[4]
CB3 = Data[11] ^ Data[10] ^ Data[ 9]     // i.e. Data[3]
CB2 = Data[11] ^ Data[10] ^ Data[ 8]     // i.e. Data[2]
CB1 = Data[11] ^ Data[ 9] ^ Data[ 8]     // i.e. Data[1]
CB0 = Data[10] ^ Data[ 9] ^ Data[ 8]     // i.e. Data[0]
```

39.2.1 Memory access

The memory controller supports 32/16/8-bit single accesses and 32-bit burst accesses to the SRAM. A 32-bit or a 16-bit access is performed as multiple 8-bit accesses on the 16-bit memory bus, where data is transferred on data lines 8 to 15 (Data[15:8]). The eight checkbits generated/used by the EDAC are transferred on the eight first data lines (Data[7:0]). For 32-bit and 16-bit accesses, the bytes read from the memory are arranged according to the big-endian order (i.e. for a 32-bit read access, the bytes read from memory address A, A+1, A+2, and A+3 correspond to the bit[31:24], bit[23:16], bit[15:8], and bit[7:0] in the 32-bit word transferred to the AMBA bus. The table 462 shows the expected latency from the memory controller.

*Table 453.*FTSCTRL8 access latency

Accesses	Single data	First data (burst)	Middle data (burst)	Last data (burst)
32-bit write	10	8	8	10
32-bit read	6	6	4	4
16-bit write	4 (+1)	-	-	-
16-bit read	4	-	-	-
8-bit write	4	-	-	-
8-bit read	3	-	-	-

One extra cycle is added for 16-bit burst accesses when Bus Exception is enabled.

39.2.2 I/O access

The memory controller accepts 32/16/8-bit single accesses to the I/O area, but the access generated towards the I/O device is always 16-bit. The two least significant bits of the AMBA address (byte address) determine which half word that should be transferred to the I/O device. (i.e. If the byte address is 0 and it is a 32-bit access, bits 16 to 31 on the AHB bus is transferred on the 16-bit memory bus. If the byte address is 2 and it is a 16-bit access, bit 0 to 15 on the AHB bus is transferred on the 16-bit memory bus.) If the access is an 8-bit access, the data is transferred on data lines 8 to 15 (Data[15:8]) on the memory bus. In case of a write, data lines 0 to 7 is also written to the I/O device but these data lines do not transfer any valid data.

39.2.3 Using Bus Exception

The active low Bus Exception signal (BEXCN) can be used to signal access errors. It is enabled by setting the BEXCEN bit in MCFG1 and is only active for the I/O area. The BEXCN signal is sampled on the same cycle as data is written to memory or read data is sampled. When a bus exception is detected an error response will be generated for the access. One additional latency cycle is added to the AMBA access when the Bus Exception is enable.

39.2.4 Using Bus Ready

The Bus Ready (BRDYN) signal can be used to add waitstates to I/O-area accesses. It is enabled by setting the Bus Ready Enable (BRDYEN) bit in the MCFG1 register. An access will have at least the amount of waitstates set with the VHDL generic or through the register, but will be further stretched until BRDYN is asserted. Additional waitstates can thus be inserted after the pre-set number of wait-states by deasserting the BRDYN signal. BRDYN should be asserted in the cycle preceding the last one. It is recommended that BRDY remains asserted until the IOSN signal is de-asserted, to ensure that the access has been properly completed and avoiding the system to stall.

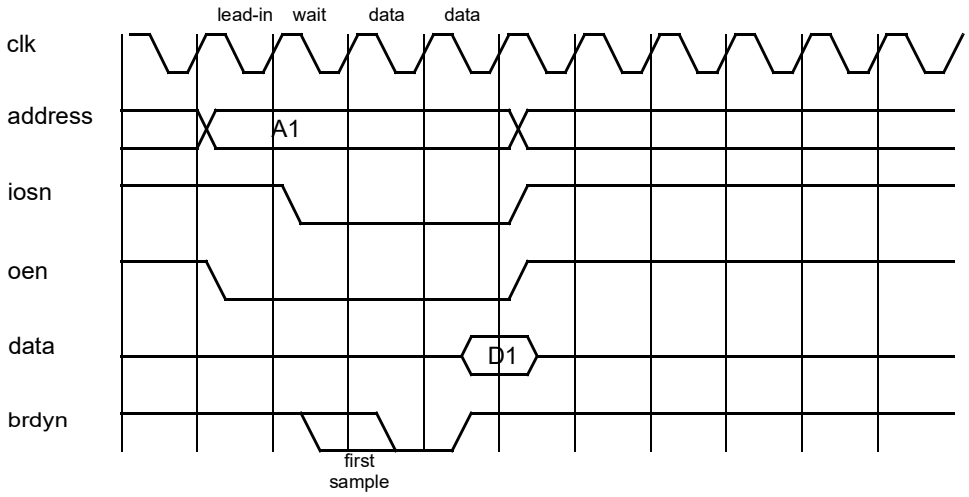


Figure 112. I/O READ cycle, programmed with 1 wait state, and with an extra data cycle added with BRDYN.

39.3 SRAM/IO waveforms

The internal and external waveforms of the interface are presented in the figures below.

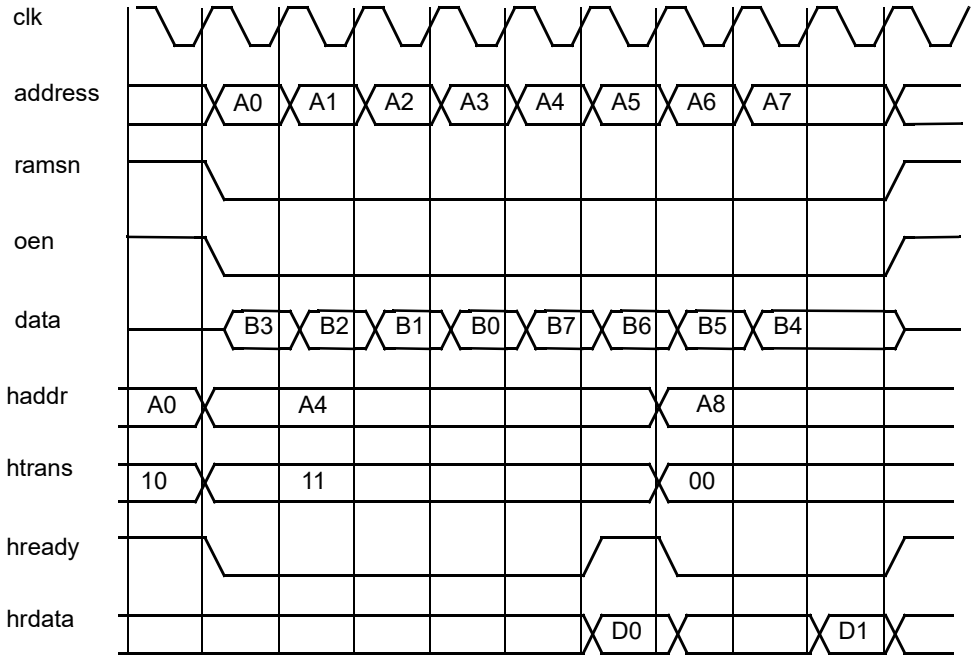


Figure 113. 32-bit SRAM sequential read accesses with 0 wait-states and EDAC enabled.

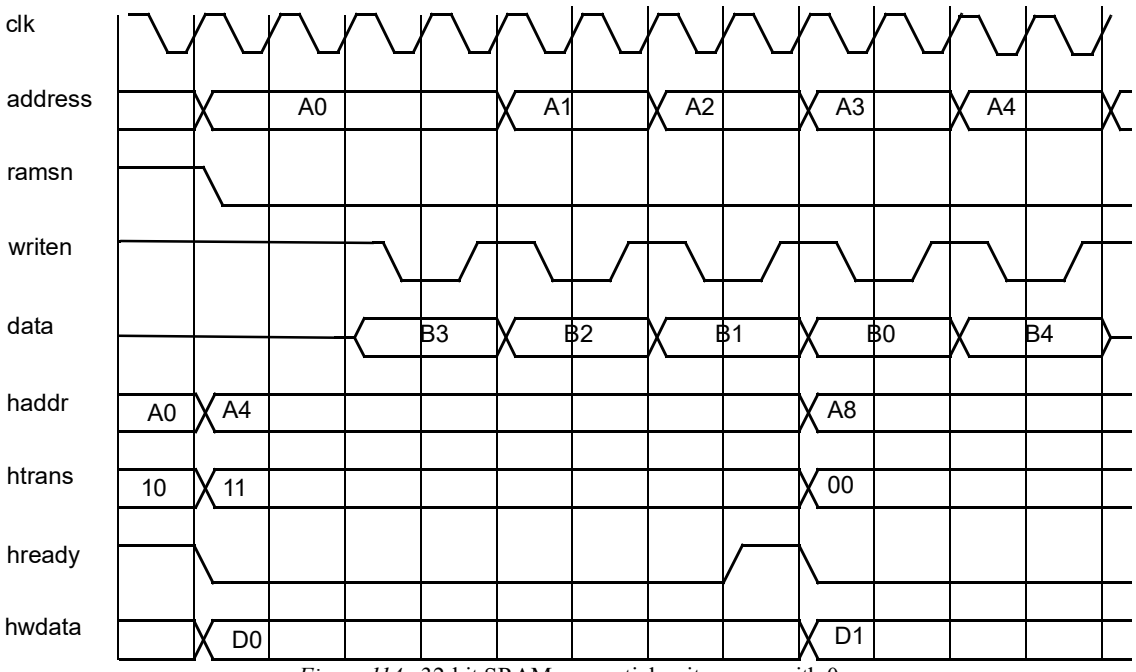


Figure 114. 32-bit SRAM sequential writeaccess with 0 wait-states and EDAC enabled.

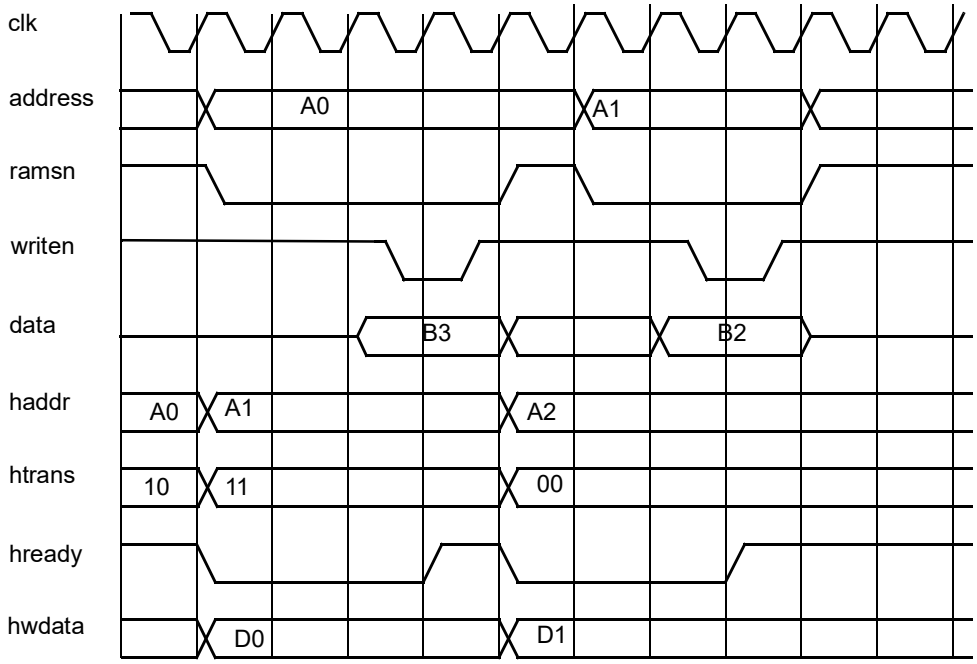


Figure 115. 8-bit SRAM non-sequential write access with 0 wait-states and EDAC enabled.

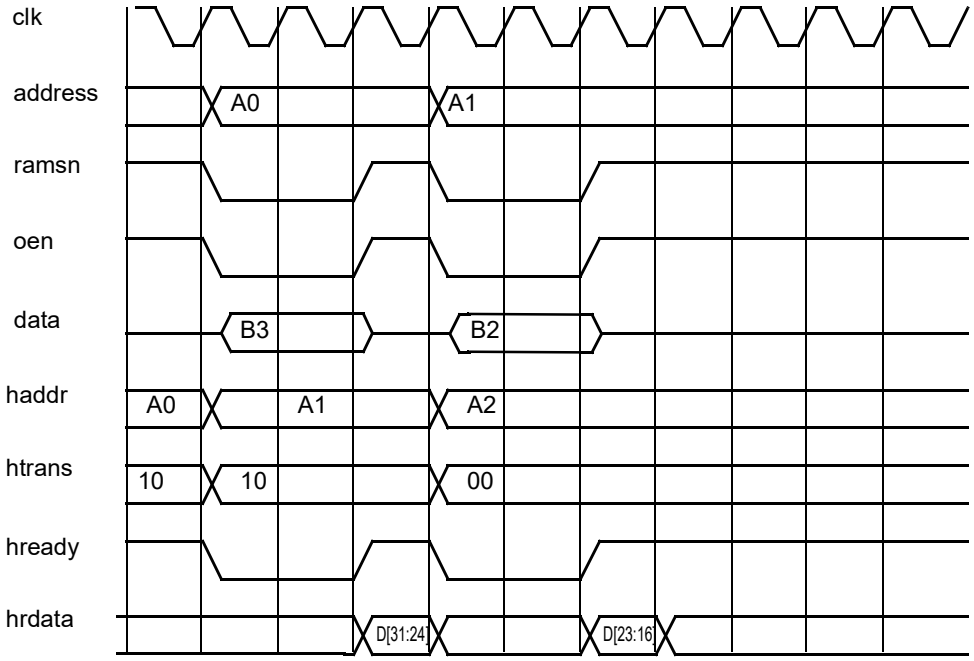


Figure 116. 8-bit SRAM non-sequential read access with 0 wait-states and EDAC enabled.

On a read access, data is sampled one clock cycle before HREADY is asserted.

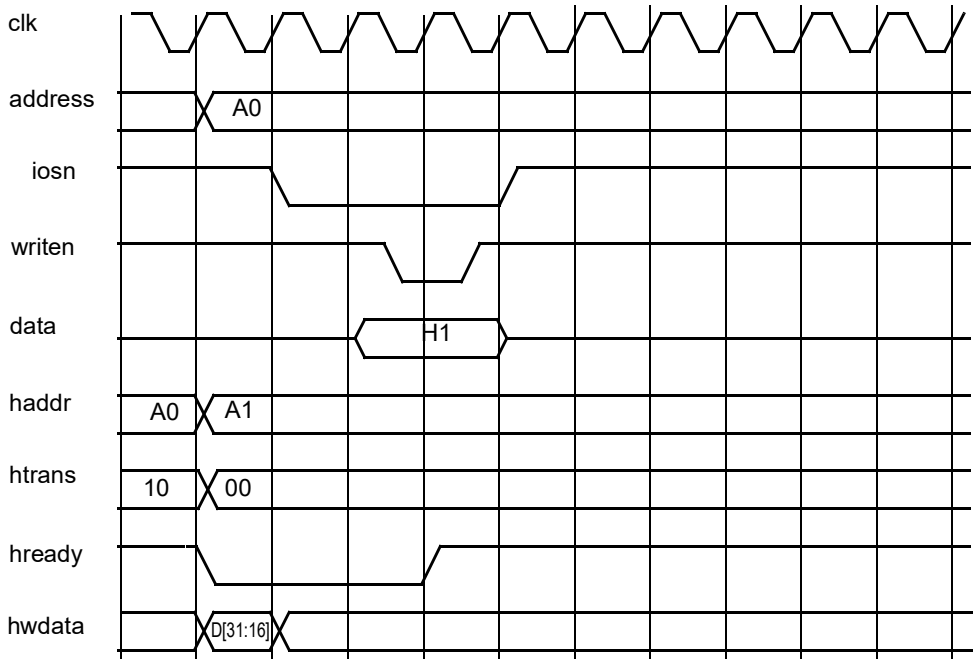


Figure 117. 16-bit I/O non-sequential write access with 0 wait-states.

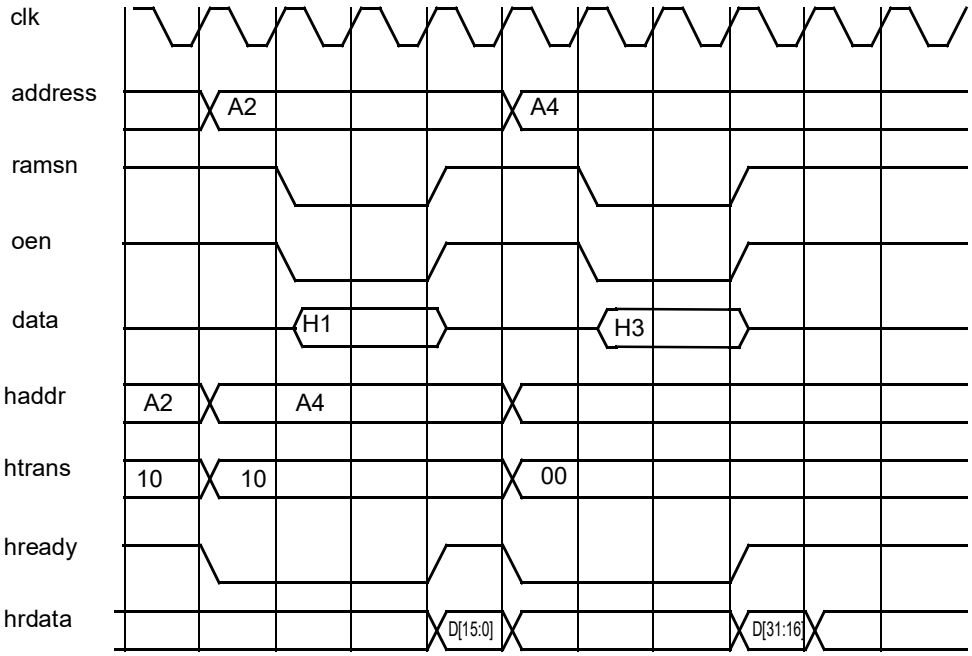


Figure 118. 16-bit I/O non-sequential read access with 0 wait-states.

I/O write accesses are extended with one extra latency cycle if the bus exception is enabled.

If waitstates are configured through the VHDL generics or registers, one extra data cycle will be inserted for each waitstate in both read and write cycles.

39.4 Endianness

The core is designed for big-endian systems.

39.5 Registers

The core is programmed through registers mapped into APB address space.

*Table 454.*FT SRAM/IO controller registers

APB Address offset	Register
0x00	Memory configuration register 1
0x04	Memory configuration register 2
0x08	Memory configuration register 3

39.5.1 Memory Configuration Register 1

*Table 455.*0x00 - MCFG1 - Memory configuration register 1

31 27	26	25	24	23 20	19 0
RESERVED	BRDY	BEXC	R	IOWS	RESERVED
0	0	0	0	0xF	0
r	rw	rw	r	rw	r

31 : 27 RESERVED

26 BRDYEN: Enables the BRDYN signal.

25 BEXCEN: Enables the BEXCN signal.

24 RESERVED

23 : 20 IOWS: Sets the number of waitstates for accesses to the IO area. Only available if the wsreg VHDL generic is set to one.

19 : 0 RESERVED

39.5.2 Memory Configuration Register 2

*Table 456.*0x04 - MCFG2 - Memory configuration register 2

31 13	12 9	8 2	1 0
RESERVED	RAMBSZ	RESERVED	RAMWS
0	*	0	*
r	rw*	r	rw*

31 : 12 RESERVED

12 : 9 RAMBSZ: Sets the SRAM bank size. Only available if the banksz VHDL generic is set to zero. Otherwise the banksz VHDL generic sets the bank size. 0 = 8 kB, 15 = 256 MB.

8 : 2 RESERVED

1 : 0 RAMWS: Sets the number of waitstates for accesses to the RAM area. Only available if the wsreg VHDL generic is set to one.

39.5.3 Memory Configuration 3

*Table 457.*0x08 - MCFG2 - Memory configuration register 3

31 cnt + 13	cnt + 12 12	11	10	9	8	7 0
RESERVED	SEC	WB	RB	SEN		TCB
0	0	0	0	0	0	NR
r	wc	rw	rw	rw	r	rw

31 : cnt+13 RESERVED

cnt+12 : 12 SEC. Single error counter. This field increments each time a single error is detected. It saturates at the maximum value that can be stored in this field. Each bit can be reset by writing a one to it. cnt = the number of counter bits.

11 WB: Write bypass. If set, the TCB field will be used as checkbits in all write operations.

10 RB: Read bypass. If set, checkbits read from memory in all read operations will be stored in the TCB field.

9 SEN: SRAM EDAC enable. If set, EDAC will be active for the SRAM area.

8 RESERVED

7 : 0 TCB: Used as checkbits in write operations when WB is one and checkbits from read operations are stored here when RB is one.

All the fields in the MCFG3 register are available if the edacen VHDL generic is set to one except for the SEC field which also requires that the errcnt VHDL generic is set to one.

39.6 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x056. For description of vendor and device identifiers see the GRLIB IP Library User's Manual.

39.7 Implementation

39.7.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers. The registers driving external chip select, output enable and output enables for the data bus have asynchronous reset.

39.7.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

39.8 Configuration options

Table 454 shows the configuration options of the core (VHDL generics).

Table 458. Controller configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index.	1 - NAHBSLV-1	0
ramaddr	ADDR field of the AHB BAR1 defining RAM address space. Default RAM area is 0x40000000-0x40FFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR1 defining RAM address space.	0 -16#FFF#	16#FF0#
ioaddr	ADDR field of the AHB BAR2 defining IO address space. Default RAM area is 0x20000000-0x20FFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR2 defining IO address space.	0 - 16#FFF#	16#FF0#
ramws	Number of waitstates during access to SRAM area.	0 - 15	0
iows	Number of waitstates during access to IO area.	0 - 15	2
srbanks	Set the number of RAM banks.	1 - 8	1
banksz	Set the size of bank 1 - 4. 1 = 16 kB, ... , 15 = 256 MB. If set to zero, the bank size is set with the rambsz field in the MCFG2 register.	0 - 15	15
pindex	APB slave index.	1 - NAPBSLV-1	0
paddr	APB address.	1 - 16#FFF#	0
pmask	APB address mask.	1 - 16#FFF#	16#FFF#
edacen	EDAC enable. If set to one, EDAC logic is synthesized.	0 - 1	0
errcnt	If one, a single error counter is added.	0 - 1	0
cntbits	Number of bits in the single error counter.	1 - 8	1
wsreg	Enable programmable waitstate generation.	0 - 1	0

39.9 Signal descriptions

Table 459 shows the interface signals of the core (VHDL ports).

*Table 459.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
SRI	DATA[31:0]	Input	Memory data: [15:0] used for IO accesses [7:0] used for checkbits for SRAM accesses [15:8] use for data for SRAM accesses	High
	BRDYN	Input	Bus ready strobe	Low
	BEXCN	Input	Bus exception	Low
	WRN[3:0]	Input	Not used	-
	BWIDTH[1:0]	Input	Not used	-
	SD[31:0]	Input	Not used	-
	CB[7:0]	Input	Not used	-
	PROMDATA[31:0]	Input	Not used	-
	EDAC	Input	Not used	-

Table 459. Signal descriptions

Signal name	Field	Type	Function	Active
SRO	ADDRESS[31:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data: [15:0] used for IO accesses [7:0] used for checkbits for SRAM accesses [15:8] use for data for SRAM accesses	High
	RAMSN[7:0]	Output	SRAM chip-select	Low
	RAMOEN[7:0]	Output	SRAM output enable	Low
	IOSN	Output	IO area chip select	Low
	ROMSN[7:0]	Output	Not used	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[15:8], WRN[1] corresponds to DATA[7:0], WRN[3:2] Not used	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus. Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[15:8], BDRIVE[1] corresponds to DATA[7:0], BDRIVE[3:2] Not used	Low
	VBDRIVE[31:0]	Output	Vectored I/O-pad drive signal.	Low
	READ	Output	Read strobe	High
	RAMN	Output	Common SRAM Chip Select. Always asserted when one of the 8 RAMSN signals is asserted.	Low
	ROMN	Output	Not used	-
	SA[14:0]	Output	Not used	-
	CB[7:0]	Output	Not used	-
	PSEL	Output	Not used	-
	CE	Output	Single error detected.	High
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-

* see GRLIB IP Library User's Manual

39.10 Signal definitions and reset values

The signals and their reset values are described in table 460.

*Table 460.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
address[25:0]	Output	Memory address	High	Undefined
data[31:0]	Input/Output	Memory data	High	Tri-state
cb[7:0]	Input/Output	Check bits	High	Tri-state
ramsn[3:0]	Output	SRAM chip select	Low	Logical 1
ramoen[3:0]	Output	SRAM output enable	Low	Logical 1
rwen[3:0]	Output,	SRAM write enable: rwen[0] corresponds to data[15:8], rwen[1] corresponds to data[7:0], rwen[3:2] Not used	Low	Logical 1
ramben[3:0]	Output	SRAM bank enable: ramben[0] corresponds to data[15:8], ramben[1] corresponds to data[7:8], ramben[3:2] Not used	Low	Logical 1
oen	Output	Output enable	Low	Logical 1
writen	Output	Write strobe	Low	Logical 1
read	Output	Read strobe	High	Logical 1
iosn	Output	IO area chip select	Low	Logical 1
brdyn	Input	Bus ready. Extends accesses to the IO area.	Low	-
bexcn	Input	Bus exception.	Low	-

39.11 Timing

The timing waveforms and timing parameters are shown in figure 119 and are defined in table 461.

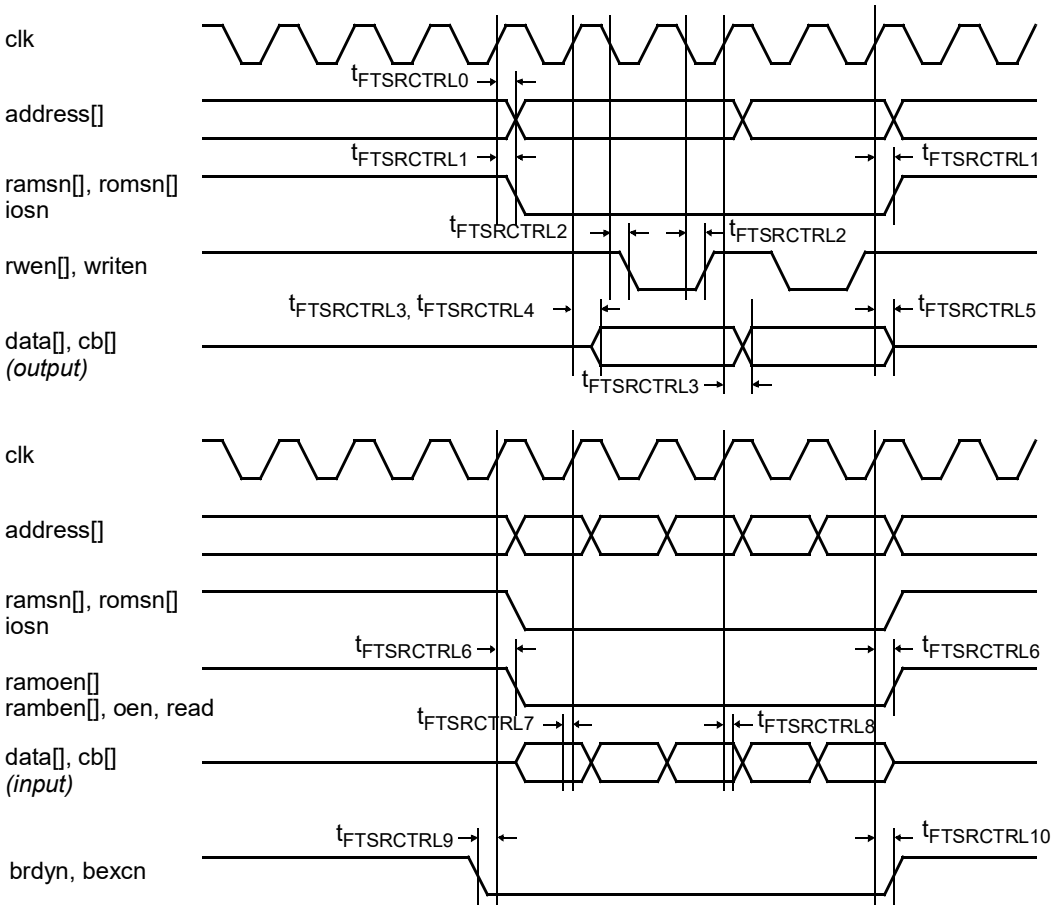


Figure 119. Timing waveforms

*Table 461.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{FTSRCTRL0}$	address clock to output delay	rising clk edge	TBD	TBD	ns
$t_{FTSRCTRL1}$	clock to output delay	rising clk edge	TBD	TBD	ns
$t_{FTSRCTRL2}$	clock to output delay	rising clk edge	TBD	TBD	ns
$t_{FTSRCTRL3}$	clock to data output delay	falling clk edge	TBD	TBD	ns
$t_{FTSRCTRL4}$	clock to data non-tri-state delay	rising clk edge	TBD	TBD	ns
$t_{FTSRCTRL5}$	clock to data tri-state delay	rising clk edge	TBD	TBD	ns
$t_{FTSRCTRL6}$	clock to output delay	rising clk edge	TBD	TBD	ns
$t_{FTSRCTRL7}$	data input to clock setup	rising clk edge	TBD	-	ns
$t_{FTSRCTRL8}$	data input from clock hold	rising clk edge	TBD	-	ns
$t_{FTSRCTRL9}$	input to clock setup	rising clk edge	TBD	-	ns
$t_{FTSRCTRL10}$	input from clock hold	rising clk edge	TBD	-	ns

39.12 Library dependencies

Table 462 shows libraries used when instantiating the core (VHDL libraries).

*Table 462.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

39.13 Component declaration

The core has the following component declaration.

```
component ftsrctrl8 is
  generic (
    hindex       : integer := 0;
    ramaddr      : integer := 16#400#;
    rammask      : integer := 16#ff0#;
    ioaddr       : integer := 16#200#;
    iomask       : integer := 16#ff0#;
    ramws        : integer := 0;
    iows         : integer := 2;
    srbanks      : integer range 1 to 8  := 1;
    banksz       : integer range 0 to 15 := 15;
    pindex       : integer := 0;
    paddr        : integer := 0;
    pmask        : integer := 16#fff#;
    edacen       : integer range 0 to 1 := 1;
    errcnt       : integer range 0 to 1 := 0;
    cntbits      : integer range 1 to 8 := 1;
    wsreg        : integer := 0;
    oepol        : integer := 0
  );
  port (
    rst          : in  std_ulogic;
    clk          : in  std_ulogic;
    ahbsi        : in  ahb_slv_in_type;
    ahbso        : out ahb_slv_out_type;
    apbi         : in  apb_slv_in_type;
    apbo         : out apb_slv_out_type;
    sri          : in  memory_in_type;
    sro          : out memory_out_type
  );
end component;
```

39.14 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined in the example design's port map and connected to the memory controller. The system clock and reset are generated by GR Clock Generator and Reset Generator. The CE signal of the memory controller is also connected to the AHB status register.

The memory controller decodes default memory areas: I/O area is 0x20000000 - 0x20FFFFFF and RAM area is 0x40000000 - 0x40FFFFF.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library techmap;
use techmap.gencomp.all;
library gaisler;
```

```
use gaisler.memctrl.all;
use gaisler.misc.all;

entity ftsrctrl8_ex is
  port (
    resetn     : in  std_ulogic;
    clk        : in  std_ulogic;

    address    : out std_logic_vector(27 downto 0);
    data       : inout std_logic_vector(31 downto 0);
    ramsn      : out std_logic_vector (3 downto 0);
    ramoen     : out std_logic_vector (3 downto 0);
    rwen       : out std_logic_vector (3 downto 0);
    oen        : out std_ulogic;
    writen     : out std_ulogic;
    read       : out std_ulogic;
    iosn       : out std_ulogic;
    brdyn      : in  std_ulogic; -- Bus ready
    bexcn      : in  std_ulogic  -- Bus exception
   );
end;

architecture rtl of ftsrctrl8_ex is
signal memi  : memory_in_type;
signal memo  : memory_out_type;

signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector := (others => apb_none);
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
signal ahbmi : ahb_mst_in_type;
signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

signal clkm, rstn, rstraw : std_ulogic;
signal cgi   : clkgen_in_type;
signal cgo   : clkgen_out_type;

signal stati : ahbstat_in_type;

begin

  -- clock and reset
  cgi.pllctrl <= "00"; cgi.pllrst <= rstraw; cgi.pllref  <= '0';
  clk_pad : clkpad port map (clk, clkm);
  rst0 : rstgen               -- reset generator
  port map (resetn, clkm, '1', rstn, rstraw);

  -- AHB controller
  ahb0 : ahbctrl              -- AHB arbiter/multiplexer
  generic map (rrobin => 1, ioaddr => 16#fff#, devid => 16#201#)
  port map (rstn, clkm, ahbmi, ahbmo, ahbsi, ahbso);

  -- Memory controller
  sr0 : ftsrctrl8 generic map (hindex => 0, pindex => 0, edacen => 1)
    port map (rstn, clkm, ahbsi, ahbso(0), apbi, apbo(0), memi, memo);

  brdyn_pad : inpad port map (brdyn, memi.brdyn);
  bexcn_pad : inpad port map (bexcn, memi.bexcn);

  addr_pad : outpadv generic map (width => 28 )
    port map (address, memo.address(27 downto 0));
  rams_pad : outpadv generic map (width => 4)
    port map (ramsn, memo.ramsn(3 downto 0));
  oen_pad  : outpad
    port map (oen, memo.oen);
  rwen_pad : outpadv generic map (width => 4)
    port map (rwen, memo.wrn);
  roen_pad : outpadv generic map (width => 4)
    port map (ramoen, memo.ramoen(3 downto 0));
  wri_pad  : outpad
    port map (writen, memo.writen);
```

```
  read_pad : outpad
    port map (read, memo.read);
  iosn_pad : outpad
    port map (iosn, memo.iosn);
  data_pad : iopadvv generic map (width => 8) -- SRAM and I/O Data
    port map (data(15 downto 8), memo.data(15 downto 8),
       memo.vbdrive(15 downto 8), memi.data(15 downto 8));
  cbdata_pad : iopadvv generic map (width => 8) -- SRAM checkbits and I/O Data
    port map (data(7 downto 0), memo.data(7 downto 0),
       memo.vbdrive(7 downto 0), memi.data(7 downto 0));

  -- APB bridge and AHB stat
  apb0 : apbctrl              -- AHB/APB bridge
  generic map (hindex => 1, haddr => 16#800#)
    port map (rstn, clkm, ahbsi, ahbso(1), apbi, apbo );

  stati.cerror(0) <= memo.ce;
  ahbstat0 : ahbstat generic map (pindex => 15, paddr => 15, pirq => 1)
    port map (rstn, clkm, ahbmi, ahbsi, stati, apbi, apbo(15));
end;
```

40 GPTIMER - General Purpose Timer Unit

40.1 Overview

The General Purpose Timer Unit provides a common prescaler and decrementing timer(s). The number of timers is configurable through the *ntimers* VHDL generic in the range 1 to 7. The prescaler width is configured through the *sbits* VHDL generic. Timer width is configured through the *nbits* VHDL generic. The timer unit acts a slave on AMBA APB bus. The unit is capable of asserting interrupts on timer underflow. The interrupt to use is configurable to be common for the whole unit or separate for each timer.

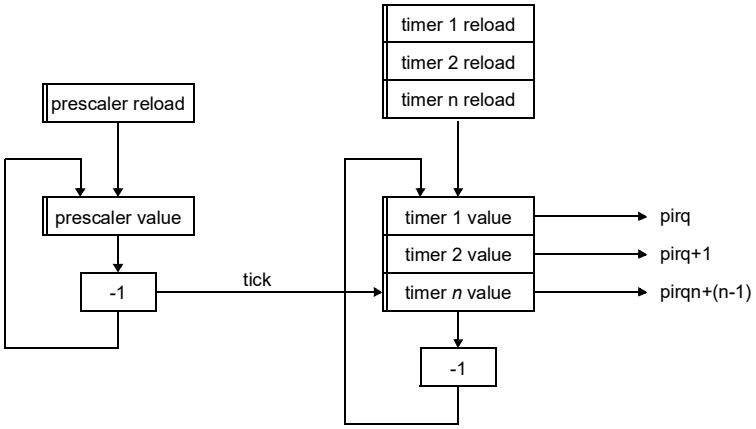


Figure 120. General Purpose Timer Unit block diagram

40.2 Operation

The prescaler is clocked by the system clock and decremented on each clock cycle when at least one timer is enabled. When the prescaler underflows, it is reloaded from the prescaler reload register and a timer tick is generated.

The operation of each timer is controlled through its control register. A timer is enabled by setting the enable bit in the control register. The timer value is then decremented on each prescaler tick. When a timer underflows, it will automatically be reloaded with the value of the corresponding timer reload register if the restart bit in the control register is set, otherwise it will stop at -1 and reset the enable bit.

The timer unit can be configured to a generate shared interrupt line through a VHDL-generic. The shared interrupt will be signaled when any of the timers with interrupt enable bit underflows. The timer unit will signal an interrupt on appropriate line when a timer underflows (if the interrupt enable bit for the current timer is set), when configured to signal interrupt for each timer. The interrupt pending bit in the control register of the underflowed timer will be set and remain set until cleared by writing '1'.

To reduce area, the timers share a single decrementer. A consequence of this is that when a prescaler tick occurs, the individual timers are not decremented on the same clock cycle, but in a staggered manner. If a prescaler tick occurs on system clock cycle *x*, then timer *n* will be decremented on clock cycle *x*+*n*. Another consequence of the shared decrementer is that the minimum allowed prescaler division factor is *ntimers*+1 (reload register = *ntimers*) where *ntimers* is the number of implemented timers.

By setting the chain bit in the control register, timer *n* can be chained with the preceding timer *n*-1. Timer *n* will then be decremented only when timer *n*-1 underflows. This can be used to form a timer wider than 32 bits. Again, the shared decrementer results in timer *n* being decremented one clock

cycle after timer *n*-1. Additionally, the APB interface allows only one timer to be read out per access. This can result in a race-condition when reading out the values of chained timers, because it is possible for one of the timers to be decremented in between readouts.

Each timer can be reloaded with the value in its reload register at any time by writing a 'one' to the load bit in the control register. The last timer acts as a watchdog, driving a watchdog output signal when expired, if the *wdog* VHDL generic is set to a time-out value larger than 0. The watchdog timer also implements a window functionality when the *wdogwin* VHDL generic is set to 1. This enables a decrementing counter which reloads each time the timer is reloaded. If the timer is reloaded and the window counter has not reached zero, this will also assert the watchdog output.

Each timer can be configured to latch its value to a dedicated register when an event is detected on the interrupt (functionality enabled via VHDL generic *glatch*). All timers can be forced to reload when an event is detected on the interrupt bus (functionality enabled via VHDL generic *gset*). A dedicated mask register is provided to filter the interrupts.

At reset, all timers are disabled except the watchdog timer (if enabled by the generics). The prescaler value and reload registers are set to all ones, while the watchdog timer is set to the *wdog* VHDL generic. All other registers are uninitialized except for the WDOGDIS and WDOGNMI fields that are reset to '0'.

40.3 Registers

The core is programmed through registers mapped into APB address space. The number of implemented registers depend on the number of implemented timers.

*Table 463.*General Purpose Timer Unit registers

APB address offset	Register
0x00	Scaler value
0x04	Scaler reload value
0x08	Configuration register
0x0C	Timer latch configuration register
0x10	Timer 1 counter value register
0x14	Timer 1 reload value register
0x18	Timer 1 control register
0x1C	Timer 1 latch register
0x*n*0	Timer *n* counter value register
0x*n*4	Timer *n* reload value register
0x*n*8	Timer *n* control register
0x*n*C	Timer *n* latch register

40.3.1 Scaler Value Register

Table 464. 0x00 - SCALER - Scaler value register

31 16	16-1 0
RESERVED	SCALER
0	all 1
r	rw

16-1: 0 Scaler value. This value will also be set by writes to the Scaler reload value register.
Any unused most significant bits are reserved. Always reads as ‘000...0’.

40.3.2 Scaler Reload Value Register

Table 465. 0x04 - SRELOAD - Scaler reload value register

31 16	16-1 0
RESERVED	SCALER RELOAD VALUE
0	all 1
r	rw

16-1: 0 Scaler reload value. Writes to this register also set the scaler value.
Any unused most significant bits are reserved. Always read as ‘000...0’.

40.3.3 Configuration Register

Table 466. 0x08 - CONFIG - Configuration register

31 23	22 16	15 14	13	12	11	10	9	8	7 3	2 0
"000..0"	TIMEREN	00	EV	ES	EL	EE	DF	SI	IRQ	TIMERS
0	0	0	0	0	0	0	0	*	*	*
r	w	r	rw	rw	rw	rw	rw	r	r	r

31: 23 Reserved. Always reads as '000...0'.

22: 16 Enable bits for each timer. Writing '1' to one of these bits sets the enable bit in the corresponding timer control register. Writing '0' has no effect to the timers. bit[16] corresponds to timer0, bit[17] to timer 1, etc. Write-only.

15: 14 Reserved

13 External Events (EV). If set then the latch events are taken from the secondary input. If this field is zero then the source of the latch events is the interrupt bus.

12 Enable set (ES). If set, on the next matching interrupt, the timers will be loaded with the corresponding timer reload values. The bit is then automatically cleared, not to reload the timer values until set again.

11 Enable latching (EL). If set, on the next matching interrupt, the latches will be loaded with the corresponding timer values. The bit is then automatically cleared, not to load a timer value until set again.

10 Enable external clock source (EE). If set the prescaler is clocked from the external clock source.

9 Disable timer freeze (DF). If set the timer unit can not be freezed, otherwise signal GPTI.DHALT freezes the timer unit.

8 Separate interrupts (SI). Reads '1' if the timer unit generates separate interrupts for each timer, otherwise '0'. Read-only.

7: 3 APB Interrupt: If configured to use common interrupt all timers will drive APB interrupt nr. IRQ, otherwise timer *n* will drive APB Interrupt IRQ+*n* (has to be less the MAXIRQ). Read-only.

Note that the maximum value of this register is 31. If the number of interrupts in the system is set higher than 32, this must be taken into account.

2: 0 Number of implemented timers. Read-only.

40.3.4 Timer Latch Configuration Register

Table 467. 0x0C - CATCHCFG - Timer latch configuration register

31 0
LATCHSEL
0
rw

31: 0 Specifies what bits of the interrupt bus, or external latch vector, bus that shall cause the Timer Latch Registers to latch the timer values. If the configuration register EV field is zero then latching is done based on events on the interrupt bus. If the EV field is '1' then the external latch vecor is used.

40.3.5 Timer N Counter Value Register

Table 468. 0xn0, when n selects the times - TCNTVALn - Timer n counter value register

32-1 0
TCVAL
0
rw

32-1: 0 Timer Counter value. Decremented by 1 for each prescaler tick.

Any unused most significant bits are reserved. Always reads as '000...0'.

40.3.6 Timer N Reload Value Register

*Table 469.*0xn4, when n selects the times - TRLDVALn - Timer n reload value register

32-1	0
TRCDUAL	
*	
rw	

32-1: 0 Timer Reload value. This value is loaded into the timer counter value register when '1' is written to the LD bit in the timer's control register or when the RS bit is set in the control register and the timer underflows.

Any unused most significant bits are reserved. Always reads as '000...0'.

40.3.7 Timer N Control Register

*Table 470.*0xn8, when n selects the times - TCTRLn - Timer n control register

31	16	15	9	8	7	6	5	4	3	2	1	0
WDOGWINC		RESERVED		WS	WN	DH	CH	IP	IE	LD	RS	EN
0		0		0	0	0	0	0	*	0	*	*
rw		r		rw	rw	r	rw	wc	rw	rw	rw	rw

31: 16 Reload value for the watchdog window counter. The window counter is reloaded with this value each time the watchdog counter is reloaded. This functionality is only available when the core has been implemented with VHDL generic *wdog* /= 0, *wdogwin* /= 0 and only for the last timer

15: 9 Reserved. Always reads as '000...0'.

8 Disable Watchdog Output (WS/WDOGDIS): If this field is set to '1' then the GPTO.WDOG and GPTO.WDOGN outputs are disabled (fixed to '0' and '1' respectively). This functionality is only available when the core has been implemented with VHDL generic wdog /= 0 and only for the last timer. If wdog = 0 then this register is read-only and always '0'.

7 Enable Watchdog NMI (WN/WDOGNMI): If this field is set to '1' then the watchdog timer will also generate a non-maskable interrupt (interrupt 15) when an interrupt is signalled. This functionality is only available when the core has been implemented with VHDL generic wdog /= 0 and only for the last timer. If wdog = 0 then this register is read-only and always '0'.

6 Debug Halt (DH): Value of GPTI.DHALT signal which is used to freeze counters (e.g. when a system is in debug mode). Read-only.

5 Chain (CH): Chain with preceding timer. If set for timer *n*, timer *n* will be decremented each time when timer (*n*-1) underflows.

4 Interrupt Pending (IP): The core sets this bit to '1' when an interrupt is signalled. This bit remains '1' until cleared by writing '1' to this bit, writes of '0' have no effect.

3 Interrupt Enable (IE): If set the timer signals interrupt when it underflows. The reset value for this bit is '0' unless watchdog functionality has been enabled. If watchdog functionality has been enabled then this bit for the last timer will have reset value '1'.

2 Load (LD): Load value from the timer reload register to the timer counter value register. This bit is automatically cleared when the value has been loaded.

1 Restart (RS): If set, the timer counter value register is reloaded with the value of the reload register when the timer underflows

0 Enable (EN): Enable the timer.

40.3.8 Timer N Latch Register

*Table 471.*0xnC, when n selects the times - TLATCHn - Timer n latch register

31	0
LTCV	
0	
r	

31: 0 Latched timer counter value (LTCV): Valued latched from corresponding timer. Read-only.

40.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x011. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

40.5 Implementation

40.5.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

40.6 Configuration options

Table 472 shows the configuration options of the core (VHDL generics).

*Table 472.*Configuration options

Generic	Function	Allowed range	Default
pindex	Selects which APB select signal (PSEL) will be used to access the timer unit	0 to NAPBSLV-1	0
paddr	The 12-bit MSB APB address	0 to 4095	0
pmask	The APB address mask	0 to 4095	4095
nbits	Defines the number of bits in the timers	1 to 32	32
ntimers	Defines the number of timers in the unit	1 to 7	1
pirq	Defines which APB interrupt the timers will generate	0 to NAHBIRQ-1	0
sepirq	If set to 1, each timer will drive an individual interrupt line, starting with interrupt *pirq*. If set to 0, all timers will drive the same interrupt line (*pirq*).	0 to 1 (note: *ntimers* + p*irq* must be less than or equal to NAHBIRQ if *sepirq* is set to 1)	0
sbits	Defines the number of bits in the scaler	1 to 32	16
wdog	Watchdog reset value. When set to a non-zero value, the last timer will be enabled and pre-loaded with this value at reset. When the timer value reaches 0, the WDOG output is driven active.	0 to 2^{nbits} - 1	0
ewdogen	External watchdog enable. When set to a non-zero value, the enable bit of the watchdog timer will be set during core reset via the signal gpti.wdogen.Otherwise the enable bit will be set to ‘1’ during core reset.	0 - 1	0
glatch	Enable external timer latch (via interrupt or external vector)	0 - 1	0
gextclk	Enable external timer clock input	0 - 1	0
gset	Enable external timer reload (via interrupt or external vector)	0 - 1	0
gelatch	Enable support for external latch events 0: Timer latch/set is only support for interrupt bus (if enabled via glatch and gset generics) 1: Timer latch/set is disabled after an, unmasked, event on GPTI.LATCHV 2: Timer latch/set is performed on an, unmasked, event on GPTI.LATCHV and timer latch/set is disabled on GPTI.LATCHD events.	0 - 2	0
wdogwin	Enables the watchdog window counter.	0 - 1	0

40.7 Signal descriptions

Table 473 shows the interface signals of the core (VHDL ports).

*Table 473.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
GPTI	DHALT	Input	Freeze timers	High
	EXTCLK	Input	Use as alternative clock	-
	WDOGEN	Input	Sets enable bit of the watchdog timer if VHDL generics wdog and ewdogen are set to non-zero values.	-
	LATCHV[31:0]	Input	External latch/set vector, used if VHDLgeneric gelatch /=0	High
	LATCHD[31:0]	Input	External latch/set disable vector, used if VHDL generic gelatch = 2.	High
GPTO	TICK[0:7]	Output	Timer ticks. TICK[0] is high for one clock each time the scaler underflows. TICK[1-n] are high for one clock each time the corresponding timer underflows.	High
	WDOG	Output	Watchdog output. Equivalent to interrupt pending bit of last timer.	High
	WDOGN	Output	Watchdog output. Equivalent to interrupt pending bit of last timer.	Low

* see GRLIB IP Library User's Manual

40.8 Signal definitions and reset values

One common use of the *wdogn* output is to connect it to an output pad with active low assertion and tri-stated when negated. This allows multiple on-board devices, including the timer watchdog, to generate a system reset if the signal has an external pull-up. To accomplish this the wdogn output of the core can be used as an output enable of the external pad.

When the watchdog times out, the *wdogn* output is driven active low, else it is in tri-state and therefore requires an external pull-up.

The signals and their reset values are described in table 474.

*Table 474.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
wdogn	Tri-state output	Watchdog output. Equivalent to interrupt pending bit of last timer.	Low	Tri-state

40.9 Timing

The timing waveforms and timing parameters are shown in figure 121 and are defined in table 475.

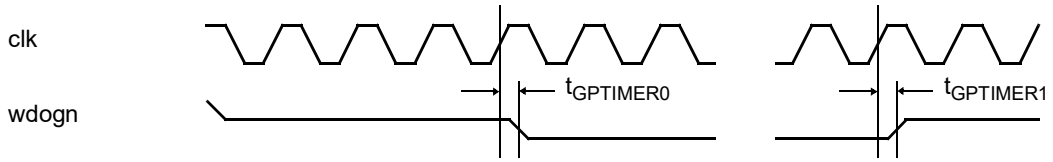


Figure 121. Timing waveforms

*Table 475.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{GPTIMER0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns
$t_{GPTIMER1}$	clock to output tri-state	rising *clk* edge	TBD	TBD	ns

40.10 Library dependencies

Table 476 shows libraries used when instantiating the core (VHDL libraries).

*Table 476.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Signals, component	Component declaration

40.11 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

entity gptimer_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    ... -- other signals
    );
end;

architecture rtl of gptimer_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- GP Timer Unit input signals
  signal gpti : gptimer_in_type;

begin
```

```
  -- AMBA Components are instantiated here
  ...


  -- General Purpose Timer Unit
  timer0 : gptimer
  generic map (pindex => 3, paddr => 3, pirq => 8, sepirq => 1)
  port map (rstn, clk, apbi, apbo(3), gpti, open);

  gpti.dhalt <= '0'; gpti.extclk <= '0'; -- unused inputs

end;
```

41 GR1553B - MIL-STD-1553B / AS15531 Interface

41.1 Overview

This interface core connects the AMBA AHB/APB bus to a single- or dual redundant MIL-STD-1553B bus, and can act as either Bus Controller, Remote Terminal or Bus Monitor.

MIL-STD-1553B (and derived standard SAE AS15531) is a bus standard for transferring data between up to 32 devices over a shared (typically dual-redundant) differential wire. The bus is designed for predictable real-time behavior and fault-tolerance. The raw bus data rate is fixed at 1 Mbit/s, giving a maximum of around 770 kbit/s payload data rate.

One of the terminals on the bus is the Bus Controller (BC), which controls all traffic on the bus. The other terminals are Remote Terminals (RTs), which act on commands issued by the bus controller. Each RT is assigned a unique address between 0-30. In addition, the bus may have passive Bus Monitors (BM:s) connected.

There are 5 possible data transfer types on the MIL-STD-1553 bus:

- BC-to-RT transfer (“receive”)
- RT-to-BC transfer (“transmit”)
- RT-to-RT transfer
- Broadcast BC-to-RTs
- Broadcast RT-to-RTs

Each transfer can contain 1-32 data words of 16 bits each.

The bus controller can also send “mode codes” to the RTs to perform administrative tasks such as time synchronization, and reading out terminal status.

41.2 Electrical interface

The core is connected to the MIL-STD-1553B bus wire through single or dual transceivers, isolation transformers and transformer or stub couplers as shown in figure 122. If single-redundancy is used, the unused bus receive P/N signals should be tied both-high or both-low. The transmitter enables are typically inverted and therefore called transmitter inibit (txinh). See the standard and the respective component’s data sheets for more information on the electrical connection.

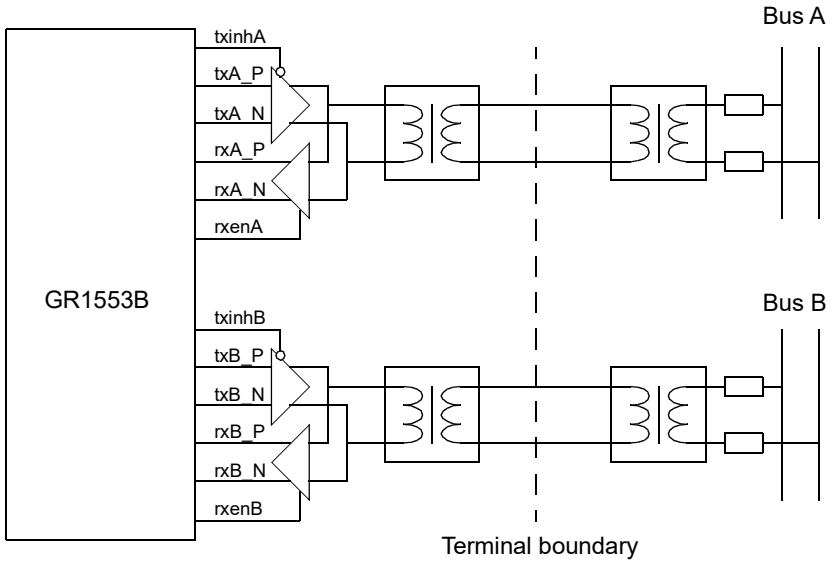


Figure 122. Interface between core and MIL-STD-1553B bus (dual-redundant, transformer coupled)

41.3 Operation

41.3.1 Operating modes

The core contains three separate control units for the Bus Controller, Remote Terminal and Bus Monitor handling, with a shared 1553 codec. All parts may not be present in the hardware, which parts are available can be checked from software by looking at the BCSUP/RTSUP/BMSUP register bits.

The operating mode of the core is controlled by starting and stopping of the BC/RT/BM units via register writes. At start-up, none of the parts are enabled, and the core is completely passive on both the 1553 and AMBA bus.

The BC and RT parts of the core can not be active on the 1553 bus at the same time. While the BC is running or suspended, only the BC (and possibly BM) has access to the 1553 bus, and the RT can only receive and respond to commands when both the BC schedules are completely stopped (not running or even suspended).

The Bus Monitor, however, is only listening on the codec receivers and can therefore operate regardless of the enabled/disabled state of the other two parts.

41.3.2 Register interface

The core is configured and controlled through control registers accessed over the APB bus. Each of the BC,RT,BM parts has a separate set of registers, plus there is a small set of shared registers.

Some of the control register fields for the BC and RT are protected using a ‘key’, a field in the same register that has to be written with a certain value for the write to take effect. The purpose of the keys are to give RT/BM designers a way to ensure that the software can not interfere with the bus traffic by enabling the BC or changing the RT address. If the software is built without knowledge of the key to a certain register, it is very unlikely that it will accidentally perform a write with the correct key to that control register.

41.3.3 Interrupting

The core has one interrupt output, which can be generated from several different source events. Which events should cause an interrupt can be controlled through the IRQ Enable Mask register.

41.3.4 MIL-STD-1553 Codec

The core’s internal codec receives and transmits data words on the 1553 bus, and generates and checks sync patterns and parity.

Loop-back checking logic checks that each transmitted word is also seen on the receive inputs. If the transmitted word is not echoed back, the transmitter stops and signals an error condition, which is then reported back to the user.

Two versions of the codec are available and can be selected at build time through the codecver generic. The new version (codecver=1) provides better noise rejection performance, otherwise there is no functional difference. The new version of the codec is recommended for new designs, and the old version will eventually be deprecated.

41.3.5 Data buffer

The core can be configured to include an additional data buffer that can hold the entire contents (32 words of 16 bits each) of a data transfer.

When enabled, the core’s DMA engine can fetch or store the entire contents of the data buffer in one go rather than fetching or storing one 32-bit word at a time as the transfer is ongoing. The DMA engine will also do bursts on the AHB bus as far as possible to make more efficient use of the AHB bus.

When using the core as bus controller, for BC-to-RT transfers, and the data buffer is enabled, the core will wait for all the outgoing data to be sent to be in the data buffer before strarting the transfer. That way, the BC will operate correctly even if the AHB bus has unpredictable access time in excess of what is described in section 41.20.

41.4 Bus Controller Operation

41.4.1 Overview

When operating as Bus Controller, the core acts as master on the MIL-STD-1553 bus, initiates and performs transfers.

This mode works based on a scheduled transfer list concept. The software sets up in memory a sequence of transfer descriptors and branches, data buffers for sent and received data, and an IRQ pointer ring buffer. When the schedule is started (through a BC action register write), the core processes the list, performs the transfers one after another and writes resulting status into the transfer list and incoming data into the corresponding buffers.

41.4.2 Timing control

In each transfer descriptor in the schedule is a “slot time” field. If the scheduled transfer finishes sooner than its slot time, the core will pause the remaining time before scheduling the next command. This allows the user to accurately control the message timing during a communication frame.

If the transfer uses more than its slot time, the overshooting time will be subtracted from the following command’s time slot. The following command may in turn borrow time from the following command and so on. The core can keep track of up to one second of borrowed time, and will not insert pauses again until the balance is positive, except for intermessage gaps and pauses that the standard requires.

If you wish to execute the schedule as fast as possible you can set all slot times in the schedule to zero. If you want to group a number of transfers you can move all the slot time to the last transfer.

The schedule can be stopped or suspended by writing into the BC action register. When suspended, the schedule’s time will still be accounted, so that the schedule timing will still be correct when the schedule is resumed. When stopped, on the other hand, the schedule’s timers will be reset.

When the extsync bit is set in the schedule’s next transfer descriptor, the core will wait for a positive edge on the external sync input before starting the command. The schedule timer and the time slot balance will then be reset and the command is started. If the sync pulse arrives before the transfer is reached, it is stored so the command will begin immediately. The trigger memory is cleared when stopping (but not when suspending) the schedule. Also, the trigger can be set/cleared by software through the BC action register.

41.4.3 Bus selection

Each transfer descriptor has a bus selection bit that allows you to control on which one of the two redundant buses (‘0’ for bus A, ‘1’ for bus B) the transfer will occur.

Another way to control the bus usage is through the per-RT bus swap register, which has one register bit for each RT address. The bus swap register is an optional feature, software can check the BCFEAT read-only register field to see if it is available.

Writing a ‘1’ to a bit in the per-RT Bus Swap register inverts the meaning of the bus selection bit for all transfers to the corresponding RT, so ‘0’ now means bus ‘B’ and ‘1’ means bus ‘A’. This allows you to switch all transfers to one or a set of RT:s over to the other bus with a single register write and without having to modify any descriptors.

The hardware determines which bus to use by taking the exclusive-or of the bus swap register bit and the bus selection bit. Normally it only makes sense to use one of these two methods for each RT, either the bus selection bit is always zero and the swap register is used, or the swap register bit is always zero and the bus selection bit is used.

If the bus swap register is used for bus selection, the store-bus descriptor bit can be enabled to automatically update the register depending on transfer outcome. If the transfer succeeded on bus A, the bus swap register bit is set to ‘0’, if it succeeds on bus B, the swap register bit is set to ‘1’. If the transfer fails, the bus swap register is set to the opposite value.

41.4.4 Secondary transfer list

The core can be set up with a secondary “asynchronous” transfer list with the same format as the ordinary schedule. This transfer list can be commanded to start at any time during the ordinary schedule. While the core is waiting for a scheduled command’s slot time to finish, it will check if the next asynchronous transfer’s slot time is lower than the remaining sleep time. In that case, the asynchronous command will be scheduled.

If the asynchronous command doesn’t finish in time, time will be borrowed from the next command in the ordinary schedule. In order to not disturb the ordinary schedule, the slot time for the asynchronous messages must therefore be set to pessimistic values.

The exclusive bit in the transfer descriptor can be set if one does not want an asynchronous command scheduled during the sleep time following the transfer.

Asynchronous messages will not be scheduled while the schedule is waiting for a sync pulse or the schedule is suspended and the current slot time has expired, since it is then not known when the next scheduled command will start.

41.4.5 Interrupt generation

Each command in the transfer schedule can be set to generate an interrupt after certain transfers have completed, with or without error. Invalid command descriptors always generate interrupts and stop the schedule. Before a transfer-triggered interrupt is generated, the address to the corresponding descriptor is written into the BC transfer-triggered IRQ ring buffer and the BC Transfer-triggered IRQ Ring Position Register is incremented.

A separate error interrupt signals DMA errors. If a DMA error occurs when reading/writing descriptors, the executing schedule will be suspended. DMA errors in data buffers will cause the corresponding transfer to fail with an error code (see table 480).

Whether any of these interrupt events actually cause an interrupt request on the AMBA bus is controlled by the IRQ Mask Register setting.

41.4.6 Transfer list format

The BC:s transfer list is an array of transfer descriptors mixed with branches as shown in table 477. Each entry has to be aligned to start on a 128-bit (16-byte) boundary. The two unused words in the branch case are free to be used by software to store arbitrary data.

*Table 477.*GR1553B transfer descriptor format

Offset	Value for transfer descriptor	DMA R/W	Value for branch	DMA R/W
0x00	Transfer descriptor word 0 (see table 478)	R	Condition word (see table 482)	R
0x04	Transfer descriptor word 1 (see table 479)	R	Jump address, 128-bit aligned	R
0x08	Data buffer pointer, 16-bit aligned. For write buffers, if bit 0 is set the received data is discarded and the pointer is ignored. This can be used for RT-to-RT transfers where the BC is not interested in the data transferred.	R	Unused	-
0x0C	Result word, written by core (see table 480)	W	Unused	-

The transfer descriptor words are structured as shown in tables 478-480 below.

*Table 478.*GR1553B BC transfer descriptor word 0 (offset 0x00)

31	30	29	28	27	26	25	24	23	22	20	19	18	17	16	15	0
0	WTRIG	EXCL	IRQE	IRQN	SUSE	SUSN	RETMD		NRET		STBUS	GAP	RESERVED		STIME	

31	Must be 0 to identify as descriptor
30	Wait for external trigger (WTRIG)
29	Exclusive time slot (EXCL) - Do not schedule asynchronous messages
28	IRQ after transfer on Error (IRQE)
27	IRQ normally (IRQN) - Always interrupts after transfer
26	Suspend on Error (SUSE) - Suspends the schedule (or stops the async transfer list) on error
25	Suspend normally (SUSN) - Always suspends after transfer
24 : 23	Retry mode (RETMD). 00 - Retry on same bus only. 01 - Retry alternating on both buses 10: Retry first on same bus, then on alternating bus. 11 - Reserved, do not use
22 : 20	Number of retries (NRET) - Number of automatic retries per bus The total number of tries (including the first attempt) is NRET+1 for RETMD=00, 2 x (NRET+1) for RETMD=01/10
19	Store bus (STBUS) - If the transfer succeeds and this bit is set, store the bus on which the transfer succeeded (0 for bus A, 1 for bus B) into the per-RT bus swap register. If the transfer fails and this bit is set, store the opposite bus instead. (only if the per-RT bus mask is supported in the core) See section 41.4.3 for more information.
18	Extended intermessage gap (GAP) - If set, adds an additional amount of gap time, corresponding to the RTTO field, after the transfer
17 : 16	Reserved - Set to 0 for forward compatibility
15 : 0	Slot time (STIME) - Allocated time in 4 microsecond units, remaining time after transfer will insert delay

*Table 479.*GR1553B BC transfer descriptor word 1 (offset 0x04)

31	30	29	26	25	21	20	16	15	11	10	9	5	4	0
DUM	BUS	RTTO		RTAD2		RTSA2		RTAD1		TR	RTSA1		WCMC	

31	Dummy transfer (DUM) - If set to ‘1’ no bus traffic is generated and transfer “succeeds” immediately For dummy transfers, the EXCL,IRQN,SUSN,STBUS,GAP,STIME settings are still in effect, other bits and the data buffer pointer are ignored.	
30	Bus selection (BUS) - Bus to use for transfer, 0 - Bus A, 1 - Bus B	
29:26	RT Timeout (RTTO) - Extra RT status word timeout above nominal in units of 4 us (0000 -14 us, 1111 -74 us). Note: This extra time is also used as extra intermessage gap time if the GAP bit is set.	
25:21	Second RT Address for RT-to-RT transfer (RTAD2)	See table 481 for details on how to setup RTAD1,RTSA1,RTAD2,RTSA2,WCMC,TR for different transfer types. Note that bits 15:0 correspond to the (first) command word on the 1553 bus
20:16	Second RT Subaddress for RT-to-RT transfer (RTSA2)	
15:11	RT Address (RTAD1)	
10	Transmit/receive (TR)	
9:5	RT Subaddress (RTSA1)	
4:0	Word count/Mode code (WCMC)	

Table 480. GR1553B transfer descriptor result word (offset 0x0C)

31	30 24	23 16	15 8	7 4	3	2 0
0	Reserved	RT2ST	RTST	RETCNT	RES	TFRST

31	Always written as 0
30:24	Reserved - Mask away on read for forward compatibility
23:16	RT 2 Status Bits (RT2ST) - Status bits from receiving RT in RT-to-RT transfer, otherwise 0 Same bit pattern as for RTST below
15:8	RT Status Bits (RTST) - Status bits from RT (transmitting RT in RT-to-RT transfer) 15 - Message error, 14 - Instrumentation bit or reserved bit set, 13 - Service request, 12 - Broadcast command received, 11 - Busy bit, 10 - Subsystem flag, 9 - Dynamic bus control acceptance, 8 - Terminal flag
7:4	Retry count (RETCNT) - Number of retries performed
3	Reserved - Mask away on read for forward compatibility
2:0	Transfer status (TFRST) - Outcome of last try 000 - Success (or dummy bit was set) 001 - RT did not respond (transmitting RT in RT-to-RT transfer) 010 - Receiving RT of RT-to-RT transfer did not respond 011 - A responding RT:s status word had message error, busy, instrumentation or reserved bit set (*) 100 - Protocol error (improperly timed data words, decoder error, wrong number of data words) 101 - The transfer descriptor was invalid 110 - Data buffer DMA timeout or error response 111 - Transfer aborted due to loop back check failure

* Error code 011 is issued only when the number of data words match the success case, otherwise code 100 is used. Error code 011 can be issued for a correctly executed “transmit last command” or “transmit last status word” mode code since these commands do not reset the status word.

Table 481. GR1553B BC Transfer configuration bits for different transfer types

Transfer type	**RTAD1 (15:11)**	**RTSA1 (9:5)**	**RTAD2 (25:21)**	**RTSA2 (20:16)**	**WCMC (4:0)**	**TR (10)**	**Data buffer direction**
Data, BC-to-RT	RT address (0-30)	RT subaddr (1-30)	Don’t care	0	Word count (0 for 32)	0	Read (2-64 bytes)
Data, RT-to-BC	RT address (0-30)	RT subaddr (1-30)	Don’t care	0	Word count (0 for 32)	1	Write (2-64 bytes)
Data, RT-to-RT	Recv-RT addr (0-30)	Recv-RT subad. (1-30)	Xmit-RT addr (0-30)	Xmit-RT subad. (1-30)	Word count (0 for 32)	0	Write (2-64 bytes)
Mode, no data	RT address (0-30)	0 or 31 (*)	Don’t care	Don’t care	Mode code (0-8)	1	Unused
Mode, RT-to-BC	RT address (0-30)	0 or 31 (*)	Don’t care	Don’t care	Mode code (16/18/19)	1	Write (2 bytes)
Mode, BC-to-RT	RT address (0-30)	0 or 31 (*)	Don’t care	Don’t care	Mode code (17/20/21)	0	Read (2 bytes)
Broadcast Data, BC-to-RTs	31	RTs subaddr (1-30)	Don’t care	0	Word count (0 for 32)	0	Read (2-64 bytes)
Broadcast Data, RT-to-RTs	31	Recv-RTs subad. (1-30)	Xmit-RT addr (0-30)	Xmit-RT subad. (1-30)	Word count (0 for 32)	0	Write (2-64 bytes)
Broadcast Mode, no data	31	0 or 31 (*)	Don’t care	Don’t care	Mode code (1, 3-8)	1	Unused
Broadcast Mode, BC-to-RT	31	0 or 31 (*)	Don’t care	Don’t care	Mode code (17/20/21)	0	Read (2 bytes)

(*) The standard allows using either of subaddress 0 or 31 for mode commands.

The branch condition word is formed as shown in table 482.

*Table 482.*GR1553B branch condition word (offset 0x00)

31	30 27	26	25	24	23 16	15 8	7 0
1	Reserved (0)	IRQC	ACT	MODE	RT2CC	RTCC	STCC

31	Must be 1 to identify as branch
30 : 27	Reserved - Set to 0
26	Interrupt if condition met (IRQC)
25	Action (ACT) - What to do if condition is met, 0 - Suspend schedule, 1 - Jump
24	Logic mode (MODE): 0 = Or mode (any bit set in RT2CC, RTCC is set in RT2ST,RTST, or result is in STCC mask) 1 - And mode (all bits set in RT2CC,RTCC are set in RT2ST,RTST and result is in STCC mask)
23:16	RT 2 Condition Code (RT2CC) - Mask with bits corresponding to RT2ST in result word of last transfer
15:8	RT Condition Code (RTCC) - Mask with bits corresponding to RTST in result word of last transfer
7:0	Status Condition Code (STCC) - Mask with bits corresponding to status value of last transfer

Note that you can get a constant true condition by setting MODE=0 and STCC=0xFF, and a constant false condition by setting STCC=0x00. 0x800000FF can thus be used as an end-of-list marker.

41.5 Remote Terminal Operation

41.5.1 Overview

When operating as Remote Terminal, the core acts as a slave on the MIL-STD-1553B bus. It listens for requests to its own RT address (or broadcast transfers), checks whether they are configured as legal and, if legal, performs the corresponding transfer or, if illegal, sets the message error flag in the status word. Legality is controlled by the subaddress control word for data transfers and by the mode code control register for mode codes.

To start the RT, set up the subaddress table and log ring buffer, and then write the address and RT enable bit is into the RT Config Register.

41.5.2 Data transfer handling

The Remote Terminal mode uses a three-level structure to handle data transfer DMA. The top level is a subaddress table, where each subaddress has a subaddress control word, and pointers to a transmit descriptor and a receive descriptor. Each descriptor in turn contains a descriptor control/status word, pointer to a data buffer, and a pointer to a next descriptor, forming a linked list or ring of descriptors. Data buffers can reside anywhere in memory with 16-bit alignment.

When the RT receives a data transfer request, it checks in the subaddress table that the request is legal. If it is legal, the transfer is then performed with DMA to or from the corresponding data buffer. After a data transfer, the descriptor's control/status word is updated with success or failure status and the subaddress table pointer is changed to point to the next descriptor.

If logging is enabled, a log entry will be written into a log ring buffer area. A transfer-triggered IRQ may also be enabled. To identify which transfer caused the interrupt, the RT Event Log IRQ Position points to the corresponding log entry. For that reason, logging must be enabled in order to enable interrupts.

If a request is legal but can not be fulfilled, either because there is no valid descriptor ready or because the data can not be accessed within the required response time, the core will signal a RT table access error interrupt and not respond to the request. Optionally, the terminal flag status bit can be automatically set on these error conditions.

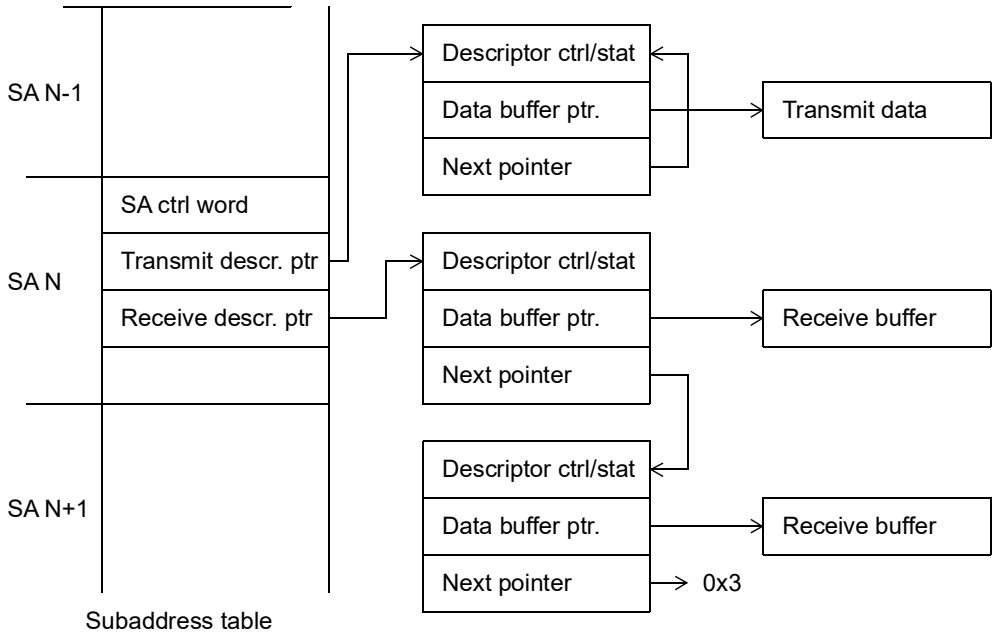


Figure 123. RT subaddress data structure example diagram

41.5.3 Mode Codes

Which of the MIL-STD-1553B mode codes that are legal and should be logged and interrupted are controlled by the RT Mode Code Control register. As for data transfers, to enable interrupts you must also enable logging. Inhibit mode codes are controlled by the same fields as their non-inhibit counterpart and mode codes that can be broadcast have two separate fields to control the broadcast and non-broadcast variants.

The different mode codes and the corresponding action taken by the RT are tabulated below. Some mode codes do not have a built-in action, so they will need to be implemented in software if desired. The relation between each mode code to the fields in the RT Mode Code control register is also shown.

*Table 483.*RT Mode Codes

Mode code		Description	Built-in action, if mode code is enabled	Can log/ IRQ	Enabled after reset	Ctrl. reg bits
0	00000	Dynamic bus control	If the DBCA bit is set in the RT Bus Status register, a Dynamic Bus Control Acceptance response is sent.	Yes	No	17:16
1	00001	Synchronize	The time field in the RT sync register is updated. The output rtsync is pulsed high one AMBA cycle.	Yes	Yes	3:0
2	00010	Transmit status word	Transmits the RT:s status word Enabled always, can not be logged or disabled.	No	Yes	-
3	00011	Initiate self test	No built-in action	Yes	No	21:18
4	00100	Transmitter shutdown	The RT will stop responding to commands on the other bus (not the bus on which this command was given).	Yes	Yes	11:8
5	00101	Override transmitter shutdown	Removes the effect of an earlier transmitter shutdown mode code received on the same bus	Yes	Yes	11:8
6	00110	Inhibit terminal flag	Masks the terminal flag of the sent RT status words	Yes	No	25:22
7	00111	Override inhibit terminal flag	Removes the effect of an earlier inhibit terminal flag mode code.	Yes	No	25:22
8	01000	Reset remote terminal	The fail-safe timers, transmitter shutdown and inhibit terminal flag inhibit status are reset. The Terminal Flag and Service Request bits in the RT Bus Status register are cleared. The extreset output is pulsed high one AMBA cycle.	Yes	No	29:26
16	10000	Transmit vector word	Responds with vector word from RT Status Words Register	Yes	No	13:12
17	10001	Synchronize with data word	The time and data fields in the RT sync register are updated. The rtsync output is pulsed high one AMBA cycle	Yes	Yes	7:4
18	10010	Transmit last command	Transmits the last command sent to the RT. Enabled always, can not be logged or disabled.	No	Yes	-
19	10011	Transmit BIT word	Responds with BIT word from RT Status Words Register	Yes	No	15:14
20	10100	Selected transmitter shutdown	No built-in action	No	No	-
21	10101	Override selected transmitter shutdown	No built-in action	No	No	-

41.5.4 Event Log

The event log is a ring of 32-bit entries, each entry having the format given in table 484. Note that for data transfers, bits 23-0 in the event log are identical to bits 23-0 in the descriptor status word.

*Table 484.*GR1553B RT Event Log entry format

31	30 29	28 24	23 10	9	8 3	2 0
IRQSR	TYPE	SAMC	TIMEL	BC	SZ	TRES

31	IRQ Source (IRQSRC) - Set to '1' if this transfer caused an interrupt
30 : 29	Transfer type (TYPE) - 00 - Transmit data, 01 - Receive data, 10 - Mode code
28 : 24	Subaddress / Mode code (SAMC) - If TYPE=00/01 this is the transfer subaddress, If TYPE=10, this is the mode code
23 : 10	TIMEL - Low 14 bits of time tag counter.
9	Broadcast (BC) - Set to 1 if request was to the broadcast address
8 : 3	Transfer size (SZ) - Count in 16-bit words (0-32)
2 : 0	Transfer result (TRES) 000 = Success 001 = Superseded (canceled because a new command was given on the other bus) 010 = DMA error or memory timeout occurred 011 = Protocol error (improperly timed data words or decoder error) 100 = The busy bit or message error bit was set in the transmitted status word and no data was sent 101 = Transfer aborted due to loop back checker error

41.5.5 Subaddress table format

*Table 485.*GR1553B RT Subaddress table entry for subaddress number N, 0<N<31

Offset	Value	DMA R/W
0x10*N + 0x00	Subaddress N control word (table 486)	R
0x10*N + 0x04	Transmit descriptor pointer, 16-byte aligned (0x3 to indicate invalid pointer)	R/W
0x10*N + 0x08	Receive descriptor pointer, 16-byte aligned (0x3 to indicate invalid pointer)	R/W
0x10*N + 0x0C	Unused	-

Note: The table entries for mode code subaddresses 0 and 31 are never accessed by the core.

*Table 486.*GR1553B RT Subaddress table control word (offset 0x00)

31 19	18	17	16	15	14	13	12 8	7	6	5	4 0
0 (reserved)	WRAP	IGNDV	BCRXE	RXEN	RXLOG	RXIRQ	RXSZ	TXEN	TXLOG	TXIRQ	TXSZ

31 : 19	Reserved - set to 0 for forward compatibility
18	Auto-wraparound enable (WRAP) - Enables a test mode for this subaddress, where transmit transfers send back the last received data. This is done by copying the finished transfer's descriptor pointer to the transmit descriptor pointer address after each successful transfer. Note: If WRAP=1, you should not set TXSZ > RXSZ as this might cause reading beyond buffer end
17	Ignore data valid bit (IGNDV) - If this is '1' then receive transfers will proceed (and overwrite the buffer) if the receive descriptor has the data valid bit set, instead of not responding to the request. This can be used for descriptor rings where you don't care if the oldest data is overwritten.
16	Broadcast receive enable (BCRXEN) - Allow broadcast receive transfers to this subaddress
15	Receive enable (RXEN) - Allow receive transfers to this subaddress
14	Log receive transfers (RXLOG) - Log all receive transfers in event log ring (only used if RXEN=1)
13	Interrupt on receive transfers (RXIRQ) - Each receive transfer will cause an interrupt (only if also RXEN,RXLOG=1)
12 : 8	Maximum legal receive size (RXSZ) to this subaddress - in16-bit words, 0 means 32
7	Transmit enable (TXEN) - Allow transmit transfers from this subaddress
6	Log transmit transfers (TXLOG) - Log all transmit transfers in event log ring (only if also TXEN=1)
5	Interrupt on transmit transfers (TXIRQ) - Each transmit transfer will cause an interrupt (only if TXEN,TXLOG=1)
4 : 0	Maximum legal transmit size (TXSZ) from this subaddress - in 16-bit words, 0 means 32

*Table 487.*GR1553B RT Descriptor format

Offset	Value	DMA R/W
0x00	Control and status word, see table 488	R/W
0x04	Data buffer pointer, 16-bit aligned	R
0x08	Pointer to next descriptor, 16-byte aligned or 0x0000003 to indicate end of list	R

*Table 488.*GR1553B RT Descriptor control/status word (offset 0x00)

31	30	29 26	25 10	9	8 3	2 0
DV	IRQEN	Reserved (0)	TIME	BC	SZ	TRES

31	Data valid (DV) - Should be set to 0 by software before and set to 1 by hardware after transfer. If DV=1 in the current receive descriptor before the receive transfer begins then a descriptor table error will be triggered. You can override this by setting the IGNDV bit in the subaddress table.
30	IRQ Enable override (IRQEN) - Log and IRQ after transfer regardless of SA control word settings Can be used for getting an interrupt when nearing the end of a descriptor list.
29 : 26	Reserved - Write 0 and mask out on read for forward compatibility
25 : 10	Transmission time tag (TTIME) - Set by the core to the value of the RT timer when the transfer finished.
9	Broadcast (BC) - Set by the core if the transfer was a broadcast transfer
8 : 3	Transfer size (SZ) - Count in 16-bit words (0-32)
2 : 0	Transfer result (TRES) 000 = Success 001 = Superseded (canceled because a new command was given on the other bus) 010 = DMA error or memory timeout occurred 011 = Protocol error (improperly timed data words or decoder error) 100 = The busy bit or message error bit was set in the transmitted status word and no data was sent 101 = Transfer aborted due to loop back checker error

41.6 Bus Monitor Operation

41.6.1 Overview

The Bus Monitor (BM) can be enabled by itself, or in parallel to the BC or RT. The BM acts as a passive logging device, writing received data with time stamps to a ring buffer.

41.6.2 Filtering

The Bus Monitor can also support filtering. This is an optional feature, software can check for this by testing whether the BM filter registers are writable.

Transfers can be filtered per RT address and per subaddress or mode code, and the filter conditions are logically AND:ed. If all bits of the three filter registers and bits 2-3 of the control register are set to '1', the BM core will log all words that are received on the bus.

In order to filter on subaddress/mode code, the BM has logic to track 1553 words belonging to the same message. All 10 message types are supported. If an unexpected word appears, the filter logic will restart. Data words not appearing to belong to any message can be logged by setting a bit in the control register.

The filter logic can be manually restarted by setting the BM enable bit low and then back to high. This feature is mainly to improve testability of the BM itself.

The filtering capability can be configured out of the BM to save area. If this is done, all words seen are logged and the filter control registers become read-only and always read out as all-ones. You can, however, still control whether Manchester/parity errors are logged.

41.6.3 No-response handling

In the MIL-STD-1553B protocol, a command word for a mode code using indicator 0 or a regular transfer to subaddress 8 has the same structure as a legal status word. Therefore ambiguity can arise when the subaddress or mode code filters are used, an RT is not responding on a subaddress, and the BC then commands the same RT again on subaddress 8 or mode code indicator 0 on the same bus. This can lead to the second command word being interpreted as a status word and filtered out.

The BM can use the instrumentation bit and reserved bits to disambiguate, which means that this case will never occur when subaddresses 1-7, 9-30 and mode code indicator 31 are used. Also, this case does not occur when the subaddress/mode code filters are unused and only the RT address filter is used.

41.6.4 Log entry format

Each log entry is two 32-bit words.

Table 489. GR1553B BM Log entry word 0 (offset 0x00)

31	30 24	23 0
1	Reserved	TIME

31	Always written as 1
30 : 24	Reserved - Mask out on read for forward compatibility
23 : 0	Time tag (TIME)

Table 490. GR1553B BM Log entry word 1 (offset 0x04)

31	30 ... 20	19	18 ... 17	16	15 ... 0
0	Reserved	BUS	WST	WTP	WD

31	Always written as 0
30 : 20	Reserved - Mask out on read for forward compatibility
19	Receive data bus (BUS) - 0:A, 1:B
18 : 17	Word status (WST) - 00=word OK, 01=Manchester error, 10=Parity error
16	Word type (WTP) - 0:Data, 1:Command/status
15 : 0	Word data (WD)

41.7 Clocking

The core needs a separate clock for the 1553 codec.

The core operates in two clock domains, the AMBA clock domain and the 1553 codec clock domain, with synchronization and handshaking between the domains. The AMBA clock can be at any frequency but must be at a minimum of 10 MHz. A propagation delay of up to one codec clock cycle (50 ns) can be tolerated in each clock-domain crossing signal.

The core has two separate reset inputs for the two clock domains. They should be reset simultaneously, for instance by using two Reset generator cores connected to the same reset input but clocked by the respective clocks.

41.8 AXI support

The core is designed for an AMBA system but can be adapted for AXI using the AHBM2AXI adapter.

41.9 Registers

The core is programmed through registers mapped into APB address space. If the RT, BC or BM parts of the core have been configured out, the corresponding registers will become unimplemented and return zero when read. Reserved register fields should be written as zeroes and masked out on read.

Table 491. MIL-STD-1553B interface registers

APB address offset	Register	R/W	Reset value
0x00	IRQ Register	RW (write '1' to clear)	0x00000000
0x04	IRQ Enable	RW	0x00000000
0x08...0x0F	(Reserved)		
0x10	Hardware config register	R (constant) **	0x00000000*
0x14...0x3F	(Reserved)		
0x40...0x7F	BC Register area (see table 492)		
0x80...0xBF	RT Register area (see table 493)		
0xC0...0xFF	BM Register area (see table 494)		

(*) May differ depending on core configuration

(**) Read-writable in special case of codecver=2

*Table 492.*MIL-STD-1553B interface BC-specific registers

APB address offset	Register	R/W	Reset value
0x40	BC Status and Config register	RW	0xf0000000*
0x44	BC Action register	W	
0x48	BC Transfer list next pointer	RW	0x00000000
0x4C	BC Asynchronous list next pointer	RW	0x00000000
0x50	BC Timer register	R	0x00000000
0x54	BC Timer wake-up register	RW	0x00000000
0x58	BC Transfer-triggered IRQ ring position	RW	0x00000000
0x5C	BC Per-RT bus swap register	RW	0x00000000
0x60...0x67	(Reserved)		
0x68	BC Transfer list current slot pointer	R	0x00000000
0x6C	BC Asynchronous list current slot pointer	R	0x00000000
0x70...0x7F	(Reserved)		

(*) May differ depending on core configuration

*Table 493.*MIL-STD-1553B interface RT-specific registers

APB address offset	Register	R/W	Reset value
0x80	RT Status register	R	0x80000000*
0x84	RT Config register	RW	0x0000e03e***
0x88	RT Bus status bits register	RW	0x00000000
0x8C	RT Status words register	RW	0x00000000
0x90	RT Sync register	R	0x00000000
0x94	RT Subaddress table base address	RW	0x00000000
0x98	RT Mode code control register	RW	0x00000555
0x9C...0xA3	(Reserved)		
0xA4	RT Time tag control register	RW	0x00000000
0xA8	(Reserved)		
0xAC	RT Event log size mask	RW	0xfffffffc
0xB0	RT Event log position	RW	0x00000000
0xB4	RT Event log interrupt position	R	0x00000000
0xB8.. 0xBF	(Reserved)		

(*) May differ depending on core configuration

(***) Reset value is affected by the external RTADDR/RTPAR input signals

*Table 494.*MIL-STD-1553B interface BM-specific registers

APB address offset	Register	R/W	Reset value
0xC0	BM Status register	R	0x80000000*
0xC4	BM Control register	RW	0x00000000
0xC8	BM RT Address filter register	RW	0xffffffff
0xCC	BM RT Subaddress filter register	RW	0xffffffff
0xD0	BM RT Mode code filter register	RW	0xffffffff
0xD4	BM Log buffer start	RW	0x00000000
0xD8	BM Log buffer end	RW	0x00000007
0xDC	BM Log buffer position	RW	0x00000000
0xE0	BM Time tag control register	RW	0x00000000
0xE4...0xFF	(Reserved)		

(*) May differ depending on core configuration

41.9.1 IRQ Register

*Table 495.*0x00 - IRQ - GR1553B IRQ Register

31 18	17	16	15 11	10	9	8	7 3	2	1	0
RESERVED	BMTOF	BMD	RESERVED	RTTE	RTD	RTEV	RESERVED	BCWK	BCD	BCEV
0	0	0	0	0	0	0	0	0	0	0
r	wc	wc	r	wc	wc	wc	r	wc	wc	wc

Bits read ‘1’ if interrupt occurred, write back ‘1’ to acknowledge

17 BM Timer overflow (BMTOF)

16 BM DMA Error (BMD)

10 RT Table access error (RTTE)

9 RT DMA Error (RTD)

8 RT transfer-triggered event interrupt (RTEV)

2 BC Wake-up timer interrupt (BCWK)

1 BC DMA Error (BCD)

0 BC Transfer-triggered event interrupt (BCEV)

41.9.2 IRQ Enable Register

*Table 496.*0x04 - IRQE - GR1553B IRQ Enable Register

31 18	17	16	15 11	10	9	8	7 3	2	1	0
RESERVED	BMTOE	BMDE	RESERVED	RTTEE	RTDE	RTEVE	RESERVED	BCWKE	BCDE	BCEVE
0	0	0	0	0	0	0	0	0	0	0
r	rw	rw	r	rw	rw	rw	r	rw	rw	rw

17 BM Timer overflow interrupt enable (BMTOE)

16 BM DMA error interrupt enable (BMDE)

10 RT Table access error interrupt enable (RTTEE)

9 RT DMA error interrupt enable (RTDE)

8 RT Transfer-triggered event interrupt enable (RTEVE)

2 BC Wake up timer interrupt (BCWKE)

1 BC DMA Error Enable (BCDE)

0 BC Transfer-triggered event interrupt (BCEVE)

41.9.3 Hardware Configuration Register

*Table 497.*GR1553B Hardware Configuration Register

31	30 14	13	12	11	10 9	8	7 0
MOD	RESERVED	DBUF	CVER	XKEYS	ENDIAN	SCLK	CCFREQ
*	0	*	*	*	*	*	‘
r	r	rw	r(w)	r	r	r	r

31 Modified (MOD) - Reserved to indicate that the core has been modified / customized in an unspecified manner

13 Data buffer enable (DBUF) - set to 1 to enable the data buffer function. This bit is writable and defaults to 1 if the data buffer feature has been enabled in the hardware, otherwise it will always read as 0.

12 Codec version (CVER) - 0=Old version, 1=New version. If the core has been set to soft-configurable version (codecver=2) this bit is also writable.

11 Set if safety keys are enabled for the BM Control Register and for all RT Control Register fields.

10 : 9 AHB Endianness - 00=Big-endian, 01=Little-endian, 10/11=Reserved

8 Same clock (SCLK) - Reserved for future versions to indicate that the core has been modified to run with a single clock

7 : 0 Codec clock frequency (CCFREQ) - Reserved for future versions of the core to indicate that the core runs at a different codec clock frequency. Frequency value in MHz, a value of 0 means 20 MHz.

41.9.4 BC Status and Config Register

*Table 498.*0x40 - BCSL - GR1553B BC Status and Config Register

31	30 28	27 17	16	15 11	10	9 8	7 3	2 0
BCSUP	BCFEAT	RESERVED	BCCHK	ASADL	R	ASST	SCADL	SCST
*	*	0	0	0	0	0	0	0
r	r	r	rw	r	r	r	r	r

31 BC Supported (BCSUP) - Reads '1' if core supports BC mode

30 : 28 BC Features (BCFEAT) - Bit field describing supported optional features ('1'=supported):

- 30 BC Schedule timer supported
- 29 BC Schedule time wake-up interrupt supported
- 28 BC per-RT bus swap register and STBUS descriptor bit supported

16 Check broadcasts (BCCHK) - Writable bit, if set to '1' enables waiting and checking for (unexpected) responses to all broadcasts.

15 : 11 Asynchronous list address low bits (ASADL) - Bit 8-4 of currently executing (if ASST=01) or next asynchronous command descriptor address

9 : 8 Asynchronous list state (ASST) - 00=Stopped, 01=Executing command, 10=Waiting for time slot

7 : 3 Schedule address low bits (SCADL) - Bit 8-4 of currently executing (if SCST=001) or next schedule descriptor address

2 : 0 Schedule state (SCST) - 000=Stopped, 001=Executing command, 010=Waiting for time slot, 011=Suspended, 100=Waiting for external trigger

41.9.5 BC Action Register

*Table 499.*0x44 - BCA - GR1553B BC Action Register

31 16	15 10	9	8	7 5	4	3	2	1	0
BCKEY	RESERVED	ASSTP	ASSRT	RESERVED	CLRT	SETT	SCSTP	SCSUS	SCSRT
-	-	-	-	-	-	-	-	-	-
w	-	w	w	-	w	w	w	w	w

31 : 16 Safety code (BCKEY) - Must be 0x1552 when writing, otherwise register write is ignored

9 Asynchronous list stop (ASSTP) - Write '1' to stop asynchronous list (after current transfer, if executing)

8 Asynchronous list start (ASSRT) - Write '1' to start asynchronous list

4 Clear external trigger (CLRT) - Write '1' to clear trigger memory

3 Set external trigger (SETT) - Write '1' to force the trigger memory to set

2 Schedule stop (SCSTP) - Write '1' to stop schedule (after current transfer, if executing)

1 Schedule suspend (SCSUS) - Write '1' to suspend schedule (after current transfer, if executing)

0 Schedule start (SCSRT) - Write '1' to start schedule

41.9.6 BC Transfer List Next Pointer Register

*Table 500.*0x48 - BCTNP - GR1553B BC Transfer list next pointer register

31 0
SCHEDULE TRANSFER LIST POINTER
0
rw

31 : 0 Read: Currently executing (if SCST=001) or next transfer to be executed in regular schedule.
Write: Change address. If running, this will cause a jump after the current transfer has finished.

41.9.7 BC Asynchronous List Next Pointer Register

*Table 501.*0x4C - BCANP - GR1553B BC Asynchronous list next pointer register

31	0
ASYNCHRONOUS LIST POINTER	
0	
rw	

31 :0 Read: Currently executing (if ASST=01) or next transfer to be executed in asynchronous schedule.
Write: Change address. If running, this will cause a jump after the current transfer has finished.

41.9.8 BC Timer Register

*Table 502.*0x50 - BCT - GR1553B BC Timer register

31 24	23 0
RESERVED	SCHEDULE TIME (SCTM)
0	0
r	r

23 : 0 Elapsed “transfer list” time in microseconds (read-only)
Set to zero when schedule is stopped or on external sync.

Note: This register is an optional feature, see BC Status and Config Register, bit 30

41.9.9 BC Timer Wake-up Register

*Table 503.*0x54 - BCTW - GR1553B BC Timer Wake-up register

31	30 24	23 0
WKEN	RESERVED	WAKE-UP TIME (WKTM)
0	0	0
rw	r	rw

31 Wake-up timer enable (WKEN) - If set, an interrupt will be triggered when WKTM=SCTM

23 : 0 Wake-up time (WKTM).

Note: This register is an optional feature, see BC Status and Config Register, bit 29

41.9.10 BC Transfer-triggered IRQ Ring Position Register

*Table 504.*0x58 - BCRD - GR1553B BC Transfer-triggered IRQ ring position register

31	0
BC IRQ SOURCE POINTER RING POSITION	
0	
rw	

31 : 0 The current write pointer into the transfer-tirggered IRQ descriptor pointer ring.
Bits 1:0 are constant zero (4-byte aligned)
The ring wraps at the 64-byte boundary, so bits 31:6 are only changed by user

41.9.11 BC per-RT Bus Swap Register

*Table 505.*0x5C - BCBS - GR1553B BC per-RT Bus swap register

31	0
BC PER-RT BUS SWAP	
0	
rw	

31 : 0 The bus selection value will be logically exclusive-or:ed with the bit in this mask corresponding to the addressed RT (the receiving RT for RT-to-RT transfers). This register gets updated by the core if the STBUS descriptor bit is used.

For more information on how to use this feature, see section 41.4.3.

Note: This register is an optional feature, see BC Status and Config Register, bit 28

41.9.12 BC Transfer List Current Slot Pointer

*Table 506.*0x68 - BCTCP - GR1553B BC Transfer list current slot pointer

31	0
BC TRANSFER SLOT POINTER	
0	
r	

31 : 0 Points to the transfer descriptor corresponding to the current time slot (read-only, only valid while transfer list is running).

Bits 3:0 are constant zero (128-bit/16-byte aligned)

41.9.13 BC Asynchronous List Current Slot Pointer

*Table 507.*0x6C - BCACP - GR1553B BC Asynchronous list current slot pointer

31	0
BC TRANSFER SLOT POINTER	
0	
r	

31 : 0 Points to the transfer descriptor corresponding to the current asynchronous schedule time slot (read-only, only valid while asynchronous list is running).

Bits 3:0 are constant zero (128-bit/16-byte aligned)

41.9.14 RT Status Register

*Table 508.*0x80 - RTS - GR1553B RT Status register (read-only)

31	30	4	3	2	1	0
RTSUP	RESERVED		ACT	SHDA	SHDB	RUN

31 RT Supported (RTSUP) - Reads ‘1’ if core supports RT mode

3 RT Active (ACT) - ‘1’ if RT is currently processing a transfer

2 Bus A shutdown (SHDA) - Reads ‘1’ if bus A has been shut down by the BC (using the transmitter shutdown mode command on bus B)

1 Bus B shutdown (SHDB) - Reads ‘1’ if bus B has been shut down by the BC (using the transmitter shutdown mode command on bus A)

0 RT Running (RUN) - ‘1’ if the RT is listening to commands.

41.9.15 RT Config Register

*Table 509.*0x84 - RTC - GR1553B RT Config register

31 16	15	14	13	12 7	6	5 1	0
RTKEY	SYS	SYDS	BRS	RESERVED	RTEIS	RTADDR	RTEN
0	1	1	1	0	*	*	0
w	rw	rw	rw	r	r	rw	rw

31 : 16	Safety code (RTKEY) - Must be written as 0x1553 when changing the RT address, otherwise the address field is unaffected by the write. When reading the register, this field reads 0x0000. If extra safety keys are enabled (see Hardware Config Register), the lower half of the key is used to also protect the other fields in this register.
15	Sync signal enable (SYS) - Set to '1' to pulse the rtsync output when a synchronize mode code (without data) has been received
14	Sync with data signal enable (SYDS) - Set to '1' to pulse the rtsync output when a synchronize with data word mode code has been received
13	Bus reset signal enable (BRS) - Set to '1' to pulse the busreset output when a reset remote terminal mode code has been received.
6	Reads '1' if current address was set through external inputs. After setting the address from software this field is set to '0'
5 : 1	RT Address (RTADDR) - This RT:s address (0-30)
0	RT Enable (RTEN) - Set to '1' to enable listening for requests

41.9.16 RT Bus Status Register

*Table 510.*0x88 - RTBS - GR1553B RT Bus status register

31 9	8	7 5	4	3	2	1	0
RESERVED	TFDE	RESERVED	SREQ	BUSY	SSF	DBCA	TFLG
0	0	0	0	0	0	0	0
r	rw	rw	rw	rw	rw	rw	rw

8	Set Terminal flag automatically on DMA and descriptor table errors (TFDE)
4 : 0	These bits will be sent in the RT:s status responses over the 1553 bus.
4	Service request (SREQ)
3	Busy bit (BUSY) Note: If the busy bit is set, the RT will respond with only the status word and the transfer "fails"
2	Subsystem Flag (SSF)
1	Dynamic Bus Control Acceptance (DBCA) Note: This bit is only sent in response to the Dynamic Bus Control mode code
0	Terminal Flag (TFLG) The BC can mask this flag using the "inhibit terminal flag" mode command, if legal

41.9.17 RT Status Words Register

*Table 511.*0x8C - RTSW - GR1553B RT Status words register

31 16	15 0
BIT WORD (BITW)	VECTOR WORD (VECW)
0	0
rw	rw

31 : 16	BIT Word - Transmitted in response to the "Transmit BIT Word" mode command, if legal
15 : 0	Vector word - Transmitted in response to the "Transmit vector word" mode command, if legal.

41.9.18 RT Sync Register

*Table 512.*0x90 - RTSY - GR1553B RT Sync register

31 16	15 0
SYNC TIME (SYTM)	SYNC DATA (SYD)
0	0
r	r

31 : 16	The value of the RT timer at the last sync or sync with data word mode command, if legal.
15 : 0	The data received with the last synchronize with data word mode command, if legal

41.9.19 Sub Address Table Base Address Register

*Table 513.*0x94 - RTSTBA - GR1553B RT Sub address table base address register

31 9	8 0
SUBADDRESS TABLE BASE (SATB)	RESERVED
0	0
rw	r

31 : 9	Base address, bits 31-9 for subaddress table
8 : 0	Always read ‘0’, writing has no effect

41.9.20 RT Mode Code Control Register

*Table 514.*0x98 - RTMCC - GR1553B RT Mode code control register

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16
RESERVED	RRTB	RRT	ITFB	ITF	ISTB	IST	DBC
0	0	0	0	0	0	0	0
r	rw	rw	rw	rw	rw	rw	rw

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
TBW	TVW	TSB	TS	SDB	SD	SB	S
0	0	1	1	1	1	1	1
rw	rw	rw	rw	rw	rw	rw	rw

	For each mode code: “00” - Illegal, “01” - Legal, “10” - Legal, log enabled, “11” - Legal, log and interrupt
29 : 28	Reset remote terminal broadcast (RRTB)
27 : 26	Reset remote terminal (RRT)
25 : 24	Inhibit & override inhibit terminal flag bit broadcast (ITFB)
23 : 22	Inhibit & override inhibit terminal flag (ITF)
21 : 20	Initiate self test broadcast (ISTB)
19 : 18	Initiate self test (IST)
17 : 16	Dynamic bus control (DBC)
15 : 14	Transmit BIT word (TBW)
13 : 12	Transmit vector word (TVW)
11 : 10	Transmitter shutdown & override transmitter shutdown broadcast (TSB)
9 : 8	Transmitter shutdown & override transmitter shutdown (TS)
7 : 6	Synchronize with data word broadcast (SDB)
5 : 4	Synchronize with data word (SD)
3 : 2	Synchronize broadcast (SB)
1 : 0	Synchronize (S)

41.9.21 RT Time Tag Control Register

Table 515. 0xA4 - RTTTC - GR1553B RT Time tag control register

31 16	15 0
TIME RESOLUTION (TRES)	TIME TAG VALUE (TVAL)
0	0
rw	rw

31 : 16 Time tag resolution (TRES) - Time unit of RT:s time tag counter in microseconds, minus 1

15 : 0 Time tag value (TVAL) - Current value of running time tag counter

41.9.22 RT Event Log Mask Register

Table 516. 0xAC - RTELM - GR1553B RT Event Log mask register

31 21	16 2	1 0
RESERVED	EVENT LOG SIZE MASK	RES
	0xFFFFFFC	
r	rw	r

31 : 0 Mask determining size and alignment of the RT event log ring buffer. All bits “above” the size should be set to ‘1’, all bits below should be set to ‘0’

41.9.23 RT Event Log Position Register

Table 517. 0xB0 - RTELP - GR1553B RT Event Log position register

31 0
EVENT LOG WRITE POINTER
0
rw

31 : 0 Address to first unused/oldest entry of event log buffer, 32-bit aligned

41.9.24 RT Event Log Interrupt Position Register

Table 518. 0xB4 - RTELIP - GR1553B RT Event Log interrupt position register

31 0
EVENT LOG IRQ POINTER
0
r

31 : 0 Address to event log entry corresponding to interrupt, 32-bit aligned
The register is set for the first interrupt and not set again until the interrupt has been acknowledged.

41.9.25 BM Status Register

Table 519. 0xC0 - BMS - GR1553B BM Status register

31	30	29 0
BMSUP	KEYEN	RESERVED
*	*	0
r	r	r

31 BM Supported (BMSUP) - Reads ‘1’ if BM support is in the core.

30 Key Enabled (KEYEN) - Reads ‘1’ if the BM validates the BMKEY field when the control register is written.

41.9.26 BM Control Register

*Table 520.*0xC4 - BMC - GR1553B BM Control register

31 16	15 6	5	4	3	2	1	0
BMKEY	RESERVED	WRSTP	EXST	IMCL	UDWL	MANL	BMEN
0	0	0	0	0	0	0	0
rw	r	rw	rw	rw	rw	rw	rw

31 : 16	Safety key - If extra safety keys are enabled (see KEYEN), this field must be 0x1543 for a write to be accepted. Is 0x0000 when read.
5	Wrap stop (WRSTP) - If set to ‘1’, BMEN will be set to ‘0’ and stop the BM when the BM log position wraps around from buffer end to buffer start
4	External sync start (EXST) - If set to ‘1’,BMEN will be set to ‘1’ and the BM is started when an external BC sync pulse is received
3	Invalid mode code log (IMCL) - Set to ‘1’ to log invalid or reserved mode codes.
2	Unexpected data word logging (UDWL) - Set to ‘1’ to log data words not seeming to be part of any command
1	Manchester/parity error logging (MANL) - Set to ‘1’ to log bit decoding errors
0	BM Enable (BMEN) - Must be set to ‘1’ to enable any BM logging

41.9.27 BMRT Address Filter Register

*Table 521.*0xC8 - BMRTAF - GR1553B BM RT Address filter register

31 0
ADDRESS FILTER MASK
0xFFFFFFFF
rw

31	Enables logging of broadcast transfers
30 : 0	Each bit position set to ‘1’ enables logging of transfers with the corresponding RT address

41.9.28 BMRT Sub address Filter Register

*Table 522.*0xCC - BMRTSF - GR1553B BM RT Sub address filter register

31 0
SUBADDRESS FILTER MASK
0xFFFFFFFF
rw

31	Enables logging of mode commands on sub address 31
30 : 1	Each bit position set to ‘1’ enables logging of transfers with the corresponding RT sub address
0	Enables logging of mode commands on sub address 0

41.9.29 BMRT Mode Code Filter Register

*Table 523.*0xCC - BMRTMC - GR1553B BM RT Mode code filter register

31	19	18	17	16
RESERVED		STSB	STS	TLC
0x1ttt		1	1	1
r		rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSW	RRTB	RRT	ITFB	ITF	ISTB	IST	DBC	TBW	TVW	TSB	TS	SDB	SD	SB	S
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Each bit set to '1' enables logging of a mode code:

- 18 Selected transmitter shutdown broadcast & override selected transmitter shutdown broadcast (STSB)
- 17 Selected transmitter shutdown & override selected transmitter shutdown (STS)
- 16 Transmit last command (TLC)
- 15 Transmit status word (TSW)
- 14 Reset remote terminal broadcast (RRTB)
- 13 Reset remote terminal (RRT)
- 12 Inhibit & override inhibit terminal flag bit broadcast (ITFB)
- 11 Inhibit & override inhibit terminal flag (ITF)
- 10 Initiate self test broadcast (ISTB)
- 9 Initiate self test (IST)
- 8 Dynamic bus control (DBC)
- 7 Transmit BIT word (TBW)
- 6 Transmit vector word (TVW)
- 5 Transmitter shutdown & override transmitter shutdown broadcast (TSB)
- 4 Transmitter shutdown & override transmitter shutdown (TS)
- 3 Synchronize with data word broadcast (SDB)
- 2 Synchronize with data word (SD)
- 1 Synchronize broadcast (SB)
- 0 Synchronize (S)

41.9.30 BMLog Buffer Start

*Table 524.*0xD4 - BMLBS - GR1553B BM Log buffer start

31	0
BM LOG BUFFER START	
0	
rw	

31 : 0 Pointer to the lowest address of the BM log buffer (8-byte aligned)
Due to alignment, bits 2:0 are always 0.

41.9.31 BMLog Buffer End

*Table 525.*0xD8 - BMLBE - GR1553B BM Log buffer end

31 22	21 3	2 0
-	BM LOG BUFFER END	-
	0x0000007	
r	rw	r

31 : 0 Pointer to the highest address of the BM log buffer
Only bits 21:3 are settable, i.e. the buffer can not cross a 4 MB boundary Bits 31:22 read the same as the buffer start address.Due to alignment, bits 2:0 are always equal to 1

41.9.32 BMLog Buffer Position

*Table 526.*0xDC - BMLBP - GR1553B BM Log buffer position

31 22	21 3	2 0
-	BM LOG BUFFER POSITION	-
	0x00000000	
r	rw	r

31 : 0 Pointer to the next position that will be written to in the BM log buffer

Only bits 21:3 are settable, i.e. the buffer can not cross a 4 MB boundary Bits 31:22 read the same as the buffer start address.Due to alignment, bits 2:0 are always equal to 0

41.9.33 BM Time Tag Control Register

*Table 527.*0xE0 - BMTTC - GR1553B BM Time tag control register

31 24	23 0
TIME TAG RESOLUTION	TIME TAG VALUE
0	0
rw	rw

31 : 24 Time tag resolution (TRES) - Time unit of BM:s time tag counter in microseconds, minus 1

23 : 0 Time tag value (TVAL) - Current value of running time tag counter

41.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x04D. For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

41.11 Implementation

41.11.1 Reset

The core does not changs reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core has two separate reset inputs for the two clock domains. They should be reset simultaneously, for instance by using two Reset generator cores connected to the same reset input but clocked by the respective clocks See the documentation of the *syncrst* VHDL generic for possible reset implementations.

41.11.2 External control

An external control synthesis option is available that changes the core so that some of the configuration register fields inside the core driven directly from external signals (auxin.extctrl) instead from the internal logic of the IP core. The fields that are externally controlled can no longer be overwritten through the register interface, however the registers can still be read out through the APB interface and will return the values that were externally assigned. Note also that the external control of rtaddr/rten fields will also override the ordinary auxin.rtaddr/auxin.rtpar functionality.

41.11.3 Endianness

The wrapper supports both big-endian and little-endian systems and will in default configuration change endianness behaviour depending on the incoming endianness signal in the AMBA records. This allows the endianness to be configured through the global GRLIB configuration package, or the *ahbendian* generic into the AHB controller. See GRLIB User's Manual for further details.

There is a legacy generic ahbendian on the core that can be used to select a fixed endian-ness. It is recommended to keep this at the default value which will use the signal as described above.

41.12 Configuration options

Table 528 shows the configuration options of the core (VHDL generics).

*Table 528.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Index of the interrupt line.	0 - NAHBIRQ-1	0
bc_enable	Selects whether BC support is built into the core	0 - 1	1
rt_enable	Selects whether RT support is built into the core	0 - 1	1
bm_enable	Selects whether BM support is built into the core	0 - 1	1
bc_timer	Selects whether the BC timer and wake-up interrupt features are built into the core. 0=None, 1=Timer, 2=Timer and wake-up	0-2	1
bc_rtbusmask	Selects whether the BC per-RT bus swap register is built into the core.	0-1	1
extra_regkeys	Enables extra safety keys for the BM control register and for all fields in the RT control registers	0-1	0
syncrst	Selects reset configuration: 0: Asynchronous reset, all registers in core are reset 1: Synchronous, minimal set of registers are reset 2: Synchronous, most registers reset (increases area slightly to simplify netlist simulation)	0-2	1
ahbendian	Selects AHB bus endianness (for use in non-GRLIB systems), 0=Big endian, 1=Little endian, 2=Select based on signal in AHBMI record (default, recommended)	0 - 2	2
bm_filters	Enable BM filtering capability	0 - 1	1
codecfreq	Codec clock domain frequency in MHz	20 or 24	20
sameclk	AMBA clock and reset is same as codec (removes internal synchronization)	0 - 1	0
codecver	Version selection for internal 1553 codec. 0: Old version, 1: New version, 2: Soft programmable	0 - 2	1
extctrlen	Enables external signal control of register fields from auxin.extctrl record	0 - 1	0
databufen	Enables separate buffer for transfer data.	0 - 1	0

41.13 Signal descriptions

Tables 529-530 shows the interface signals of the core (VHDL ports).

*Table 529.*Signal descriptions on AMBA side

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock, AMBA clock domain	-
RST	N/A	Input	Reset for registers in CLK clock domain	Low
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
APBSI	*	Input	APB slave input signals	-
APBSO	*	Output	APB slave output signals	-
AUXIN	EXTSYNC	Input	External sync input for Bus Controller Re-synchronized to AMBA clk internally. Edge-detection checks for the sampled pattern “01”, i.e. pulses should be at least one CLK cycle to always get detected.	Pos. edge
	RTADDR	Input	Reset value for RT address, if parity matches.	-
	RTPAR	Input	RT address odd parity	-
	EXTCTRL	Input	External control register input record, used only when extctrlen=1. RTEN, RTADDR,BRS,SYS,SYDS:: Controls fields with the same name in the RT Configuration register BUSY: Controls the BUSY field in the RT Bus Status register SATB: Controls the RT Subaddress table base register. MCCR: Controls the RT mode code control register	-
AUXOUT	RTSYNC	Output	Pulsed for one CLK cycle after receiving a synchronize mode command in RT mode	High
	BUSRESET	Output	Pulsed for one CLK cycle after receiving a reset remote terminal mode command in RT mode	High
	VALIDCMDA	Output	Pulsed for one CLK cycle after receiving a valid command word on bus A/B in RT mode	High
	VALIDCMDB	Output		High
	TIMEDOUTA	Output	Asserted when the terminal fail-safe timer has triggered on bus A/B.	High
	TIMEDOUTB	Output		High
	BADREG	Output	Pulsed for one CLK cycle when an invalid register access is performed, either: - an access to an undefined register, - read/write from a write-only/read-only register, - a read/write to a non-implemented part of the core - an incorrect BCKEY/BMKEY	High
	IRQVEC	Output	Auxiliary IRQ vector. Pulsed at the same time as the ordinary PIRQ line, but with a separate line for each interrupt: 7: BM Timer overflow, 6: BM DMA Error, 5: RT Table error, 4: RT DMA Error, 3: RT Event 2: BC Wake-up, 1: BC DMA Error, 0: BC Event	High

* see GRLIB IP Library User’s Manual

*Table 530.*Signal descriptions on 1553 side

Signal name	Field	Type	Function	Active
CODEC_CLK	N/A	Input	Codec clock	-
CODEC_RST	N/A	Input	Reset for registers in CODEC_CLK domain	Low
TXOUT	BUSA_TXP	Output	Bus A transmitter, positive output	High **
	BUSA_TXN	Output	Bus A transmitter, negative output	High **
	BUSA_TXEN	Output	Bus A transmitter enable	High
	BUSA_RXEN	Output	Bus A receiver enable	High
	BUSB_TXP	Output	Bus B transmitter, positive output	High **
	BUSB_TXN	Output	Bus B transmitter, negative output	High **
	BUSB_TXEN	Output	Bus B transmitter enable	High
	BUSB_RXEN	Output	Bus B receiver enable	High
	BUSA_TXIN	Output	Inverted version of BUSA_TXEN (for VHDL coding convenience)	High
	BUSB_TXIN	Output	Inverted version of BUSB_TXEN	High
TXOUT_FB	See TXOUT	Input	Feedback input to the terminal fail-safe timers. Should be tied directly to TXOUT, but are exposed to allow testing the fail-safe timer function. This input is re synchronized to CODEC_CLK so it can be asynchronous.	See TXOUT
RXIN	BUSA_RXP	Input	Bus A receiver, positive input	High **
	BUSA_RXN	Input	Bus A receiver, negative input	High **
	BUSB_RXP	Input	Bus B receiver, positive input	High **
	BUSB_RXN	Input	Bus B receiver, negative input	High **

** The core will put both P/N outputs low when not transmitting. For input, it accepts either both-low or both-high idle.

41.14 Signal definitions and reset values

The signals and their reset values are described in table 531.

*Table 531.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
busa_rxen	Output	Enable for the A receiver	High	Logical 0
busa_rxp	Input	Positive data input from the A receiver	High*	-
busa_rxn	Input	Negative data input from the A receiver	High*	-
busa_txinh	Output	Enable for the A transmitter	Low**	Logical 1
busa_txp	Output	Positive data to the A transmitter	High	Logical 0
busa_txn	Output	Negative data to the A transmitter	High	Logical 0
busb_rxen	Output	Enable for the B receiver	High	Logical 0
busb_rxp	Input	Positive data input from the B receiver	High*	-
busb_rxn	Input	Negative data input from the B receiver	High*	-
busb_txinh	Output	Enable for the B transmitter	Low**	Logical 1
busb_txp	Output	Positive data to the B transmitter	High	Logical 0
busb_txn	Output	Negative data to the B transmitter	High	Logical 0

* rx inputs can be either both-high or both-low when bus is idle

** txinh inhibits (disables) transmission when high, enables transmission when low

41.15 Timing

The timing waveforms and timing parameters are shown in figure 124 and are defined in table 532.

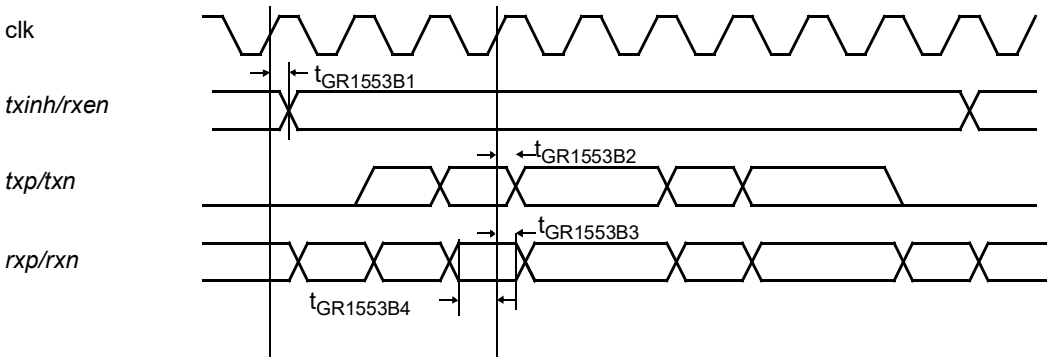


Figure 124. Timing waveforms

*Table 532.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{GR1553B1}$	clock to output delay, control signals	rising *clk* edge	-	TBD	ns
$t_{GR1553B2}$	clock to output delay, transmit data	rising clk edge	TBD	TBD	ns
$t_{GR1553B3}$	data input to clock setup	rising *clk* edge	TBD *	-	ns
$t_{GR1553B4}$	data input from clock hold	rising *clk* edge	TBD *	-	ns

* The rx input signals are re-synchronized to clk internally

41.16 Library dependencies

Table 533 shows libraries used when instantiating the core (VHDL libraries).

*Table 533.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB/APB signal definitions
GAISLER	GR1553B_PKG	Signals, component	signal and component declaration

41.17 Instantiation

This example shows how the core can be instantiated in a GRLIB design.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, gaisler;
use grlib.amba.all;
use gaisler.gr1553b_pkg.all;
use gaisler.misc.rstgen;

entity gr1553b_ex is
  generic (
    padtech     : integer
    );
  port (
    rstn        : in std_ulogic;
    clk         : in std_ulogic;
    codec_clk   : in std_ulogic;
```

```
    -- MIL-STD-1553 signals
    txAen       : out std_ulogic;
    txAP        : out std_ulogic;
    txAN        : out std_ulogic;
    rxAen       : out std_ulogic;
    rxAP        : in std_ulogic;
    rxAN        : in std_ulogic;
    txAen       : out std_ulogic;
    txAP        : out std_ulogic;
    txAN        : out std_ulogic;
    rxAen       : out std_ulogic;
    rxAP        : in std_ulogic;
    rxAN        : in std_ulogic
    );
end;

architecture rtl of gr1553b_ex is

  -- System-wide synchronous reset
  signal rst       : std_logic;

  -- AMBA signals
  signal apbi      : apb_slv_in_type;
  signal apbo      : apb_slv_out_vector := (others => apb_none);
  signal ahbi      : ahb_mst_in_type;
  signal ahbo      : ahb_mst_out_vector := (others => apb_none);

  -- GR1553B signals
  signal codec_rst : std_ulogic;
  signal txout     : gr1553b_txout_type;
  signal rxin      : gr1553b_rxin_type;
  signal auxin     : gr1553b_auxin_type;
  signal auxout    : gr1553b_auxout_type;

begin

  rg0: rstgen port map (rstn, clk, '1', rst, open);

  -- AMBA Components are instantiated here
  ...

  -- Reset generation for 1553 codec
  rgc: rstgen port map (rstn, codec_clk, '1', codec_rst, open);

  -- GR1553B

  gr1553b0: gr1553b
  generic map (hindex => 4, pindex => 7, paddr => 7, pirq => 13, syncrst => 1,
               bc_enable => 1, rt_enable => 1, bm_enable => 1)
  port map (clk, rst, ahbi, ahbo(4), apbi, apbo(7), auxin, auxout,
            codec_clk, codec_rst, txout, txout, rxin);

  p: gr1553b_pads
  generic map (padtech => padtech, outen_pol => 0)
  port map (txout,rxin,
            rxAen,rxAP,rxAN,txAen,txAP,txAN,
            rxBen,rxBP,rxBN,txBen,txBP,txBN);

  auxin         <= gr1553b_auxin_zero;

end;
```

41.18 Constraints

This section contains example constraints for GR1553B.

0. Define a clock called 'mil_clk'

1. milclkperiod = 1553B Maximum clock frequency

2. tech_lib_setup = Setup timing for FlipFlop used in technology
3. tech_lib_hold = Hold timing for FlipFlop used in technology

```
set_input_delay -clock [get_clocks mil_clk] -min [expr $tech_lib_hold] [get_ports
$mil_inputs]
set_input_delay -clock [get_clocks mil_clk] -max [expr $milclkperiod/2 - $tech_lib_setup]
[get_ports $mil_inputs] -add_delay
set_output_delay -clock [get_clocks mil_clk] -max [expr $milclkperiod/2 + $tech_lib_setup]
[get_ports $mil_outputs] -add_delay
set_output_delay -clock [get_clocks mil_clk] -min [expr -1 * $tech_lib_hold]
[get_ports $mil_outputs] -add_delay
```

41.19 Note: AHB Interface Compatibility

41.19.1 Introduction

When using the GR1553B core in a non-GRLIB environment, the AHB system designer must make sure that the slaves to be accessed by the core are compatible with the accesses the master makes.

41.19.2 Generic access patterns

The GR1553B core performs the following accesses on the AHB bus:

- 32-bit sequential read burst, unspecified length, Length 2-3, 128-bit aligned start address. (BC,RT)
- 32-bit sequential write burst, length 2 (BM)
- Single 32-bit read/write (BC,RT)
- Single 16-bit write (BC,RT)
- Idle transfers

The master supports wait states (hready low) as well as split and retry responses. In either case, it will retry accesses indefinitely until getting an OKAY or ERROR response.

Busy cycles and locked transfers are not used by the core.

41.19.3 Endianness

The GR1553B core, in its standard configuration, only works on big-endian systems. Byte-swapping in software is not enough because of the 16-bit writes.

Little-endian bus support can be configured by setting the ahbendian generic to 1. The endian-ness setting only changes the handling of data buffers, data structures are still read using 32-bit reads/ writes and bit fields extracted from the same bit positions.

For data buffers, the core is designed to make 16-bit addressing correct, so that each 16-bit data word in memory is transferred msb to lsb in increasing address order. Note that with little-endian addressing this means that the data will not be sent in byte order. This means that care must be taken to ensure correct ordering when transferring data from 1553 buffers to/from byte streams (files, network packets, etc.).

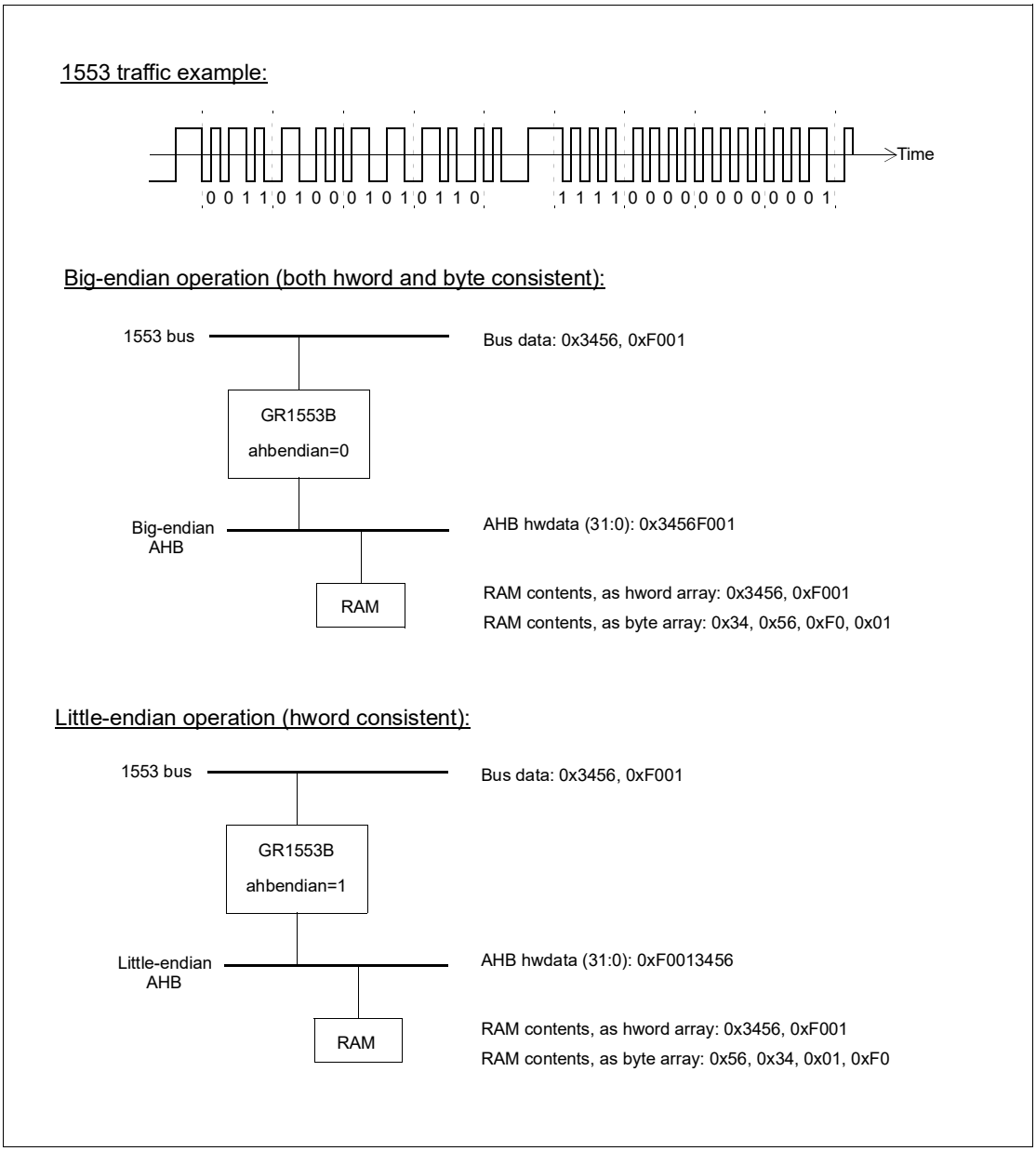


Figure 125. Relation between transferred 1553 data order and RAM contents for little/big-

41.20 Note: AHB Latency and throughput requirements

41.20.1 Introduction

Since the AMBA AHB bus standard does not in itself guarantee any maximum latency or throughput, the AHB bus system must be carefully designed so that the core can satisfy the 1553 requirements reliably.

Throughput is not normally a problem, since even at the lowest supported AMBA frequency of 10 MHz, the core can only use a few percent of the bus bandwidth.

Latency is a larger concern, especially when there are other bus masters with high bandwidth utilization or large burst access patterns.

Some general recommendations:

- Make GR1553B the highest priority master on the bus
- Limit the maximum length of other master's bursts to the order of 5 us.
- Use local RAM for descriptors and preferably also data.

41.20.2 BC Descriptor processing

Between transfers, the BC core first writes the descriptor status word using a 32-bit write, then fetches the following descriptor using a 3 x 32-bit read burst.

If the next location in the schedule is a branch, then the burst fetches only the first two 32-bit words, then processes the condition for one cycle, and then continues.

The core keeps the bus request signal high continuously until the next descriptor has been read. Between the status write and the descriptor read, there is one idle AHB transaction, i.e. the descriptor read is started the cycle after the status write finishes. Between a branch read and the next descriptor read, there are two idle AHB transactions, i.e. one completely idle cycle between the end of branch read and the start of descriptor read.

41.20.3 BC Asynchronous scheduling

If the asynchronous list is started and there is slack in the regular schedule, the BC needs to read the next asynchronous descriptor in order to make the scheduling decision. In case there is less slack than was specified in the asynchronous descriptor, the BC will then read the regular descriptor and proceed with the regular schedule. In this case, the BC remembers the asynchronous time requirement and will not need to re-read the asynchronous descriptor until the asynchronous transfer is actually scheduled.

If there is less than 24 us of slack in the regular schedule, then the BC will assume there is no time for the asynchronous transfer, not read the asynchronous descriptor and just proceed with the regular schedule. This means that as long as two BC descriptors can be read within 24 us, the asynchronous descriptor processing can not affect the ordinary schedule.

41.20.4 BC Data Buffer processing

When transferring data from BC-to-RT, the core reads the first one or two (depending on alignment) 16-bit data words from memory using a 32-bit read, while the command word is being transferred. The core then reads an additional 32-bit word every time the last previously read data word is sent and there is more to send.

For transfers from RT-to-BC, the core either writes the first data word using a 16-bit write, or writes the first two data words using a 32-bit write, depending on data buffer alignment. It will then perform a 32-bit write after every two data words received. While performing the write, the core does not buffer another data word so the write has to complete within 20 us.

If there is one received data word left after the transfer, it is written during the 5 us message gap. If it doesn't finish in that time, the descriptor processing will be delayed until after the write finishes.

41.20.5 BC Requirements

The hard requirement on the BC comes from the data buffers, where one read/write should finish in 20 us. This applies only if the data buffer feature is not enabled. If the data buffer feature is enabled, there is no hard timing requirement but the time taken to write out the data buffer from the previous transfer and read in the entire data buffer will affect the earliest possible time the transfer can begin.

The time needed for descriptor processing will affect the schedule. In order to reduce this time, the time for a 32-bit write followed by a 3x32-bit read burst should be reduced, preferably to the 2-3 us range. This is more of a soft requirement, and for larger schedules it can often be viewed as an average over minor frames or other groups of transfers.

If the asynchronous scheduling feature is used, and you do not want the regular schedule to be affected by this, two 3x32-bit read bursts must be able to finish within 24 us.

41.20.6 RT Transfer Processing

For the RT there are two cases to consider, receive and transmit.

The receive case:

1. The RT gets the receive command

2. While the first data word is transmitted over the bus, the RT looks up the subaddress table and receive descriptor.

3. The data words are stored

4. The RT waits 5-6 us to see that there are no extra (unwanted) data words.

5. The status word is transmitted. In parallel, the results are written to the descriptor and the log and table is updated.

The transmit case:

1. The RT gets the transmit command

2. In parallel there are two processes working:

    Process A:

        A.1 The RT looks up in the subaddress table that the request is legal.

        A.2 The RT looks up the descriptor and reads the first word of data

    Process B:

        B.1 The RT waits 5-6 us to see that there are no extra (unwanted) data words received.

        B.2 Wait for A.1 to finish

        B.3 Send status word

        B.4 Send data

3. The data words are sent

4. The results are written to the descriptor and the log and table is updated.

In the receive case, the subaddress table and descriptor reads (two 3x32-bit read bursts) have to finish within the 20 us period that the data word is transferred.

In the transmit case, the subaddress table read (one 2x32-bit read burst) has to finish in time so that the RT satisfies the response time requirement.

The RT has a status word response time requirement of 12 us mid-bit to mid-sync, which translates into in 10 us maximum bus dead time before responding. These 10 us also include transceiver delays, so the actual time available is typically closer to 9 us.

The core has a safety limit of 9 us, after which a table error interrupt is generated and no response is generated. However, this should be seen as a fallback solution and triggering this time-out should be considered a fault at the AMBA bus design level.

41.20.7 RT Requirements

From the cases above, the following requirements can be derived:

One 2x32-bit read burst must finish in 8.5-9 us (from the transmit case above)

Two 3x32-bit read bursts plus one 32-bit read must finish in 20 us.

Each 32-bit data buffer read/write must finish in 20 us.

41.20.8 Bus Monitor

The bus monitor will at full bus traffic, write 2x32-bit data words every 20 us.

If run in parallel with the BC or RT, this will need to be added to the requirements. The core is designed so that, after an RT+BM receives a command word, the RT will access the bus first to do the more urgent accesses.

41.21 Note: BC transfer timing

41.21.1 Introduction

In order to design a transfer schedule for the Bus Controller, the worst-case times for each transfer must be calculated. This note is intended to give some hints on how to do this. Understanding of the 1553B protocol is assumed, see the AS15531 standard (in particular, Figure 9) for details.

41.21.2 Overview

Except for the case of automatic retries, the longest time a transfer takes is the success case, where all permitted time-outs and slack is used.

When reading the standard, there are a few points to note for the calculations:

- The time-outs and gaps in the standard are specified mid-parity to mid-sync. To convert into bus dead time, one must subtract 2 us.
- The timings are specified on the bus side, so transceiver delays must be taken into account.

41.21.3 Transceiver delay

The timings in the 1553 standard are specified on the terminal boundary which is on the bus side of the transceiver. However, the core operates on the other side of the transceiver. Therefore, the transceiver delays must be taken into account.

The BC mode uses the loopback checking mechanism to compensate for transceiver delay. After the command words have been sent, it waits for the words to loop back through the receiver and then starts the RT time-out timer.

41.21.4 BC Transfer Steps: Parts

A BC transfer can be divided into the following steps:

1. Transmission of control words and receive data

This is a continuous transmission which takes 20 us per word sent. Also include 0.2 us starting delay.

2. Transceiver turnaround

The core waits for the transmitted command to loop back into the receiver. By doing this before starting the RT time-out clock, we get an accurate RT timeout regardless of transceiver delay.

The time this takes is the sum of the transmitter and receiver delay, plus internal decoding delays of 0.15-0.40 us.

3. RT Response Time-out

The core allows a maximum of 12.0-12.5 us bus dead time before the beginning of the RT Response. Note that this time can be increased via a descriptor setting.

4. RT Response and transmit data

The BC receives the RT status word and the specified number of data words.

The Bus Controller checks for message continuity and allows a maximum of 1 us of sync drift over the entire message. The maximum time for this part is thus 20 us/word + 1 us.

5. Second RT Response Time-out

For RT-to-RT transfers. See step 3.

6. Second RT Response

For RT-to-RT transfers. See step 4.

7. Word count verification

For non-broadcast and RT-to-RT messages, the BC waits an additional 5 us to ensure that there is no additional word sent out by the last transmitting RT.

8. Store result, fetch next descriptor

The time this takes to perform depends completely on the AMBA system the core is connected to.

9. Broadcast message gap and descriptor processing

For single-RT broadcasts, instead of step 7-8 the core inserts a 3 us message gap, and in parallel starts fetching the next descriptor. This step therefore takes the maximum of 3 us and the time needed for step 8.

41.21.5 BC transfer steps: Composition

The different 1553 message types map to the above steps as:

BC-to-RT, RT-to-BC, Mode: Steps 1-4,7-8

RT-to-RT: Steps 1-8

BC-to-RT broadcast, Mode broadcast: Steps 1,2,9

RT-to-RT broadcast: Steps 1-4,7-8

If broadcast response checking is enabled in the BC status register, the core waits and checks that no RT produces a response on the bus after each broadcast. Thus, the broadcast case changes to 1-4,7-8, and 30-35 us is added to the worst-case transfer time.

41.21.6 Timing calculation

Based on the steps above, and also taking automatic replies into account, we end up with the following calculations:

BC-to-RT, RT-to-BC or Mode, N data words:

$T = C_{try} \times (20.2 + Nx20 + T_{loop} + 12.5 + T_{extra} + 21 + 5) + T_{dpr}$
$= C_{try} \times (58.7 + Nx20 + T_{extra} + T_{loop}) + T_{dpr}$

RT-to-RT, N data words:

$T = C_{try} \times (40.2 + T_{loop} + 12.5 + T_{extra} + 21 + Nx20 + 12.5 + T_{extra} + 21 + 5) + T_{dpr}$
$= C_{try} \times (112.2 + Nx20 + 2T_{extra} + T_{loop}) + T_{dpr}$

BC-to-RT broadcast or Mode broadcast, N data words:

$T = Ctry \times (20.2 + Nx20 + T_{loop} + 3) - 3 + \max(T_{dpr},3)$

RT-to-RT broadcast:

$T = Ctry \times (40.2 + T_{loop} + 12.5 + T_{extra} + 21 + Nx20 + 5) + T_{dpr}$

Where:

T Worst-case time usage for transfer (i.e. how much the following transfer is delayed when running at maximum rate)

C_{try} The maximum number of attempts, controlled by descriptor, equal to 1 unless automatic retries are used.

T_{loop} Time from the end of a word transmission to receiver decoding the looped-back word. Sum of transceiver transmit and receive delay, plus internal delay of 400 ns.

T_{dpr} Time needed for data processing between this and the next transfer. This includes storing result, processing any branches, and fetching the next descriptor

T_{extra} Extra RT response time, equal to the RTTO field in the transfer descriptor multiplied by 4 us.

41.21.7 Example

Assume for this example that T_{loop} = 1.4 us (transceiver delay of 500 ns + 500 ns) and T_{dpr} = 3 us (90 cycles at 30 MHz).

An RT-to-RT transfer of 5 data words then needs up to 1 x (112.2 + 5x20 + 2x0 + 1.4) + 3 = 216.6 us of time to execute.

41.22 Note: Time synchronization

41.22.1 Introduction

The purpose of time synchronization is to get a common notion of time between the terminals on the bus. This allows the user to relate time stamps from different terminals and coordinate events. If there is an external time base available on one of the terminals it is also interesting to be able to translate the time stamps over into this time base.

This note describes the GR1553B IP core's BC-to-RT time synchronization capabilities and discusses some applications.

41.22.2 Hardware features: BC features

The BC supports sending the 1553 bus standard's two mode commands dedicated to synchronization, synchronize (code 1) and synchronize with data word (code 17). The commands can be sent either to a specific RT, or sent on the broadcast address to all RT:s on one bus.

With the "wait for external trigger" (WTRIG) descriptor bit, any data transfer or mode code in the schedule can be set up to wait for a positive edge on the IP core's auxin.extsync input signal before starting. The external sync can also be triggered from software by writing to the BC Action Register.

It is also possible to use the regular scheduling features of the core to plan synchronization commands within a frame with high precision, just like any other transfer can be planned. This makes it possible to send timed sync pulses at a different interval than the external sync pulses, or to use the internal BC timer as time master with no external time base at all.

41.22.3 Hardware features: RT features

The RT supports receiving the two synchronization mode commands that can be emitted by the BC. In case of synchronization with data word, the attached data word can be read out through the RT sync register.

The RT has an internal timer with configurable scaler, which can be read-out and configured through the RT Time tag control register. When a mode command is received, the timer value is stored into the RT Sync Register which can be read out by software.

There are also a number of core output signals related to synchronization. The validcmdA/B outputs are raised whenever a valid command word is received in the RT, and this happens at the same time as the internal time stamp is taken. There is also an output called rtsync which is set high only after a successfully received sync command. The rtsync output pulse always occurs after the validcmd pulse.

If software handling of synchronization is desired, the core can be configured to generate an IRQ after a sync mode command has been received.

41.22.4 Hardware features: Internal timers

Both the internal timer used for BC scheduling and the timer used for the RT time stamps are based on 1 MHz clock ticks generated in the codec clock domain that are resynchronized to the AMBA clock domain. When the RT timer register is written, the tick generator and the (optional) scaler gets reset, therefore for best timestamp accuracy, the RT timer should be left free-running and used as differential measurements.

In terms of frequency accuracy, the internal timers will have the same characteristics as the codec clock, which is limited by the 1553 standard to 1000 ppm long-term. If the user has a more accurate clock source than this, this will naturally translate into more accurate timing.

41.22.5 Synchronization schemes

41.22.6 Synchronization schemes: Overview

This section describes a few ways to generate and receive synchronization commands on the bus. There are many possible ways to do this, but in this section the basic scheme assumed is by broadcasting sync mode commands from the BC at a regular interval. To handle fault conditions, the sync commands are sent on alternating buses.

Typically the user also wants to send a frame number or a coarse time stamp with the sync command. This can be sent either as the attached data word when using the sync with data mode command, or it can be sent beforehand to a dedicated subaddress on the RT:s. This is not described further in the section.

41.22.7 Synchronization schemes: BC without external time base

If no external real time base is available on the BC, regular sync commands can still be generated by scheduling as shown in the example below.

Table 534. Example BC descriptor structure with synchronization

	Sum of slot times assigned in each frame = sync period/2				
Frame 1	Sync broadcast Bus A	Transfer	Transfer	...	Jump to frame 2
Frame 2	Sync broadcast Bus B	Transfer	Transfer	...	Jump to frame 3
Frame 3	Sync broadcast Bus A	Transfer	Transfer	...	Jump to frame 4
Frame 4	Sync broadcast Bus B	Transfer	Transfer	...	Jump to frame 1

41.22.8 Synchronization schemes: BC with external time base

If there is an external time base connected to the extsync input of the core, this can be used by setting the wait for external trigger descriptor bit at intervals in the schedule corresponding to the sync interval. The schedule can otherwise be kept as is. As the figure shows it is possible to have the sync pulses at a multiple of the sync interval and use the internal time base for the sync commands in between.

Table 535. Example BC descriptor structure with external synchronization

	Sum of slot times assigned in each frame = sync period/2					
Frame 1	Sync broadcast Bus A, wtrig set	Transfer	Transfer	...	Jump to frame 2	Sum of slot times in all four frames gives external sync period
Frame 2	Sync broadcast Bus B	Transfer	Transfer	...	Jump to frame 3	
Frame 3	Sync broadcast Bus A	Transfer	Transfer	...	Jump to frame 4	
Frame 4	Sync broadcast Bus B	Transfer	Transfer	...	Jump to frame 1	

41.22.9 Synchronization schemes: RT without external time base

If the resolution of the sync period from the BC is good enough, then the simplest solution is to just take the frame number sent by the BC directly and use it as the time stamp.

If more resolution is needed, the user can read out the current value of the RT Timer, compare with the last sync time stamp in the RT Sync register, and use the difference as a time offset from the last sync time from the BC.

41.22.10Synchronization schemes: RT with external time base(s)

If an external timer is available, its value can be read out at the same time as the sync time stamp by the hardware using a construct like the one shown below. The first register captures the external timer value whenever a command word is received, and the second register records the time stamp if the command was a sync mode code. With this value, the user will then obtain a common time stamp between the BC time, RT time and external time that can be used in different ways.

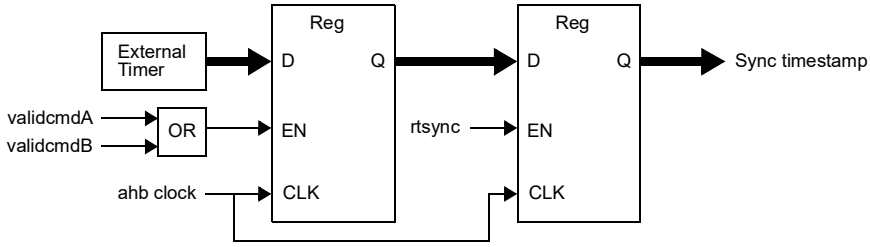


Figure 126. Obtaining an RT sync timestamp in hardware with an external time base

41.22.11Accuracy

41.22.12Accuracy: Propagation delays

When the sync command is sent, there is first an approximate 500 ns of delay inside the core, followed by analog delays, consisting of BC pad delays, BC transmitter delay, bus propagation delays, RT receiver delays, and RT pad delays.

When using the external BC sync, there is also a 2-3 AHB cycle delay before the sync is detected due to synchronization. This creates a small offset between the external time reference and the internal time.

On the RT digital side, there will first be a delay of 20000 ns to receive the whole command word, followed by internal decoding delays of approximately 500 ns before the command word is recognized and the time stamp is taken.

Most of this delay is constant, in particular the 20 us word length, and can therefore be easily compensated for by offsetting the time stamps. Offsetting the RT timestamps by 24 us should be a reasonable first-order approach, accurate within +/- 4 us or so. If more accuracy is needed, characterization measurements or further analysis of the bus system would need to be done.

41.22.13Accuracy: Clock drift

Relative drift between the clocks determine how often synchronization is necessary. Since the internal timers of the BC and RT are as accurate as the codec clocks, and both the clocks must have the 1000 ppm worst-case accuracy permitted by the 1553 standard, the BC and RT timers can drift apart up to 2 us for a sync period of 1 ms. Assuming the external time base is much more accurate, the internal and external BC time bases will drift apart of up to 1 us for every ms of external sync period.

42 GRTIMER - General Purpose Timer Unit

42.1 Overview

The GRTIMER IP core's functionality for latching timer values, external clocking and reload on external events has been merged into the GPTIMER core. All new designs should instantiate the GPTIMER IP core.
A GRTIMER entity exists that is a wrapper around the GPTIMER IP core for backward compatibility.

43 GRACECTRL - AMBA System ACE Interface Controller

43.1 Overview

The core provides an AMBA AHB interface to the microprocessor interface of a Xilinx System ACE Compact Flash Solution. Accesses to the core's memory space are directly translated to accesses on the System ACE microprocessor interface (MPU).

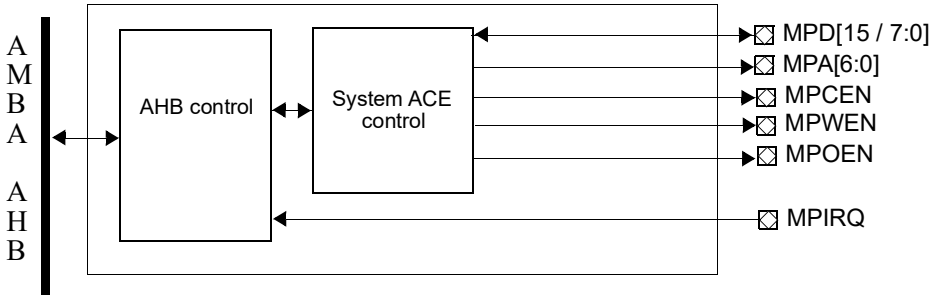


Figure 127. Block diagram

43.2 Operation

43.2.1 Operational model

The core has one AHB I/O area, accesses to this area are directly translated to accesses on the Xilinx System ACE's Microprocessor Interface (MPU). When an access is made to the I/O area, the core first checks if there already is an ongoing access on the MPU. If an access is currently active, the core will respond with an AMBA SPLIT response. If the MPU bus is available, the core will start an access on the MPU bus and issue a SPLIT response to the AMBA master. If the core has been configured for a system that does not support SPLIT responses, it will insert wait states instead.

43.2.2 Bus widths

The AMBA access is directly translated to an MPU access where bits 6:0 of the AMBA address bus are connected to the MPU address bus. The core can be configured to connect to a 16-bit MPU interface or a 8-bit MPU interface. When the core is connected to a 8-bit MPU interface it can emulate 16-bit mode by translating 16-bit (half-word) AMBA accesses into two 8-bit MPU accesses. The mode to use is decided at implementation time via the VHDL generic *mode*.

The core does not perform any checks on the size of the AMBA access and software should only make half-word (16-bit), or byte (8-bit) depending on the setting of VHDL generic *mode*, accesses to the core's memory area. Any other access size will be accepted by the core but the operation may not have the desired result. On AMBA writes the core uses address bit 1 (or address bits 1:0 for 8-bit mode) to select if it should propagate the high or the low part of the AMBA data bus to the MPU data bus. On read operations the core will propagate the read MPU data to all parts of the AMBA data bus.

It is recommended to set the *mode* VHDL generic to 2 for 8-bit MPU interfaces, and to 0 for 16-bit MPU interfaces. This way software can always assume that it communicates via a 16-bit MPU interface (accesses to the System ACE BUSMODEREG register are overriden by the core with suitable values when *mode* is set to 2).

43.2.3 Clocking and synchronization

The core has two clock inputs; the AMBA clock and the System ACE clock. The AMBA clock drives the AHB slave interface and the System ACE clock drives the System ACE interface state machine.

All signals crossing between the two clock domains are synchronized to prevent meta-stability. The system clock should have a higher frequency than the System ACE clock.

43.2.4 Endianness

The core is designed for big-endian systems.

43.3 Registers

The core does implement any registers accessible via AMBA.

43.4 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x067. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

43.5 Implementation

43.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

43.5.2 Technology mapping

The core does not instantiate any technology specific primitives.

43.5.3 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual).

43.5.4 RAM usage

The core does not use any RAM components.

43.6 Configuration options

Table 536 shows the configuration options of the core (VHDL generics).

*Table 536.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB slave index	0 - (NAHBSLV-1)	0
hirq	Interrupt line	0 - (NAHBIRQ-1)	0
haddr	ADDR field of the AHB BAR0	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0	0 - 16#FFF#	16#FFF#
split	If this generic is set to 1 the core will issue AMBA SPLIT responses when it is busy performing an access to the System ACE. Otherwise the core will insert wait states until the operation completes. Note that SPLIT support on the AHBCTRL core MUST be enabled if this generic is set to 1.	0 - 1	0

*Table 536.*Configuration options

Generic name	Function	Allowed range	Default
swap	If this generic is set to 0 the core will connect the System ACE data(15:0) to AMBA data(15:0). If this generic is set to 1, the core will swap the System ACE data line and connect: System ACE data(15:8) <-> AMBA data(7:0) System ACE data(7 :0) <-> AMBA data(15:8). This generic only has effect for *mode* = 0.	0 - 1	0
oepol	Polarity of pad output enable signal	0 - 1	0
mode	Bus width mode 0: Core is connected to 16-bit MPU. Only half-word AMBA accesses should be made to the core. 1: Core is connected to 8-bit MPU. Only byte AMBA accesses should be made to the core. 2: Core is connected to 8-bit MPU but will emulate a 16-bit MPU interface. Only half-word AMBA accesses should be made to the core (recommended setting for 8-bit MPU interfaces).	0 - 2	0

43.7 Signal descriptions

Table 537 shows the interface signals of the core (VHDL ports).

*Table 537.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
CLKACE	N/A	Input	System ACE clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
ACEI	DI(15:0)	Input	Data line	-
	IRQ	Input	System ACE interrupt request	High
ACEO	ADDR(6:0)	Output	System ACE address	-
	DO(15:0)	Output	Data line	-
	CEN	Output	System ACE chip enable	Low
	WEN	Output	System ACE write enable	Low
	OEN	Output	System ACE output enable	Low
	DOEN	Output	Data line output enable	-

* see GRLIB IP Library User's Manual

43.8 Signal definitions and reset values

The signals and their reset values are described in table 538.

*Table 538.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
d[15:0]	InputOutput	System ACE data line	-	-
irq	Input	System ACE interrupt request	Logical 1	-
addr[6:0]	Output	System ACE address	-	-
cen	Output	System ACE chip enable	Logical 0	-
wen	Output	System ACE write enable	Logical 0	-
oen	Output	System ACE output enable	Logical 0	-

43.9 Library dependencies

Table 539 shows the libraries used when instantiating the core (VHDL libraries).

*Table 539.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	MISC	Component, signals	Component and signal definitions
GRLIB	AMBA	Signals	AMBA signal definitions

43.10 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.misc.all;

entity gracectrl_ex is
  port (
    clk        : in  std_ulogic;
    clkace     : in  std_ulogic;
    rstn       : in  std_ulogic;
    sace_a     : out std_logic_vector(6 downto 0);
    sace_mpce  : out std_ulogic;
    sace_d     : inout std_logic_vector(15 downto 0);
    sace_oen   : out std_ulogic;
    sace_wen   : out std_ulogic;
    sace_mpirq : in  std_ulogic;
    );
end;

architecture rtl of gracectrl_ex is
  -- AMBA signals
  signal ahbsi  : ahb_slv_in_type;
  signal ahbso  : ahb_slv_out_vector := (others => ahbs_none);
  ...
  -- GRACECTRL signals
  signal acei : gracectrl_in_type;
  signal aceo : gracectrl_out_type;

begin
```

```
  -- AMBA Components are instantiated here
  ...

  -- GRACECTRL core is instantiated below
  grace0 : gracectrl generic map (hindex => 4, hirq => 4, haddr => 16#002#,
                                  hmask => 16#fff#, split => 1)
    port map (rstn, clk, ahbsi, ahbso(4), acei, acoo);
  sace_a_pads : outpadv generic map (width => 7, tech => padtech)
    port map (sace_a, aceo.addr);
  sace_mpce_pad : outpad generic map(tech => padtech)
    port map (sace_mpce, aceo.cen);
  sace_d_pads : iopadv generic map (tech => padtech, width => 16)
    port map (sace_d, aceo.do, aceo.doen, aceo.di);
  sace_oen_pad : outpad generic map (tech => padtech)
    port map (sace_oen, aceo.oen);
  sace_wen_pad : outpad generic map (tech => padtech)
    port map (sace_wen, aceo.wen);
  sace_mpirq_pad : inpad generic map (tech => padtech)
    port map (sace_mpirq, acei.irq);

end;
```

44 GRADCDAC - ADC / DAC Interface

44.1 Overview

The block diagram shows a possible partitioning of the combined analogue-to-digital converter (ADC) and digital-to-analogue converter (DAC) interface.

The combined analogue-to-digital converter (ADC) and digital-to-analogue converter (DAC) interface is assumed to operate in an AMBA bus system where an APB bus is present. The AMBA APB bus is used for data access, control and status handling.

The ADC/DAC interface provides a combined signal interface to parallel ADC and DAC devices. The two interfaces are merged both at the pin/pad level as well as at the interface towards the AMBA bus. The interface supports simultaneously one ADC device and one DAC device in parallel.

Address and data signals unused by the ADC and the DAC can be used for general purpose input output, providing 0, 8, 16 or 24 channels.

The ADC interface supports 8 and 16 bit data widths. It provides chip select, read/convert and ready signals. The timing is programmable. It also provides an 8-bit address output. The ADC conversion can be initiated either via the AMBA interface or by internal or external triggers. An interrupt is generated when a conversion is completed.

The DAC interface supports 8 and 16 bit data widths. It provides a write strobe signal. The timing is programmable. It also provides an 8-bit address output. The DAC conversion is initiated via the AMBA interface. An interrupt is generated when a conversion is completed.

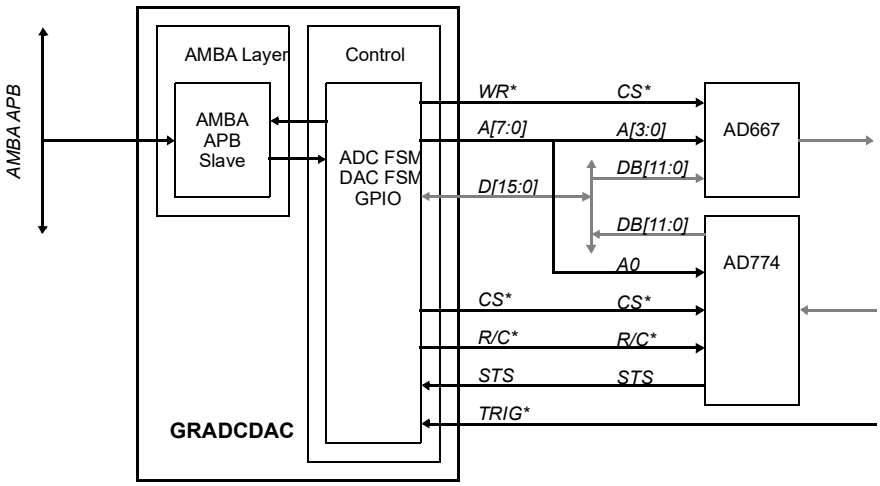


Figure 128. Block diagram and usage example

44.1.1 Function

The core implements the following functions:

- ADC interface conversion:
- ready feed-back, or
- timed open-loop
- DAC interface conversion:
- timed open-loop
- General purpose input output:
- unused data bus, and

- unused address bus
- Status and monitoring:
- on-going conversion
- completed conversion
- timed-out conversion

Note that it is not possible to perform ADC and DAC conversions simultaneously. On only one conversion can be performed at a time.

44.1.2 Interfaces

The core provides the following external and internal interfaces:

- Combined ADC/DAC interface
- programmable timing
- programmable signal polarity
- programmable conversion modes
- AMBA APB slave interface

The ADC interface is intended for amongst others the following devices:

Name:Width:Type:

AD574 12-bit R/C*, CE, CS*, RDY*, tri-state

AD674 12-bit R/C*, CE, CS*, RDY*, tri-state

AD774 12-bit R/C*, CE, CS*, RDY*, tri-state

AD670 8-bit R/W*, CE*, CS*, RDY, tri-state

AD571 10-bit Blank/Convert*, RDY*, tri-state

AD1671 12-bit Encode, RDY*, non-tri-state

LTC141414-bitConvert*, RDY, non-tri-state

The DAC interface is intended for amongst others the following devices:

Name:Width:Type:

AD56110-bitParallel-Data-in-Analogue-out

AD56512-bitParallel-Data-in-Analogue-out

AD66712-bitParallel-Data-in-Analogue-out, CS*

AD76712-bitParallel-Data-in-Analogue-out, CS*

DAC08 8-bit Parallel-Data-in-Analogue-out

44.2 Operation

44.2.1 Interfaces

The internal interface on the on-chip bus towards the core is a common AMBA APB slave for data access, configuration and status monitoring, used by both the ADC interface and the DAC interface.

The ADC address output and the DAC address output signals are shared on the external interface. The address signals are possible to use as general purpose input output channels. This is only realized when the address signals are not used by either ADC or DAC.

The ADC data input and the DAC data output signals are shared on the external interface. The data input and output signals are possible to use as general purpose input output channels. This is only realized when the data signals are not used by either ADC or DAC.

Each general purpose input output channel shall be individually programmed as input or output. This applies to both the address bus and the data bus. The default reset configuration for each general purpose input output channel is as input. The default reset value each general purpose input output channel is logical zero.

Note that protection toward spurious pulse commands during power up shall be mitigated as far as possible by means of I/O cell selection from the target technology.

44.2.2 Analogue to digital conversion

The ADC interface supports 8 and 16 bit wide input data.

The ADC interface provides an 8-bit address output, shared with the DAC interface. Note that the address timing is independent of the acquisition timing.

The ADC interface shall provide the following control signals:

- Chip Select
- Read/Convert
- Ready

The timing of the control signals is made up of two phases:

- Start Conversion
- Read Result

The Start Conversion phase is initiated by one of the following sources, provided that no other conversion is ongoing:

- Event on one of three separate trigger inputs
- Write access to the AMBA APB slave interface

Note that the trigger inputs can be connected to internal or external sources to the ASIC incorporating the core. Note that any of the trigger inputs can be connected to a timer to facilitate cyclic acquisition. The selection of the trigger source is programmable. The trigger inputs is programmable in terms of: Rising edge or Falling edge. Triggering events are ignored if ADC or DAC conversion is in progress.

The transition between the two phases is controlled by the Ready signal. The Ready input signal is programmable in terms of: Rising edge or Falling edge. The Ready input signaling is protected by means of a programmable time-out period, to assure that every conversion terminates. It is also possible to perform an ADC conversion without the use of the Ready signal, by means of a programmable conversion time duration. This can be seen as an open-loop conversion.

The Chip Select output signal is programmable in terms of:

- Polarity
- Number of assertions during a conversion, either
- Pulsed once during Start Conversion phase only,
- Pulsed once during Start Conversion phase and once during Read Result phase, or
- Asserted at the beginning of the Start Conversion phase and de-asserted at the end of the Read Result phase

The duration of the asserted period is programmable, in terms of system clock periods, for the Chip Select signal when pulsed in either of two phases.

The Read/Convert signal is de-asserted during the Start Conversion phase, and asserted during the Read Result phase. The Read/Convert output signal is programmable in terms of: Polarity. The setup timing from Read/Convert signal being asserted till the Chip Select signal is asserted is programmable, in terms of system clock periods. Note that the programming of Chip Select and Read/Convert timing is implemented as a common parameter.

At the end of the Read Result phase, an interrupt is generated, indicating that data is ready for readout via the AMBA APB slave interface. The status of an on-going conversion is possible to read out via the AMBA APB slave interface. The result of a conversion is read out via the AMBA APB slave interface. Note that this is independent of what trigger started the conversion.

An ADC conversion is non-interruptible. It is possible to perform at least 1000 conversions per second.

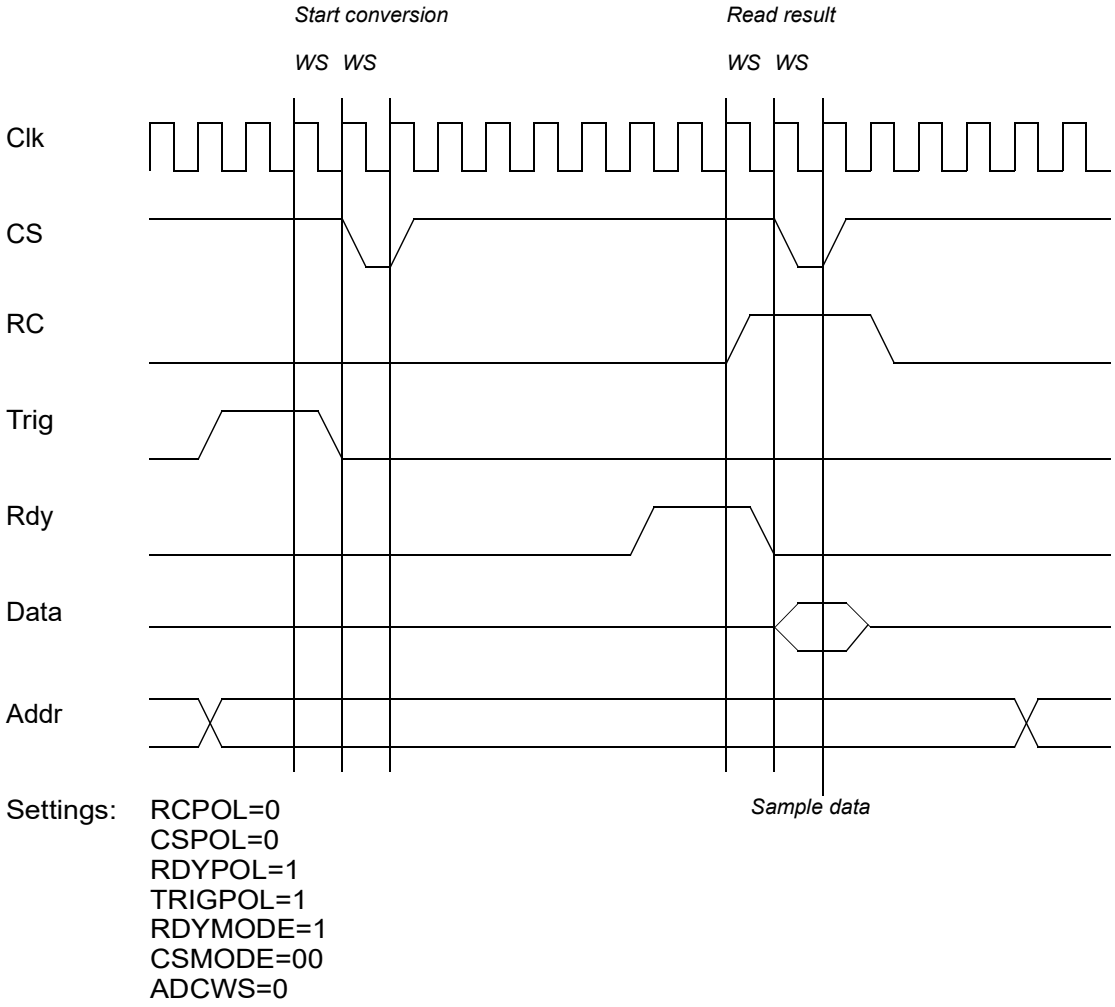


Figure 129. Analogue to digital conversion waveform, 0 wait states (WS)

44.2.3 Digital to analogue conversion

The DAC interface supports 8 and 16 bit wide output data. The data output signal is driven during the conversion and is placed in high impedance state after the conversion.

The DAC interface provides an 8-bit address output, shared with the ADC interface. Note that the address timing is independent of the acquisition timing.

The DAC interface provides the following control signal: Write Strobe. Note that the Write Strobe signal can also be used as a chip select signal. The Write Strobe output signal is programmable in terms of: Polarity. The Write Strobe signal is asserted during the conversion. The duration of the asserted period of the Write Strobe is programmable in terms of system clock periods.

At the end the conversion, an interrupt is generated. The status of an on-going conversion is possible to read out via the AMBA APB slave interface. A DAC conversion is non-interruptible.

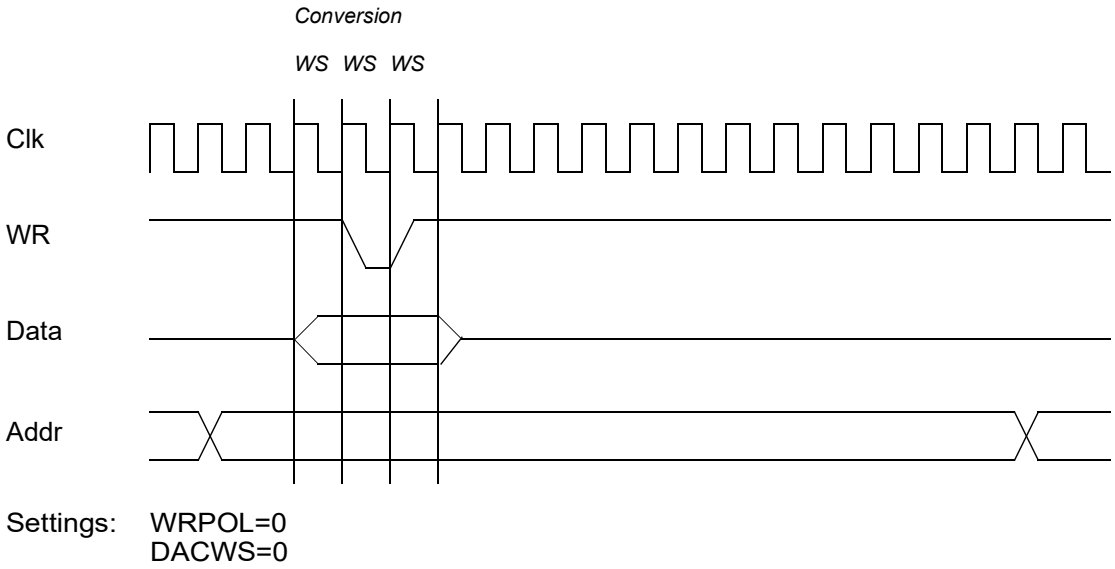


Figure 130. Digital to analogue conversion waveform, 0 wait states (WS)

44.3 Operation

44.3.1 Interrupt

Two interrupts are implemented by the ADC/DAC interface:

Index:Name:Description:

0 ADC ADC conversion ready

1 DAC DAC conversion ready

The interrupts are configured by means of the *pirq* VHDL generic.

44.3.2 Reset

After a reset the values of the output signals are as follows:

Signal:Value after reset:

ADO.Aout[7:0]de-asserted

ADO.Aen[7:0]de-asserted

ADO.Dout[15:0]de-asserted

ADO.Den[15:0]de-asserted

ADO.WRde-asserted (logical one)

ADO.CSde-asserted (logical one)

ADO.RCde-asserted (logical one)

44.3.3 Asynchronous interfaces

The following input signals are synchronized to Clk:

- ADI.Ain[7:0]
- ADI.Din[15:0]
- ADI.RDY
- ADI.TRIG[2:0]

44.4 Registers

The core is programmed through registers mapped into APB address space.

*Table 540.*GRADCDAC registers

APB address offset	Register
0x000	Configuration Register
0x004	Status Register
0x010	ADC Data Input Register
0x014	DAC Data Output Register
0x020	Address Input Register
0x024	Address Output Register
0x028	Address Direction Register
0x030	Data Input Register
0x034	Data Output Register
0x038	Data Direction Register

44.4.1 Configuration Register [ADCONF]

*Table 541.*Configuration register

31	24	23	19	18	17	16
		DACWS		WR POL	DACDW	
		0		0	0	
		rw		rw	rw	

15	11	10	9	8	7	6	5	4	3	2	1	0
ADCWS		RCP OL	CSMODE		CSP OL	RD YM OD E	RD YP OL	TRI GP OL	TRIG-MODE		ADCDW	
0		0	0		0	0	0	0	0		0	
rw		rw	rw		rw	rw	rw	rw	rw		rw	

23-19: DACWS Number of DAC wait states, 0 to 31 [5 bits]
18: WRPOL Polarity of DAC write strobe:
0b = active low
1b = active high
17-16: DACDW DAC data width
00b = none
01b = 8 bit ADO.Dout[7:0]
10b = 16 bit ADO.Dout[15:0]
11b = none/spare
15-11: ADCWS Number of ADC wait states, 0 to 31 [5 bits]

10:	RCPOL	Polarity of ADC read convert: 0b = active low read 1b = active high read
9-8:	CSMODE	Mode of ADC chip select: 00b = asserted during conversion and read phases 01b = asserted during conversion phase 10b = asserted during read phase 11b = asserted continuously during both phases
7:	CSPOL	Polarity of ADC chip select:0b = active low 1b = active high
6:	RDYMODE	Mode of ADC ready: 0b = unused, i.e. open-loop 1b = used, with time-out
5:	RDYPOL	Polarity of ADC ready: 0b = falling edge 1b = rising edge
4:	TRIGPOL	Polarity of ADC triggers: 0b = falling edge 1b = rising edge
3-2:	TRIGMODE	ADC trigger source: 00b = none 01b = ADI.TRIG[0] 10b = ADI.TRIG[1] 11b = ADI.TRIG[2]
1-0:	ADCDW	ADC data width: 00b = none 01b = 8 bit ADI.Din[7:0] 10b = 16 bit ADI.Din[15:0] 11b = none/spare

The ADCDW field defines what part of ADI.Din[15:0] is read by the ADC.

The DACDW field defines what part of ADO.Dout[15:0] is written by the DAC.

Parts of the data input/output signals used neither by ADC nor by DAC are available for the general purpose input output functionality.

Note that an ADC conversion can be initiated by means of a write access via the AMBA APB slave interface, thus not requiring an explicit ADC trigger source setting.

The ADCONF.ADCWS field defines the number of system clock periods, ranging from 1 to 32, for the following timing relationships between the ADC control signals:

- ADO.RC stable before ADO.CS period
- ADO.CS asserted period, when pulsed
- ADO.TRIG[2:0] event until ADO.CS asserted period
- Time-out period for ADO.RDY: 2048 * (1+ADCONF.ADCWS)
- Open-loop conversion timing: 512 * (1+ADCONF.ADCWS)

The ADCONF.DACWS field defines the number of system clock periods, ranging from 1 to 32, for the following timing relationships between the DAC control signals:

- ADO.Dout[15:0] stable before ADO.WR period
- ADO.WR asserted period
- ADO.Dout[15:0] stable after ADO.WR period

44.4.2 Status Register [ADSTAT]

*Table 542.*Status register

31	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED									DACNO	DACRDY	DACON	ADCTO	ADCNO	ADCRDY	ADCON
									0	0	0	0	0	0	0
									r*	r*	r	r*	r*	r*	r

6: DACNO DAC conversion request rejected (due to ongoing DAC or ADC conversion). Cleared on rest.
5: DACRDY DAC conversion completed. Cleared on rest.
4: DACON DAC conversion ongoing
3: ADCTO ADC conversion timeout. Cleared on rest.
2: ADCNO ADC conversion request rejected (due to ongoing ADC or DAC conversion). Cleared on rest.
1: ADCRDY ADC conversion completed. Cleared on rest.
0: ADCON ADC conversion ongoing

Note that the status bits can be used for monitoring the progress of a conversion or to ascertain that the interface is free for usage.

44.4.3 ADC Data Input Register [ADIN]

*Table 543.*ADC Data Input Register

31	16	15	0
		ADCIN	
		0	
		rw*	

15-0: ADCIN ADC input data A write access to the register initiates an analogue to digital conversion, provided that no other ADC or DAC conversion is ongoing (otherwise the request is rejected). A read access that occurs before an ADC conversion has been completed returns the result from a previous conversion. *ADI.Din[15:0]*

Note that any data can be written and that it cannot be read back, since not relevant to the initiation of the conversion.
Note that only the part of ADI.Din[15:0] that is specified by means of bit ADCONF.ADCDW is used by the ADC. The rest of the bits are read as zeros.

Note that only bits dwidth-1 to 0 are implemented.

44.4.4 DAC Data Output Register [ADOUT]

*Table 544.*DAC Data Output Register

31	16	15	0
		DACOUT	
		0	
		rw*	

15-0: DACOUT DAC output data A write access to the register initiates a digital to analogue conversion, provided that no other DAC or ADC conversion is ongoing (otherwise the request is rejected). *ADO.Dout[15:0]*

Note that only the part of ADO.Dout[15:0] that is specified by means of ADCONF.DACDW is driven by the DAC. The rest of the bits are not driven by the DAC during a conversion.

Note that only the part of ADO.Dout[15:0] which is specified by means of ADCONF.DACDW can be read back, whilst the rest of the bits are read as zeros.

Note that only bits dwidth-1 to 0 are implemented.

44.4.5 Address Input Register [ADAIN]

Table 545. Address Input Register

31 8	7 0
	AIN
	0
	r

7-0: AIN Input address *ADI.Ain[7:0]*

Note that only bits awidth-1 to 0 are implemented.

44.4.6 Address Output Register [ADAOUT]

Table 546. Address Output Register

31 8	7 0
	AOUT
	0
	rw

7-0: AOUT Output address *ADO.Aout[7:0]*

Note that only bits awidth-1 to 0 are implemented.

44.4.7 Address Direction Register [ADADIR]

Table 547. Address Direction Register

31 8	7 0
	ADIR
	0
	rw

7-0: ADIR Direction: *ADO.Aout[7:0]*
0b = input = high impedance,
1b = output = driven

Note that only bits awidth-1 to 0 are implemented.

44.4.8 Data Input Register [ADDIN]

*Table 548.*Data Input Register

31 16	15 0
	DIN
	0
	r

15-0: DIN Input data *ADI.Din[15:0]*

Note that only the part of ADI.Din[15:0] not used by the ADC can be used as general purpose input output, see ADCONF.ADCDW.

Note that only bits dwidth-1 to 0 are implemented.

44.4.9 1Data Output Register [ADDOUT]

*Table 549.*Data Output Register

31 16	15 0
	DOUT
	0
	rw

15-0: DOUT Output data *ADO.Dout[15:0]*

Note that only the part of ADO.Dout[15:0] neither used by the DAC nor the ADC can be used as general purpose input output, see ADCONF.DACDW and ADCONF. ADCDW.

Note that only bits dwidth-1 to 0 are implemented.

44.4.10 Data Register [ADDDIR]

*Table 550.*Data Direction Register

31 16	15 0
	DDIR
	0
	rw

15-0: DDIR Direction: *ADO.Dout[15:0]*
0b = input = high impedance,
1b = output = driven

Note that only the part of ADO.Dout[15:0] not used by the DAC can be used as general purpose input output, see ADCONF.DACDW.

Note that only bits dwidth-1 to 0 are implemented.

44.5 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x036. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

44.6 Implementation

44.6.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

44.7 Configuration options

Table 551 shows the configuration options of the core (VHDL generics).

*Table 551.*Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRADCDAC.	0 - NAHBIRQ-1	1
nchannel	Number of input/outputs	1 - 32	24
npulse	Number of pulses	1 - 32	8
invertpulse	Invert pulse output when set	1 - 32	0
cntrwidth	Pulse counter width	4 to 32	20
oepol	Output enable polarity	0, 1	1

44.8 Signal descriptions

Table 552 shows the interface signals of the core (VHDL ports).

*Table 552.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
ADI	ADI.Ain[7:0]	Input	Common Address input	-
	ADI.Din[15:0]		ADC Data input	
	ADI.RDY		ADC Ready input	
	ADI.TRIG[2:0]		ADC Trigger inputs	
ADO	ADO.Aout[7:0]	Output	Common Address output	-
	ADO.Aen[7:0]		Common Address output enable	
	ADO.Dout[15:0]		DAC Data output	-
	ADO.Den[15:0]		DAC Data output enable	
	ADO.WRDAC		Write Strobe	
	ADO.CSADC		Chip Select	
	ADO.RCADC		Read/Convert	

* see GRLIB IP Library User's Manual

Note that the VHDL generic oepol is used for configuring the logical level of ADO.Den and ADO.Aen while asserted.

44.9 Signal definitions and reset values

The signals and their reset values are described in table 553.

*Table 553.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
a[]	Input/Output	Address	High	Tri-state
d[]	Input/Output	Data	High	Tri-state
wr	Output	DAC Write Strobe	-	Logical 0
cs	Output	ADC Chip Select	-	Logical 0
rc	Output	ADC Read/Convert	-	Logical 0
rdy	Input	ADC Ready	-	-
trig[]	Input	ADC Trigger	-	-

44.10 Timing

The timing waveforms and timing parameters are shown in figure 131 and are defined in table 554. Note that the input and output polarities of control and response signals are programmable. The figures shows operation where there are zero wait states. Note also that the address timing has no direct correlation with the ADC and DAC accesses, since controlled by a separate set of registers.

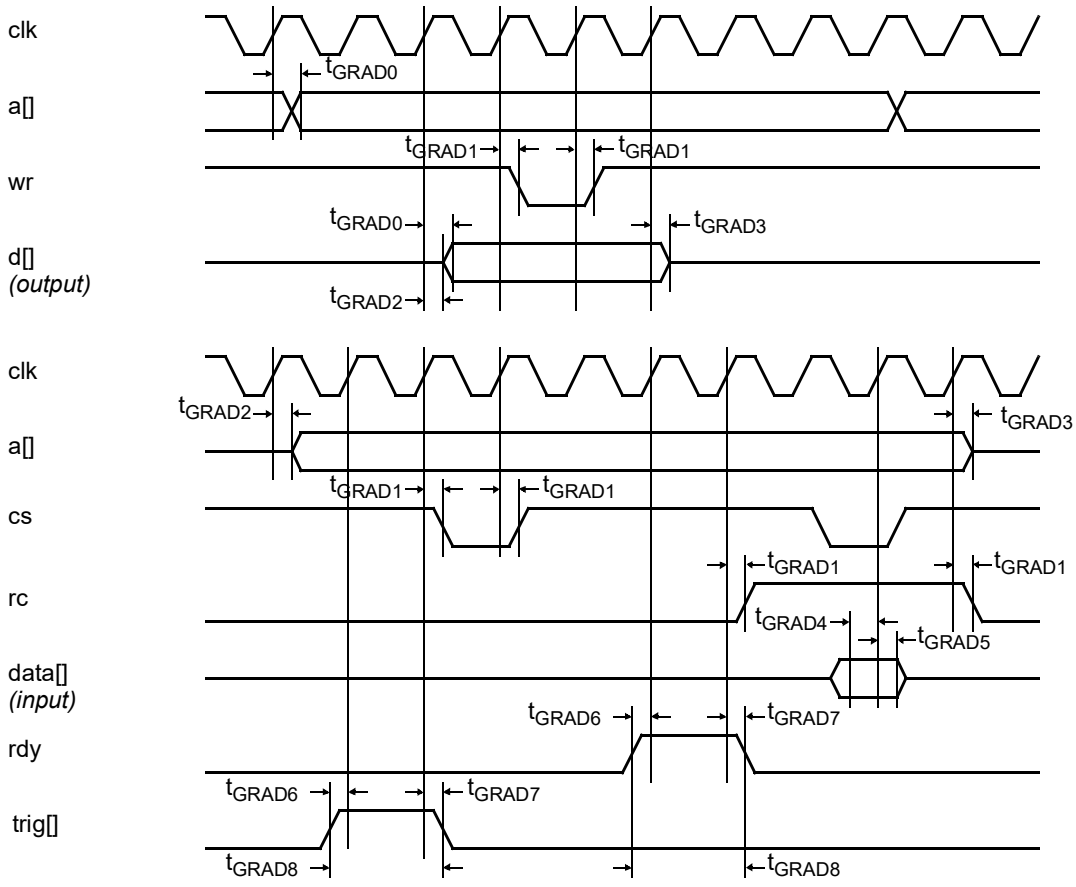


Figure 131. Timing waveforms

*Table 554.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{GRAD0}	a/d clock to output delay	rising *clk* edge	-	TBD	ns
t_{GRAD1}	clock to output delay	rising *clk* edge	-	TBD	ns
t_{GRAD2}	clock to a/d non-tri-state delay	rising *clk* edge	TBD	-	ns
t_{GRAD3}	a/d clock to data tri-state delay	rising *clk* edge	-	TBD	ns
t_{GRAD4}	a/d input to clock setup	rising *clk* edge	TBD	-	ns
t_{GRAD5}	a/d input from clock hold	rising *clk* edge	TBD	-	ns
t_{GRAD6}	input to clock setup	rising *clk* edge	-	TBD	ns
t_{GRAD7}	input from clock hold	rising *clk* edge	TBD	-	ns
t_{GRAD8}	input assertion duration	-	TBD	-	*clk* periods

44.11 Library dependencies

Table 555 shows the libraries used when instantiating the core (VHDL libraries).

*Table 555.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Signals	GRADCDAC component declaration

44.12 Instantiation

This example shows how the core can be instantiated.

```
TBD
```

45 GRAES - Advanced Encryption Standard

45.1 Overview

The Advanced Encryption Standard (AES) is a symmetric encryption algorithm for high throughput application (like audio or video streams). The GRAES core implements the AES-128 algorithm, supporting the Electronic Codebook (ECB) method. The AES-128 algorithm is specified in the "Advanced Encryption Standard (AES)" document, Federal Information Processing Standards (FIPS) Publication 197. The document is established by the National Institute of Standards and Technology (NIST).

The core provides the following internal AMBA AHB slave interface, with sideband signals as per [GRLIB] including:

- interrupt bus
- configuration information
- diagnostic information

The core can be partition in the following hierarchical elements:

- Advanced Encryption Standard (AES) core
- AMBA AHB slave
- GRLIB plug&play wrapper

Note that the core can also be used without the GRLIB plug&play information.

45.2 Operation

The input and output for the AES algorithm each consist of sequences of 128 bits (digits with values of 0 or 1). These sequences will sometimes be referred to as blocks and the number of bits they contain will be referred to as their length. The cipher key for the AES-128 algorithm is a sequence of 128 bits (can also be 192 or 256 bits for other algorithms).

To transfer a 128 bit key or data block four write operations are necessary since the bus interface is 32 bit wide. After supplying a "key will be input" command to the control register, the key is input via four registers. After supplying a "data will be input" command to the control register, the input data is written via four registers. After the last input data register is written, the encryption or decryption is started. The progress can be observed via the debug register. When the operation is completed, an interrupt is generated. The output data is then read out via four registers. Note that the above sequence must be respected. It is not required to write a new key between each data input. There is no command needed for reading out the result.

The implementation requires around 89 clock cycles for a 128 bit data block in encryption direction and around 90 clock cycles for decryption direction. For decryption an initial key calculation is required. This takes around 10 additional clock cycles per every new key. Typically large amounts of data are decrypted (and also encrypted) with the same key. The key initialization for the decryption round does not influence the throughput.

45.3 Background

The Federal Information Processing Standards (FIPS) Publication Series of the National Institute of Standards and Technology (NIST) is the official series of publications relating to standards and guidelines adopted and promulgated under the provisions of the Information Technology Management Reform Act.

The Advanced Encryption Standard (AES) standard specifies the Rijndael algorithm, a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. Rijndael was designed to handle additional block sizes and key lengths, however they are not adopted in this standard.

45.4 AES-128 parameters

The GRAES core implements AES-128. An AES algorithm is defined by the following parameters according to FIPS-197:

- *Nk* number of 32-bit words comprising the cipher key
- *Nr* number of rounds

The AES-128 algorithm is specified as *Nk*=4 and *Nr*=10.

The GRAES core has been verified against the complete set of Known Answer Test vectors included in the AES Algorithm Validation Suite (AESAVS) from National Institute of Standards and Technology (NIST), Information Technology Laboratory, Computer Security Division.

45.5 Throughput

The data throughput for the GRAES core is around 128/90 bits per clock cycle, i.e. approximately 1.4 Mbits per MHz.

The underlaying AES core has been implemented in a dual crypto chip on 250 nm technology as depicted in the figure below. The throughput at 33 MHz operating frequency was 42 Mbit/s, the power consumption was 9,6 mW, and the size was 14,5 kgates.

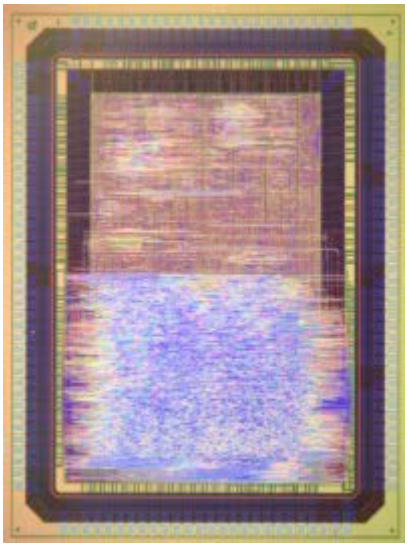


Figure 132. Dual Crypto Chip

45.6 Characteristics

The GRAES core has been synthesized for a Xilinx Virtex-2 XC2V6000-4 devices with the following results:

- LUTs: 5040 (7%)
- 256x1 ROMs (ROM256X1): 128

- Frequency:125 MHz

45.7 Registers

The core is programmed through registers mapped into AHB I/O address space.. Only 32-bit single-accesses to the registers are supported

*Table 556.*GRAES registers

AHB I/O address offset	Register
0x00	Control Register
0x10	Data Input 0 Register
0x14	Data Input 1 Register
0x18	Data Input 2 Register
0x1C	Data Input 3 Register
0x20	Data Output 0 Register
0x24	Data Output 1 Register
0x28	Data Output 2 Register
0x2C	Data Output 3 Register
0x3C	Debug Register

45.7.1 Control Register

*Table 557.*0x00 - CTRL - Control Register

31 2	1	0
RESERVED	DEC	KEY
-	-	-
w	w	w

31-2: - Unused
1: DEC 0 = “encrypt”, 1 = “decrypt” (only relevant when KEY=1)
0: KEY 0 = “data will be input”, 1 = “key will be input”

Note that the Data Input Registers cannot be written before a command is given to the Control Register. Note that the Data Input Registers must then be written in sequence, and all four registers must be written else the core ends up in an undefined state.

The KEY bit determines whether a key will be input (KEY=1), or data will be input (KEY=0). When a “key will be input” command is written, the DEC bit determines whether decryption (DEC=1) or encryption (DEC=0) should be applied to the subsequent data input.

Note that the register cannot be written after a command has been given, until the specific operation completes. A write access will be terminated with an AMBA AHB error response till the Data Input Register 3 has been written, and the with an AMBA AHB retry response till the operation completes. Any read access to this register results in an AMBA AHB error response.

45.7.2 Debug Register (R)

Table 558. 0x3C - DEBUG - Debug Register

31	0
FSM	
r	

31-0: FSM Finite State Machine

Any write access to this register results in an AMBA AHB error response.

45.7.3 Data Input Registers (W)

Table 559. 0x10 - DATAI0 - Data Input 0 Register

31	0
Data/Key(127 downto 96)	
-	
w	

Table 560. 0x14 - DATAI1 - Data Input 1 Register

31	0
Data/Key(95 downto 64)	
-	
w	

Table 561. 0x18 - DATAI2 - Data Input 2 Register

31	0
Data/Key(63 downto 32)	
-	
w	

Table 562. 0x1C - DATAI3 - Data Input 3 Register

31	0
Data/Key(31 downto 0)	
-	
w	

Note that these registers can only be written with a key after a “key will be input” command has been written to the control register. Note that the registers must then be written in sequence, and all four registers must be written else the core ends up in an undefined state.

Note that these registers can only be written with data after a “data will be input” command has been written to the control register, else an AMBA AHB error response is given. Note that the registers must then be written in sequence and all four registers must be written else the core ends up in an undefined state. The encryption or decryption operation is started when the Data Input 3 Register is written to with data.

45.7.4 Data Output Registers (R)

*Table 563.*0x20 - DATAO0 - Data Output 0 Register

31	0
Data(127 downto 96)	
-	
r*	

*Table 564.*0x24 - DATAO1 - Data Output 1 Register

31	0
Data(95 downto 64)	
-	
r*	

*Table 565.*0x28 - DATAO2 - Data Output 2 Register

31	0
Data(63 downto 32)	
-	
r*	

*Table 566.*0x2C - DATAO3 - Data Output 3 Register

31	0
Data(31 downto 0)	
-	
r*	

Note that these registers can only be read after encryption or decryption has been completed. An AMBA AHB retry response is given to read accesses that occur while the encryption or decryption is in progress. If a read access is attempted before an encryption or decryption has even been initiated, then an AMBA AHB erro response is given. Write accesses to these registers result in an AMBA AHB error response.

45.8 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x073. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

45.9 Configuration options

Table 567 shows the configuration options of the core (VHDL generics).

*Table 567.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	0 - NAHBSLV-1	0
ioaddr	Addr field of the AHB I/O BAR	0 - 16#FFF#	0
iomask	Mask field of the AHB I/O BAR	0 - 16#FFF#	16#FFC#
hirq	Interrupt line used by the GRAES	0 - NAHBIRQ-1	0

45.10 Signal descriptions

Table 568 shows the interface signals of the core (VHDL ports).

*Table 568.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBI	*	Input	AHB slave input signals	-
AHBO	*	Output	AHB slave output signals	-
DEBUG[0:4]	N/A	Output	Debug information	-

* see GRLIB IP Library User's Manual

Note that the AES core can also be used without the GRLIB plug&play information. The AMBA AHB signals are then provided as IEEE Std_Logic_1164 compatible scalars and vectors.

45.11 Library dependencies

Table 569 shows libraries used when instantiating the core (VHDL libraries).

*Table 569.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	CRYPTO	Component	GRAES component declarations

45.12 Instantiation

This example shows how the core can be instantiated.

```
library  ieee;
use      ieee.std_logic_1164.all;

library  grlib;
use      grlib.amba.all;

library  gaisler;
use      gaisler.crypto.all;
...
...
   signal debug: std_logic_vector(0 to 4);
..
..
   GRAES0: graes
      generic map (
         hindex         => hindex,
         ioaddr         => ioaddr,
         iomask         => iomask,
         hirq           => hirq)
      port map (
         rstn           => rstn,
         clk            => clk,
         ahbi           => ahbsi,
         ahbo           => ahbso(hindex),
         debug          => debug);
```

46 GRAES_DMA - Advanced Encryption Standard with DMA

46.1 Overview

The Advanced Encryption Standard (AES) is a symmetric encryption algorithm for high throughput applications (like audio or video streams). The GRAES_DMA core implements the AES algorithm with 256-bit key length using CTR mode of operation. The AES algorithm is specified in the "Advanced Encryption Standard (AES)" document, Federal Information Processing Standards (FIPS) Publication 197. The document is established by the National Institute of Standards and Technology (NIST). DMA is used for efficiently transferring plaintext and ciphertext to the cryptographic core with minimum CPU involvement.

The core provides an AMBA AHB master interface, with sideband signals as per [GRLIB] including:

- interrupt bus
- configuration information
- diagnostic information

The core can be partition in the following hierarchical elements:

- Advanced Encryption Standard (AES) core
- AMBA AHB master

46.2 Operation

The input and output for the AES algorithm each consist of sequences of 128 bits (digits with values of 0 or 1). These sequences will sometimes be referred to as blocks and the number of bits they contain will be referred to as their length. The cipher key for the AES algorithm supported in this core is a sequence of 256 bits.

To encrypt a message a descriptor must be setup. It contains pointers to memory locations where the key, initialization vector and plaintext are located. The memory addresses for the key and initialization vector must be word aligned while the plaintext can start at any address. If the previous key and/or init vector are to be reused there are control bits in the descriptor which can be used to make the core skip the fetching of the respective pointers and also subsequently skip the fetching of the actual key and initvector. Currently the initvector and key always have to be loaded for the core to operate correctly.

The core can also read the key and initialization vector from input signals. This is done by setting the keyvector and initvector address pointers to all ones.

When one or more descriptors have been enabled the core can be enabled and it will automatically start fetching the necessary values from memory, split the data into the required blocks, encrypt/decrypt and finally write back the result to memory. When each descriptor is finished the core will set the enable bit to 0. An interrupt can also optionally be generated. The result of the encryption or decryption can be either written back to the same memory address from where the plain or ciphertext was read or to a different location specified in an additional pointer. The layout of the descriptor is shown in the tables below.

Table 570. GRAES_DMA descriptor word 0 (address offset 0x0)

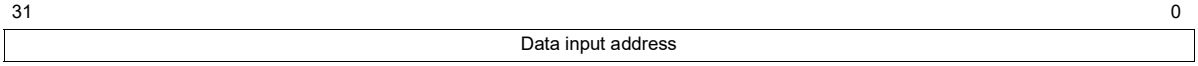
31	21	20	8	7	6	5	4	3	2	1	0
LEN		RESERVED		KE	IV	DO	ED	MD		IE	EN

31: 21	Length (LEN) - Length in bytes of message to process
20: 8	RESERVED
7	Key (KE) - When set a new key will be fetched and used from the memory address set in the key address descriptor word. If not set the currently stored key is used and the key adddress word should not be included in the descriptor.

Table 570. GRAES_DMA descriptor word 0 (address offset 0x0)

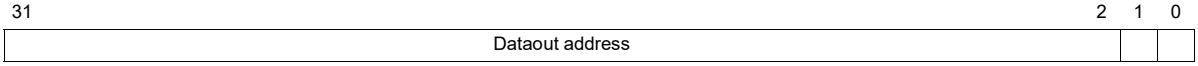
6	Initialization vector (IV) - When set a new initialization vectir will be fetched and used from the memory address set in the initialization vector address descriptor word. If not set the currently stored initialization vector is used and the initialization vector adddress word should not be included in the descriptor.
5	Dataout (DO) - When set the encrypted/decrypted output will be written to the memory address specified in the dataout descriptor word. Otherwise data is written to the same memory address from where the original plaintext/ciphertext was fetched and the dataout address word should not be included in the descriptor.
4	Encrypt-decrypt (ED) - If set to one encryption will be performed otherwise decryption
3	RESERVED
2	RESERVED
1	Interrupt enable (IE) - When set an interrupt will be generated when the processing of the current enabled descriptor is finished and the interrupt enable bit in the control register is set. It should be noted that, the enable bit in the control register might not be cleared yet when a finish interrupt is generated for the last descriptor because the core will read the next descriptor and stop after encountering a '0' on the descriptor control word enable bit which takes some clock cycles.
0	Enable (EN) - When set to '1' indicates that descriptor is enabled and the GRAES core will process it. After the processing is finished this bit will be cleared and the core will jump to the next register. It should be noted that this bit will not be cleared in case a DMA error is encountered at any point of the processing of the current descriptor. If this bit is clear on the first read from the GRAES_DMA core the core will stop processing.

Table 571. GRAES_DMA descriptor word 1 (address offset 0x4)



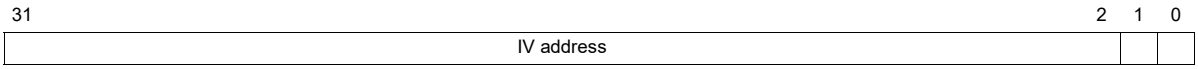
31: 0	Data input address - Memory address pointer where plaintext/ciphertext for encryption/descryption is located.

Table 572. GRAES_DMA descriptor word 2(address offset 0x8 if DO is set, otherwise not exist)



31: 2	Dataout address - Memory address where encrypted/decrypted data shall be stored. If the data should be stored at the same location as the input data (DO bit in word 0 is 0) then this word shall not be included in the descriptor.
1: 0	Reserved

Table 573. GRAES_DMA descriptor word 3(address offset 0xC if DO and IV is set; address offset 0x8 if DO is clear and IV is set; otherwise not exist.)



31: 2	Initialization vector address - Memory address where initialization vector is located. If a new init-vector is not needed (IV bit in word 0 is 0) then this word shall not be included in the descriptor. If this value is set to 0xFFFFFFFF then the core will take the initialization value from the ivin input signal instead.
1: 0	Must be set to zero unless the init value shall be taken from signal input.

Table 574. GRAES_DMA descriptor word 4(address offset 0x10 if DO, IV and KEY is set; address offset 0xC if one of the following bits are set and one of them is clear (DO, IV) and KEY is set; address offset 0x8 if DO and IV is clear and KEY is set; otherwise not exist)

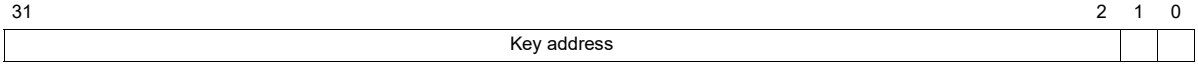
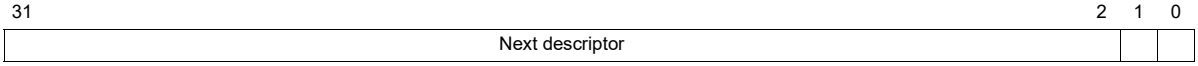


Table 574. GRAES_DMA descriptor word 4(address offset 0x10 if DO, IV and KEY is set; address offset 0xC if one of the following bits are set and one of them is clear (DO, IV) and KEY is set; address offset 0x8 if DO and IV is clear and KEY is set; otherwise not exist)

31: 2	Key address - Memory address where key is located. If a new key is not needed (KE bit in word 0 is 0) then this word shall not be included in the descriptor. If this value is set to 0xFFFFFFFF then the core will take the key value from the keyin input signal instead.
1: 0	Must be set to zero unless the init value shall be taken from signal input.

Table 575. GRAES_DMA descriptor word 5(address offset 0x14 if DO,IV, KEY is set; address offset 0x10 if two of the following bits are set and one of them is clear (DO,IV,KEY); address offset 0xC if two of the following bits are clear and one of them is set (DO,IV,KEY); address offset 0x8 if DO,IV, and KEY is clear)



31: 2	Next descriptor address - Memory address to the next descriptor.
1: 0	Reserved

The descriptor control word should be written last. If one or more words are not included the offsets of the following words should be adjusted accordingly.

46.3 Background

The Federal Information Processing Standards (FIPS) Publication Series of the National Institute of Standards and Technology (NIST) is the official series of publications relating to standards and guidelines adopted and promulgated under the provisions of the Information Technology Management Reform Act.

The Advanced Encryption Standard (AES) standard specifies the Rijndael algorithm, a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. Rijndael was designed to handle additional block sizes and key lengths, however they are not adopted in this standard.

46.4 Characteristics

The GRAES_DMA core has been synthesized for a Actel AX2000-std device with the following results:

- Combinational Cells: 9364 of 21504 (44%)
- Sequential Cells: 2374 of 10752 (22%)
- Total Cells: 11738 of 32256 (37%)
- Block Rams : 0 of 64 (0%)
- Frequency:60 MHz

46.5 Endianness

The core is designed for big-endian systems.

46.6 Registers

The core is programmed through registers mapped into APB address space.

*Table 576.*GRAES_DMA registers

APB address offset	Register
0x0	Control
0x4	Status
0x8	Descriptor address

46.6.1 Control Register

Table 577. 0x00 - CTRL - GRAES_DMA control register

31	4	3	2	1	0
RESERVED	AB	IOE	KS	IE	EN
0	0	0	0	0	0
r	rw	rw	rw	rw	rw

31: 5 RESERVED

4 Abort (AB) - If set to '1' the core will stop processing after reaching a new descriptor and it will self-clear the Abort and Enable (bit-0) bits. The software can check the status of Abort or Enable bits after setting the Abort bit to see when the core becomes idle.It should be noted that if the last descriptor that is processed when abort bit is set causes a DMA error the DMA error status bit will be set so the software has to make sure to handle a potential error after abort. This bit can only be written if it is clear. Reset value : '0'

3 Interrupt On Error (IOE)- If set to '1' then the core will generate an interrupt when an DMA error is encountered. This bit is independent from the bit-1 (IE). Read the section related to DMA error handling for further details about error handling. This bit can only be written if enable bit is clear.Reset value : '0'

2 Keysize (KS) - If set to '1' then the core will use 128 bit key length. Otherwise the core makes use of 256 bit key length. This bit can only be written if enable bit is clear.Reset value : '0'

1 Interrupt Enable (IE) - If set, an interrupt is generated each time a message has been decrypted. This bit can only be written if enable bit is clear.Reset value: '0'.

0 Enable (EN) - Write a one to this bit each time new descriptors are activated in the list. Writing a one will cause the core to read a new descriptor and perform the requested operation. This bit is automatically cleared when the core encounters a descriptor which is not enabled or if a DMA error is encountered. This bit can not be set when the error bit in the status register is set. Software has to clear the error bit in the status register to be able to set the Enable bit again. Read the Status Register section for more details. This bit can only be written when it is clear but it can initiate the processing of the last descriptor again in certain conditions to allow dynamic addition of descriptors to the link list. For further details read the descriptor processing section. Reset value: '0'

46.6.2 Status Register

Table 578. 0x04 - STAT - GRAES_DMA status register

31	0
RESERVED	ER
0	0
r	rw

31: 1 RESERVED

0 Error (ER) - The bit is automatically set to 1 when a DMA error is encountered. When it is set, the enable bit in the ctrl register is locked and can not be set until the error bit is cleared. The error bit is cleared by writing '1' to it.

46.6.3 Descriptor Address

Table 579. 0x08 - ADDR - GRAES_DMA Descriptor address

31	2	1	0
Descriptor address			R
0			0
rw			r

31: 2 Current descriptor address - Points to current descriptor. Can be initialized with a new pointer when the core is disabled. Is updated by the core while it is progressing through the list of descriptors.

1: 0 RESERVED

46.7 Descriptor Processing

Software should set up the descriptor or descriptor chain as it described in the operation section. There are two ways to finish the processing of a descriptor or a descriptor chain. First way is to link the next descriptor to itself for the last descriptor in the chain. After processing, the enable bit of the descriptor control word is automatically cleared so if the next descriptor points to itself the enable bit will be cleared and when read again and the core will stop. Another way is to allocate an empty pointer in which the enable bit of the descriptor word is cleared and link it as a last descriptor. It should be noted that when an empty pointer is allocated, size of it should be equal to the maximum possible size (6 words), although the content of the unused words does not matter as soon as the descriptor is not enabled.

If interrupt enable bit is set, an interrupt will be generated after processing an enabled descriptor. But it should be noted that after an interrupt is generated for the last enabled descriptor in the chain, the enable bit in the control word might not be cleared yet due to core being started to process the next descriptor in which it encounters a cleared enable bit and clears the enable bit in the control register. If the software updates the last disabled pointer in the list and enables it while the core is running, and sets the enable bit in the control register while it is already set and it was reading the last pointer, the core will reprocess the last descriptor if the descriptor enable bit was cleared (it did not read the updated descriptor control word), to make sure operation to continue. If the software makes this modification and sets the enable bit in the control register while it was set, then the core will anyway continue operation because it will read the last descriptor as enabled. This feature allows for safe dynamic addition of descriptors to the list while processing is ongoing. It should be noted the control word of a descriptor should always be written last.

The core will immediately stop processing on an DMA error and software has to take certain steps which are explained in the next section (Error Handling).

46.8 Error Handling

If the core encounters a DMA error, the processing will be immediately stopped (enable bit in the core control register will be cleared) and the error bit in the status register will be set. If interrupt on error is enabled an interrupt will also be generated regardless of interrupt enable bit is set on the descriptor control word. When the error bit in the status register is set, the descriptor address that resides in the descriptor address register is the descriptor that caused the DMA error. It should be noted that the enable bit in the descriptor control word will not be cleared during error. In addition, the enable bit in the control word will be locked if the error bit in the status register is set. So after an error the software has to clear the error bit by writing ‘1’ to the position of error bit in the status register. The software should check the status of error bit after the processing of a descriptor chain is finished (en bit is cleared in the control register), to make sure no error has occurred and clear the error bit after an error in order to be able to proceed with the next operations.

After an error is generated the software should either fix the problem related to the error in the descriptor or set the descriptor address to a new descriptor after clearing the error bit and before starting a new operation. Otherwise it can cause an infinite loop because the enable bit of the descriptor word which causes a DMA error is not automatically cleared and the descriptor address points to the failing descriptor when the core stops due to a DMA error.

46.9 Aborting Operation

It is possible to abort the processing of descriptor at a certain point. When the abort bit in the core’s control word is set, the processing will stop when the current descriptor has finished processing. After

the abort operation has successfully finished the abort bit and enable bit in the core's control word will be cleared. After finishing, the descriptor address register points to the descriptor which is not processed due to stopping. It should be noted that abort bit will not interrupt the processing of the current descriptor when it is set, hence it can not resolve a problem of unresponsive DMA. The abort bit can be used to make sure the core goes into idle state as soon as possible and does not create any transactions on the DMA bus anymore.

46.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x07B. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

46.11 Implementation

46.11.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers, if the GRLIB config package setting *grlib_async_reset_enable* is set.

46.11.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual).

46.12 Configuration options

Table 580 shows the configuration options of the core (VHDL generics).

Table 580. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB BAR	0 - 16#FFF#	0
pmask	Mask field of the APB BAR	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRAES	0 - NAHBIRQ-1	0
extkeyiv	Support key and IV initalization from signals. If this generic is 1 then the keyin and ivin signals can be used to set key and IV values.	0 - 1	0
scantest	Enable SCAN test support	0 - 1	0

46.13 Signal descriptions

Table 581 shows the interface signals of the core (VHDL ports).

*Table 581.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBI	*	Input	AHB master input signals	-
AHBO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
KEYIN[255:0]	N/A	Input	Alternative key input	-
IVIN[127:0]	N/A	Input	Alternative IV input	-

* see GRLIB IP Library User's Manual

46.14 Library dependencies

Table 582 shows libraries used when instantiating the core (VHDL libraries).

*Table 582.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	CRYPTO	Component	GRAES component declarations

46.15 Instantiation

This example shows how the core can be instantiated.

```
entity graes_dma_tb is
   generic(
      hindex:         in    Integer := 0;
      pindex:         in    Integer := 0;
      paddr:          in    Integer := 0;
      pmask:          in    Integer := 16#fff#;
      pirq:           in    Integer := 1);

end entity graes_dma_tb;


signal   rstn:          std_ulogic := '0';
signal   clk:           std_ulogic := '0';
signal   apbi:          apb_slv_in_type;
signal   apbo:          apb_slv_out_vector := (others => apb_none);
signal   ahbmi:         ahb_mst_in_type;
signal   ahbmo:         ahb_mst_out_vector := (others => ahbm_none);

graes0: graes_dma
     generic map(
       hindex          => hindex,
       pindex          => pindex,
       paddr           => paddr,
       pmask           => pmask,
       pirq            => pirq)
     port map(
       rstn            => rstn,
       clk             => clk,
       ahbi            => ahbmi,
```

```
      ahbo            => ahbmo(hindex),
      apbi            => apbi,
      apbo            => apbo(pindex));
```

47 GRCAN - CAN 2.0 Controller with DMA

47.1 Overview

The CAN controller is assumed to operate in an AMBA bus system where both the AMBA AHB bus and the APB bus are present. The AMBA APB bus is used for configuration, control and status handling. The AMBA AHB bus is used for retrieving and storing CAN messages in memory external to the CAN controller. This memory can be located on-chip, as shown in the block diagram, or external to the chip.

The CAN controller supports transmission and reception of sets of messages by use of circular buffers located in memory external to the core. Separate transmit and receive buffers are assumed. Reception and transmission of sets of messages can be ongoing simultaneously.

After a set of message transfers has been set up via the AMBA APB interface the DMA controller initiates a burst of read accesses on the AMBA AHB bus to fetch messages from memory, which are performed by the AHB master. The messages are then transmitted by the CAN core. When a programmable number of messages have been transmitted, the DMA controller issues an interrupt.

After the reception has been set up via the AMBA APB interface, messages are received by the CAN core. To store messages to memory, the DMA controller initiates a burst of write accesses on the AMBA AHB bus, which are performed by the AHB master. When a programmable number of messages have been received, the DMA controller issues an interrupt.

The CAN controller can detect a SYNC message and generate an interrupt, which is also available as an output signal from the core. The SYNC message identifier is programmable via the AMBA APB interface. Separate synchronisation message interrupts are provided.

The CAN controller can transmit and receive messages on either of two CAN busses, but only on one at a time. The selection is programmable via the AMBA APB interface.

Note that it is not possible to receive a CAN message while transmitting one.

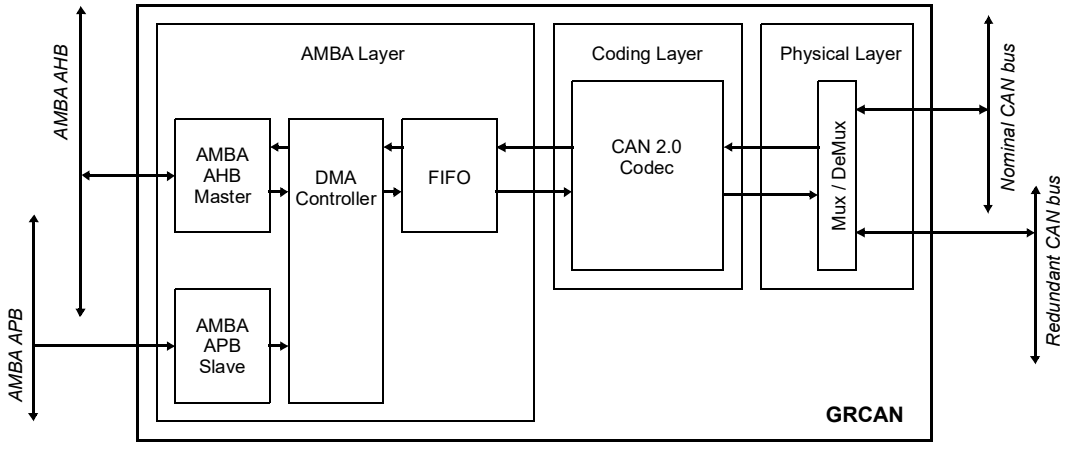


Figure 133. Block diagram

47.1.1 Function

The core implements the following functions:

- CAN protocol
- Message transmission
- Message filtering and reception

- SYNC message reception
- Status and monitoring
- Interrupt generation
- Redundancy selection

47.1.2 Interfaces

The core provides the following external and internal interfaces:

- CAN interface
- AMBA AHB master interface, with sideband signals as per [GRLIB] including:
- cacheability information
- interrupt bus
- configuration information
- diagnostic information
- AMBA APB slave interface, with sideband signals as per [GRLIB] including:
- interrupt bus
- configuration information
- diagnostic information

47.1.3 Hierarchy

The CAN controller core can be partitioned in the following hierarchical elements:

- CAN 2.0 Core
- Redundancy Multiplexer / De-multiplexer
- Direct Memory Access controller
- AMBA APB slave
- AMBA AHB master

47.2 Interface

The external interface towards the CAN bus features two redundant pairs of transmit output and receive input (i.e. 0 and 1).

The active pair (i.e. 0 or 1) is selectable by means of a configuration register bit. Note that all reception and transmission is made over the active pair.

For each pair, there is one enable output (i.e. 0 and 1), each being individually programmable. Note that the enable outputs can be used for enabling an external physical driver. Note that both pairs can be enabled simultaneously. Note that the polarity for the enable/inhibit inputs on physical interface drivers differs, thus the meaning of the enable output is undefined.

Redundancy is implemented by means of Selective Bus Access. Note that the active pair selection above provides means to meet this requirement.

47.3 Protocol

The CAN protocol is based on a CAN 2.0 controller VHDL core. The CAN controller complies with CAN Specification Version 2.0 Part B, except for the overload frame generation.

Note that there are three different CAN types generally defined:

- 2.0A, which considers 29 bit ID messages as an error

- 2.0B Passive, which ignores 29 bit ID messages
- 2.0B Active, which handles 11 and 29 bit ID messages

Only 2.0B Active is implemented.

47.4 Status and monitoring

The CAN interface incorporates status and monitoring functionalities. This includes:

- Transmitter active indicator
- Bus-Off condition indicator
- Error-Passive condition indicator
- Over-run indicator
- 8-bit Transmission error counter
- 8-bit Reception error counter

The status is available via a register and is also stored in a circular buffer for each received message.

47.5 Transmission

The transmit channel is defined by the following parameters:

- base address
- buffer size
- write pointer
- read pointer

The transmit channel can be enabled or disabled.

47.5.1 Circular buffer

The transmit channel operates on a circular buffer located in memory external to the CAN controller. The circular buffer can also be used as a straight buffer. The buffer memory is accessed via the AMBA AHB master interface.

Each CAN message occupies 4 consecutive 32-bit words in memory. Each CAN message is aligned to 4 words address boundaries (i.e. the 4 least significant byte address bits are zero for the first word in a CAN message).

The size of the buffer is defined by the CanTxSIZE.SIZE field, specifying the number of CAN messages * 4 that fit in the buffer.

E.g. CanTxSIZE.SIZE =2 means 8 CAN messages fit in the buffer.

Note however that it is not possible to fill the buffer completely, leaving at least one message position in the buffer empty. This is to simplify wrap-around condition checking.

E.g. CanTxSIZE.SIZE =2 means that 7 CAN messages fit in the buffer at any given time.

47.5.2 Write and read pointers

The write pointer (CanTxWR.WRITE) indicates the position+1 of the last CAN message written to the buffer. The write pointer operates on number of CAN messages, not on absolute or relative addresses.

The read pointer (CanTxRD.READ) indicates the position+1 of the last CAN message read from the buffer. The read pointer operates on number of CAN messages, not on absolute or relative addresses.

The difference between the write and the read pointers is the number of CAN messages available in the buffer for transmission. The difference is calculated using the buffer size, specified by the CanTxSIZE.SIZE field, taking wrap around effects of the circular buffer into account.

Examples:

- There are 2 CAN messages available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE=2 and CanTxRD.READ=0.
- There are 2 CAN messages available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE =0 and CanTxRD.READ =6.
- There are 2 CAN messages available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE =1 and CanTxRD.READ =7.
- There are 2 CAN messages available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE =5 and CanTxRD.READ =3.

When a CAN message has been successfully transmitted, the read pointer (CanTxRD.READ) is automatically incremented, taking wrap around effects of the circular buffer into account. Whenever the write pointer CanTxWR.WRITE and read pointer CanTxRD.READ are equal, there are no CAN messages available for transmission.

47.5.3 Location

The location of the circular buffer is defined by a base address (CanTxADDR.ADDR), which is an absolute address. The location of a circular buffer is aligned on a 1kbyte address boundary.

47.5.4 Transmission procedure

When the channel is enabled (CanTxCTRL.ENABLE=1), as soon as there is a difference between the write and read pointer, a message transmission will be started. Note that the channel should not be enabled if a potential difference between the write and read pointers could be created, to avoid the message transmission to start prematurely.

A message transmission will begin with a fetch of the complete CAN message from the circular buffer to a local fetch-buffer in the CAN controller. After a successful data fetch, a transmission request will be forwarded to the CAN core. If there is at least an additional CAN message available in the circular buffer, a prefetch of this CAN message from the circular buffer to a local prefetch-buffer in the CAN controller will be performed. The CAN controller can thus hold two CAN messages for transmission: one in the fetch buffer, which is fed to the CAN core, and one in the prefetch buffer.

After a message has been successfully transmitted, the prefetch-buffer contents are moved to the fetch buffer (provided that there is message ready). The read pointer (CanTxRD.READ) is automatically incremented after a successful transmission, i.e. after the fetch-buffer contents have been transmitted, taking wrap around effects of the circular buffer into account. If there is at least an additional CAN message available in the circular buffer, a new prefetch will be performed.

If the write and read pointers are equal, no more prefetches and fetches will be performed, and transmission will stop.

If the single shot mode is enabled for the transmit channel (CanTxCTRL.SINGLE=1), any message for which the arbitration is lost, or failed for some other reason, will lead to the disabling of the channel (CanTxCTRL.ENABLE=0), and the message will not be put up for re-arbitration.

Interrupts are provided to aid the user during transmission, as described in detail later in this section. The main interrupts are the Tx, TxEmpty and TxIrq which are issued on the successful transmission of a message, when all messages have been transmitted successfully and when a predefined number of messages have been transmitted successfully. The TxLoss interrupt is issued whenever transmission arbitration has been lost, could also be caused by a communications error. The TxSync interrupt is issued when a message matching the SYNC Code Filter Register.SYNC and SYNC Mask Filter Reg-

ister.MASK registers is successfully transmitted. Additional interrupts are provided to signal error conditions on the CAN bus and AMBA bus.

47.5.5 Straight buffer

It is possible to use the circular buffer as a straight buffer, with a higher granularity than the 1kbyte address boundary limited by the base address (CanTxADDR.ADDR) field.

While the channel is disabled, the read pointer (CanTxRD.READ) can be changed to an arbitrary value pointing to the first message to be transmitted, and the write pointer (CanTxWR.WRITE) can be changed to an arbitrary value.

When the channel is enabled, the transmission will start from the read pointer and continue to the write pointer.

47.5.6 AMBA AHB error

Definition:

- a message fetch occurs when no other messages is being transmitted
- a message prefetch occurs when a previously fetched message is being transmitted
- the local fetch buffer holds the message being fetched
- the local prefetch buffer holds the message being prefetched
- the local fetch buffer holds the message being transmitted by the CAN core
- a successfully prefetched message is copied from the local prefetch buffer to the local fetch buffer when that buffer is freed after a successful transmission.

An AHB error response occurring on the AMBA AHB bus while a CAN message is being fetched will result in a TxAHBErr interrupt.

If the CanCONF.ABORT bit is set to 0b, the channel causing the AHB error will skip the message being fetched from memory and will increment the read pointer. No message will be transmitted.

If the CanCONF.ABORT bit is set to 1b, the channel causing the AHB error will be disabled (CanTxCTRL.ENABLE is cleared automatically to 0 b). The read pointer can be used to determine which message caused the AHB error. Note that it could be any of the four word accesses required to read a message that caused the AHB error.

If the CanCONF.ABORT bit is set to 1b, all accesses to the AMBA AHB bus will be disabled after an AMBA AHB error occurs, as indicated by the CanSTAT.AHBErr bit being 1b. The accesses will be disabled until the CanSTAT register is read, and automatically clearing bit CanSTAT.AHBErr.

An AHB error response occurring on the AMBA AHB bus while a CAN message is being prefetched will not cause an interrupt, but will stop the ongoing prefetch and further prefetch will be prevented temporarily. The ongoing transmission of a CAN message from the fetch buffer will not be affected. When the fetch buffer is freed after a successful transmission, a new fetch will be initiated, and if this fetch results in an AHB error response occurring on the AMBA AHB bus, this will be handled as for the case above. If no AHB error occurs, prefetch will be allowed again.

47.5.7 Enable and disable

When an enabled transmit channel is disabled (CanTxCTRL.ENABLE=0b), any ongoing CAN message transfer request will not be aborted until a CAN bus arbitration is lost or the message has been sent successfully. If the message is sent successfully, the read pointer (CanTxRD.READ) is automatically incremented. Any associated interrupts will be generated.

The progress of the any ongoing access can be observed via the CanTxCTRL.ONGOING bit. The CanTxCTRL.ONGOING must be 0b before the channel can be re-configured safely (i.e. changing

address, size or read pointer). It is also possible to wait for the Tx and TxLoss interrupts described hereafter.

The channel can be re-enabled again without the need to re-configure the address, size and pointers.

Priority inversion is handled by disabling the transmitting channel, i.e. setting CanTxCTRL.ENABLE=0b as described above, and observing the progress, i.e. reading via the CanTxCTRL.ONGOING bit as described above. When the transmit channel is disabled, it can be reconfigured and a higher priority message can be transmitted. Note that the single shot mode does not require the channel to be disabled, but the progress should still be observed as above.

No message transmission is started while the channel is not enabled.

47.5.8 Interrupts

During transmission several interrupts can be generated:

- TxLoss: Message arbitration lost for transmit (could be caused by communications error, as indicated by other interrupts as well)
- TxErrCntr: Error counter incremented for transmit
- TxSync: Synchronization message transmitted
- Tx: Successful transmission of one message
- TxEmpty: Successful transmission of all messages in buffer
- TxIrq: Successful transmission of a predefined number of messages
- TxAHBErr: AHB access error during transmission
- Off: Bus-off condition
- Pass: Error-passive condition

The Tx, TxEmpty and TxIrq interrupts are only generated as the result of a successful message transmission, after the CanTxRD.READ pointer has been incremented.

47.6 Reception

The receive channel is defined by the following parameters:

- base address
- buffer size
- write pointer
- read pointer

The receive channel can be enabled or disabled.

47.6.1 Circular buffer

The receive channel operates on a circular buffer located in memory external to the CAN controller. The circular buffer can also be used as a straight buffer. The buffer memory is accessed via the AMBA AHB master interface.

Each CAN message occupies 4 consecutive 32-bit words in memory. Each CAN message is aligned to 4 words address boundaries (i.e. the 4 least significant byte address bits are zero for the first word in a CAN message).

The size of the buffer is defined by the CanRxSIZE.SIZE field, specifying the number of CAN messages * 4 that fit in the buffer.

E.g. CanRxSIZE.SIZE=2 means 8 CAN messages fit in the buffer.

Note however that it is not possible to fill the buffer completely, leaving at least one message position in the buffer empty. This is to simplify wrap-around condition checking.

E.g. CanRxSIZE.SIZE=2 means that 7 CAN messages fit in the buffer at any given time.

47.6.2 Write and read pointers

The write pointer (CanRxWR.WRITE) indicates the position+1 of the last CAN message written to the buffer. The write pointer operates on number of CAN messages, not on absolute or relative addresses.

The read pointer (CanRxRD.READ) indicates the position+1 of the last CAN message read from the buffer. The read pointer operates on number of CAN messages, not on absolute or relative addresses.

The difference between the write and the read pointers is the number of CAN message positions available in the buffer for reception. The difference is calculated using the buffer size, specified by the CanRxSIZE.SIZE field, taking wrap around effects of the circular buffer into account.

Examples:

- There are 2 CAN messages available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE=2 and CanRxRD.READ=0.
- There are 2 CAN messages available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE =0 and CanRxRD.READ=6.
- There are 2 CAN messages available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE =1 and CanRxRD.READ=7.
- There are 2 CAN messages available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE =5 and CanRxRD.READ=3.

When a CAN message has been successfully received and stored, the write pointer (CanRxWR.WRITE) is automatically incremented, taking wrap around effects of the circular buffer into account. Whenever the read pointer CanRxRD.READ equals (CanRxWR.WRITE+1) modulo (CanRxSIZE.SIZE*4), there is no space available for receiving another CAN message.

The error behavior of the CAN core is according to the CAN standard, which applies to the error counter, buss-off condition and error-passive condition.

47.6.3 Location

The location of the circular buffer is defined by a base address (CanRxADDR.ADDR), which is an absolute address. The location of a circular buffer is aligned on a 1kbyte address boundary.

47.6.4 Reception procedure

When the channel is enabled (CanRxCTRL.ENABLE=1), and there is space available for a message in the circular buffer (as defined by the write and read pointer), as soon as a message is received by the CAN core, an AMBA AHB store access will be started. The received message will be temporarily stored in a local store-buffer in the CAN controller. Note that the channel should not be enabled until the write and read pointers are configured, to avoid the message reception to start prematurely

After a message has been successfully stored the CAN controller is ready to receive a new message. The write pointer (CanRxWR.WRITE) is automatically incremented, taking wrap around effects of the circular buffer into account.

Interrupts are provided to aid the user during reception, as described in detail later in this section. The main interrupts are the Rx, RxFull and RxIrq which are issued on the successful reception of a message, when the message buffer has been successfully filled and when a predefined number of messages have been received successfully. The RxMiss interrupt is issued whenever a message has been received but does not match a message filtering setting, i.e. neither for the receive channel nor for the SYNC message described hereafter.

The RxSync interrupt is issued when a message matching the SYNC Code Filter Register.SYNC and SYNC Mask Filter Register.MASK registers has been successfully received. Additional interrupts are provided to signal error conditions on the CAN bus and AMBA bus.

47.6.5 Straight buffer

It is possible to use the circular buffer as a straight buffer, with a higher granularity than the 1kbyte address boundary limited by the base address (CanRxADDR.ADDR) field.

While the channel is disabled, the write pointer (CanRxWR.WRITE) can be changed to an arbitrary value pointing to the first message to be received, and the read pointer (CanRxRD.READ) can be changed to an arbitrary value.

When the channel is enabled, the reception will start from the write pointer and continue to the read pointer.

47.6.6 AMBA AHB error

An AHB error response occurring on the AMBA AHB bus while a CAN message is being stored will result in an RxAHBErr interrupt.

If the CanCONF.ABORT bit is set to 0b, the channel causing the AHB error will skip the received message, not storing it to memory. The write pointer will be incremented.

If the CanCONF.ABORT bit is set to 1b, the channel causing the AHB error will be disabled (CanRxCTRL.ENABLE is cleared automatically to 0b). The write pointer can be used to determine which message caused the AHB error. Note that it could be any of the four word accesses required to writ a message that caused the AHB error.

If the CanCONF.ABORT bit is set to 1b, all accesses to the AMBA AHB bus will be disabled after an AMBA AHB error occurs, as indicated by the CanSTAT.AHBErr bit being 1b. The accesses will be disabled until the CanSTAT register is read, and automatically clearing bit CanSTAT.AHBErr.

47.6.7 Enable and disable

When an enabled receive channel is disabled (CanRxCTRL.ENABLE=0b), any ongoing CAN message storage on the AHB bus will not be aborted, and no new message storage will be started. Note that only complete messages can be received from the CAN core. If the message is stored successfully, the write pointer (CanRxWR.WRITE) is automatically incremented. Any associated interrupts will be generated.

The progress of the any ongoing access can be observed via the CanRxCTRL.ONGOING bit. The CanRxCTRL.ONGOING must be 0b before the channel can be re-configured safely (i.e. changing address, size or write pointer). It is also possible to wait for the Rx and RxMiss interrupts described hereafter.

The channel can be re-enabled again without the need to re-configure the address, size and pointers.

No message reception is performed while the channel is not enabled

47.6.8 Interrupts

During reception several interrupts can be generated:

- RxMiss: Message filtered away for receive
- RxErrCntr: Error counter incremented for receive
- RxSync: Synchronization message received
- Rx: Successful reception of one message
- RxFull: Successful reception of all messages possible to store in buffer

- RxIrq: Successful reception of a predefined number of messages
- RxAHBErr: AHB access error during reception
- OR: Over-run during reception
- OFF: Bus-off condition
- PASS: Error-passive condition

The Rx, RxFull and RxIrq interrupts are only generated as the result of a successful message reception, after the CanRxWR.WRITE pointer has been incremented.

The OR interrupt is generated when a message is received while a previously received message is still being stored. A full circular buffer will lead to OR interrupts for any subsequently received messages. Note that the last message stored which fills the circular buffer will not generate an OR interrupt. The overrun is also reported with the CanSTAT.OR bit, which is cleared when reading the register.

The error behavior of the CAN core is according to the CAN standard, which applies to the error counter, buss-off condition and error-passive condition.

47.7 Global reset and enable

When the CanCTRL.RESET bit is set to 1b, a reset of the core is performed. The reset clears all the register fields to their default values. Any ongoing CAN message transfer request will be aborted, potentially violating the CAN protocol.

When the CanCTRL.ENABLE bit is cleared to 0b, the CAN core is reset and the configuration bits CanCONF.SCALER, CanCONF.PS1, CanCONF.PS2, CanCONF.RSJ and CanCONF.BPR may be modified. When disabled, the CAN controller will be in sleep mode not affecting the CAN bus by only sending recessive bits. Note that the CAN core requires that 10 recessive bits are received before any reception or transmission can be initiated. This can be caused either by no unit sending on the CAN bus, or by random bits in message transfers.

47.8 Interrupt outputs

Three interrupts are implemented by the CAN interface:

Index:	Name:	Description:
0	IRQ	Common output from interrupt handler
1	TxSYNC	Synchronization message transmitted (optional)
2	RxSYNC	Synchronization message received (optional)

The interrupts are configured by means of the *pirq* VHDL generic and the *singleirq* VHDL generic.

Unlike other IPs in the GRLIB IP core library, GRCAN generates level-based interrupts. This means that the output interrupt, pirq, will be asserted for as long as the core interrupt registers contain unmasked events. It is therefore recommended to clear the content of the interrupt registers after attending the interrupt in order to guarantee that pirq gets properly deasserted.

47.9 Endianness

The core is designed for big-endian systems.

47.10 AXI support

The core is designed for an AMBA system but can be adapted for AXI using the AHBM2AXI adapter.

47.11 Registers

The core is programmed through registers mapped into APB address space.

*Table 583.*GRCAN registers

APB address offset	Register
0x000	Configuration Register
0x004	Status Register
0x008	Control Register
0x018	SYNC Mask Filter Register
0x01C	SYNC Code Filter Register
0x100	Pending Interrupt Masked Status Register
0x104	Pending Interrupt Masked Register
0x108	Pending Interrupt Status Register
0x10C	Pending Interrupt Register
0x110	Interrupt Mask Register
0x114	Pending Interrupt Clear Register
0x200	Transmit Channel Control Register
0x204	Transmit Channel Address Register
0x208	Transmit Channel Size Register
0x20C	Transmit Channel Write Register
0x210	Transmit Channel Read Register
0x214	Transmit Channel Interrupt Register
0x300	Receive Channel Control Register
0x304	Receive Channel Address Register
0x308	Receive Channel Size Register
0x30C	Receive Channel Write Register
0x310	Receive Channel Read Register
0x314	Receive Channel Interrupt Register
0x318	Receive Channel Mask Register
0x31C	Receive Channel Code Register

47.11.1 Configuration Register

*Table 584.*0x000 - CONF - Configuration Register

31	24	23	20	19	16
SCALER		PS1		PS2	
0		0		0	
rw		rw		rw	

15	14	12	11	10	9	8	7	6	5	4	3	2	1	0
	RSJ				BPR				SAM	Silent	Select	Enable1	Enable0	Abort
	0				0				0	0	0	0	0	0
	rw				rw				rw	rw	rw	rw	rw	rw

31-24: SCALER Prescaler setting, 8-bit: system clock / (SCALER +1)
23-20: PS1 Phase Segment 1, 4-bit: (valid range 1 to 15)
19-16: PS2 Phase Segment 2, 4-bit: (valid range 2 to 8)
14-12: RSJ ReSynchronization Jumps, 3-bit: (valid range 1 to 4)

9:8:	BPR	Baud rate, 2-bit: 00b = system clock / (SCALER +1) / 1 01b = system clock / (SCALER +1) / 2 10b = system clock / (SCALER +1) / 4 11b = system clock / (SCALER +1) / 8
5:	SAM	Single sample when 0b. Triple sample when 1b.
4:	SILENT	Listen only to the CAN bus, send recessive bits.
3:	SELECT	Selection receiver input and transmitter output: Select receive input 0 as active when 0b, Select receive input 1 as active when 1b Select transmit output 0 as active when 0b, Select transmit output 1 as active when 1b
2:	ENABLE1	Set value of output 1 enable
1:	ENABLE0	Set value of output 0 enable
0:	ABORT	Abort transfer on AHB ERROR

All bits are cleared to 0 at reset.

Note that constraints on PS1, PS2 and RSJ are defined as:

- PS1 +1 >= PS2
- PS1 > PS2
- PS2 >= RSJ

Note that CAN standard TSEG1 is defined by PS1+1.

Note that CAN standard TSEG2 is defined by PS2.

Note that the SCALER setting defines the CAN time quantum, together with the BPR setting:

system clock / ((SCALER+1) * BPR)

where SCALER is in range 0 to 255, and the resulting division factor due to BPR is 1, 2, 4 or 8.

For a quantum equal to one system clock period, an additional quantum is added to the node delay. Note that for minimizing the node delay, then set either SCALER > 0 or BRP > 0.

Note that the resulting bit rate is:

system clock / ((SCALER+1) * BPR * (1+ PS1+1 + PS2))

where PS1 is in the range 1 to 15, and PS2 is in the range 2 to 8.

Note that RSJ defines the number of allowed re-synchronization jumps according to the CAN standard, being in the range 1 to 4.

For SAM = 0b (single), the bus is sampled once; recommended for high speed buses (SAE class C).

For SAM = 1b (triple), the bus is sampled three times; recommended for low/medium speed buses (SAE class A and B) where filtering spikes on the bus line is beneficial.

Note that the transmit or receive channel active during the AMBA AHB error is disabled if the ABORT bit is set to 1b. Note that all accesses to the AMBA AHB bus will be disabled after an AMBA AHB error occurs while the ABORT bit is set to 1b. The accesses will be disabled until the CanSTAT register is read.

47.11.2 Status Register

*Table 585.*0x004 - STAT - Status register

31 28	27 24	23 16
TxChannels	RxChannels	TxErrCntr
0	0	0
r	r	r

15 8	7	6	5	4	3	2	1	0
RxErrCntr				Active	AHB Err	OR	Off	Pass
0				0	0	0	0	0
r				r	r	r	r	r

31-28: TxChannels Number of TxChannels -1, 4-bit
27-24: RxChannels Number of RxChannels -1, 4-bit
23-16: TxErrCntr Transmission error counter, 8-bit
15-8: RxErrCntr Reception error counter, 8-bit
4: ACTIVE Transmission ongoing
3: AHBErr AMBA AHB master interface blocked due to previous AHB error
2: OR Overrun during reception
1: OFF Bus-off condition
0: PASS Error-passive condition

All bits are cleared to 0 at reset.

The OR bit is set if a message with a matching ID is received and cannot be stored via the AMBA AHB bus, this can be caused by bandwidth limitations or when the circular buffer for reception is already full.

The OR and AHBErr status bits are cleared when the register has been read.

Note that TxErrCntr and RxErrCntr are defined according to CAN protocol.

Note that the AHBErr bit is only set to 1b if an AMBA AHB error occurs while the Can-CONF.ABORT bit is set to 1b.

47.11.3 Control Register

*Table 586.*0x008 - CTRL - Control Register

31 2	1	0
	Reset	Enable
	0	0
	rw	rw

1: RESET Reset complete core when 1
0: ENABLE Enable CAN controller, when 1. Reset CAN controller, when 0

All bits are cleared to 0 at reset.

Note that RESET is read back as 0b.

Note that ENABLE should be cleared to 0b to while other settings are modified, ensuring that the CAN core is properly synchronized.

Note that when ENABLE is cleared to 0b, the CAN interface is in sleep mode, only outputting recessive bits.

Note that the CAN core requires that 10 recessive bits be received before receive and transmit operations can begin.

47.11.4 SYNC Code Filter Register

Table 587. 0x018 - SYNCCODE - SYNC Code Filter Register

31	30	29	28	0
			SYNC	
			0	
			rw	

28-0: SYNC Message Identifier

All bits are cleared to 0 at reset.

Note that Base ID is bits 28 to 18 and Extended ID is bits 17 to 0.

47.11.5 SYNC Mask Filter Register

Table 588. 0x01C - SYNCMASK - SYNC Mask Filter Register

31	30	29	28	0
			MASK	
			0x1FFFFFFF	
			rw	

28-0: MASK Message Identifier

All bits are set to 1 at reset.

Note that Base ID is bits 28 to 18 and Extended ID is bits 17 to 0.

A RxSYNC message ID is matched when:

((Received-ID XOR CanCODE.SYNC) AND CanMASK.MASK) = 0

A TxSYNC message ID is matched when:

((Transmitted-ID XOR CanCODE.SYNC) AND CanMASK.MASK) = 0

47.11.6 Transmit Channel Control Register

Table 589. 0x200 - TXCTRL - Transmit Channel Control Register

31												3	2	1	0
													Sin-gle	Ong oing	Ena ble
													0	0	0
													rw	rw	rw

2: SINGLE Single shot mode
1: ONGOING Transmission ongoing
0: ENABLE Enable channel

All bits are cleared to 0 at reset.

Note that if the SINGLE bit is 1b, the channel is disabled (i.e. the ENABLE bit is cleared to 0b) if the arbitration on the CAN bus is lost.

Note that in the case an AHB bus error occurs during an access while fetching transmit data, and the CanCONF.ABORT bit is 1b, then the ENABLE bit will be reset automatically.

At the time the ENABLE is cleared to 0b, any ongoing message transmission is not aborted, unless the CAN arbitration is lost or communication has failed.

Note that the ONGOING bit being 1b indicates that message transmission is ongoing and that configuration of the channel is not safe.

47.11.7 Transmit Channel Address Register

Table 590. 0x204 - TXADDR - Transmit Channel Address Register

31	10	9	0
ADDR			
0			
rw			

31-10: ADDR Base address for circular buffer

All bits are cleared to 0 at reset.

47.11.8 Transmit Channel Size Register

Table 591. 0x208 - TXSIZE - Transmit Channel Size Register

31	21	20	6	5	0
		SIZE			
		0			
		rw			

20-6: SIZE The size of the circular buffer is SIZE*4 messages

All bits are cleared to 0 at reset.

Valid SIZE values are between 0 and 16384.

Note that each message occupies four 32-bit words.

Note that the resulting behavior of invalid SIZE values is undefined.

Note that only (SIZE*4)-1 messages can be stored simultaneously in the buffer. This is to simplify wrap-around condition checking.

The width of the SIZE field may be made configurable by means of a VHDL generic. In this case it should be set to 16-1 bits width.

47.11.9 Transmit Channel Write Register

Table 592. 0x20C - TXWR - Transmit Channel Write Register

31	20	19	4	3	0
		WRITE			
		0			
		rw			

19-4: WRITE Pointer to last written message +1

All bits are cleared to 0 at reset.

The WRITE field is written to in order to initiate a transfer, indicating the position +1 of the last message to transmit.

Note that it is not possible to fill the buffer. There is always one message position in buffer unused. Software is responsible for not over-writing the buffer on wrap around (i.e. setting WRITE=READ).

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

47.11.10 Transmit Channel Read Register

Table 593. 0x210 - TXRD - Transmit Channel Read Register

31 20	19 4	3 0
	READ	
	0	
	rw	

19-4: READ Pointer to last read message +1

All bits are cleared to 0 at reset.

The READ field is written to automatically when a transfer has been completed successfully, indicating the position +1 of the last message transmitted.

Note that the READ field can be use to read out the progress of a transfer.

Note that the READ field can be written to in order to set up the starting point of a transfer. This should only be done while the transmit channel is not enabled.

Note that the READ field can be automatically incremented even if the transmit channel has been disabled, since the last requested transfer is not aborted until CAN bus arbitration is lost.

When the Transmit Channel Read Pointer catches up with the Transmit Channel Write Register, an interrupt is generated (TxEmpty). Note that this indicates that all messages in the buffer have been transmitted.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

47.11.11 Transmit Channel Interrupt Register

Table 594. 0x214 - TXIRQ - Transmit Channel Interrupt Register

31 20	19 4	3 0
	IRQ	
	0	
	rw	

19-4: IRQ Interrupt is generated when CanTxRD.READ=IRQ, as a consequence of a message transmission

All bits are cleared to 0 at reset.

Note that this indicates that a programmed number of messages have been transmitted.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

47.11.12 Receive Channel Control Register

Table 595. 0x300 - RXCTRL - Receive Channel Control Register

31	2	1	0
		OnGoing	Enable
		0	0
		r	rw

1: ONGOING Reception ongoing (read-only)
0: ENABLE Enable channel

All bits are cleared to 0 at reset.

Note that in the case an AHB bus error occurs during an access while fetching transmit data, and the CanCONF.ABORT bit is 1b, then the ENALBE bit will be reset automatically.

At the time the ENABLE is cleared to 0b, any ongoing message reception is not aborted

Note that the ONGOING bit being 1b indicates that message reception is ongoing and that configuration of the channel is not safe.

47.11.13 Receive Channel Address Register

Table 596. 0x304 - RXADDR - Receive Channel Address Register

31	10	9	0
ADDR			
0			
rw			

31-10: ADDR Base address for circular buffer

All bits are cleared to 0 at reset.

47.11.14 Receive Channel Size Register

Table 597. 0x308 - RXSIZE - Receive Channel Size Register

31	21	20	6	5	0
		SIZE			
		0			
		rw			

20-6: SIZE The size of the circular buffer is SIZE*4 messages

All bits are cleared to 0 at reset.

Valid SIZE values are between 0 and 16384.

Note that each message occupies four 32-bit words.

Note that the resulting behavior of invalid SIZE values is undefined.

Note that only (SIZE*4)-1 messages can be stored simultaneously in the buffer. This is to simplify wrap-around condition checking.

The width of the SIZE field may be made configurable by means of a VHDL generic. In this case it should be set to 16-1 bits width.

47.11.15Receive Channel Write Register

*Table 598.*0x30C - RXWR - Receive Channel Write Register

31 20	19 4	3 0
	WRITE	
	0	
	rw	

19-4: WRITE Pointer to last written message +1

All bits are cleared to 0 at reset.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

The WRITE field is written to automatically when a transfer has been completed successfully, indicating the position +1 of the last message received.

Note that the WRITE field can be use to read out the progress of a transfer.

Note that the WRITE field can be written to in order to set up the starting point of a transfer. This should only be done while the receive channel is not enabled.

47.11.16Receive Channel Read Register

*Table 599.*0x310 - RXRD - Receive Channel Read Register

31 20	19 4	3 0
	READ	
	0	
	rw	

19-4: READ Pointer to last read message +1

All bits are cleared to 0 at reset.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

The READ field is written to in order to release the receive buffer, indicating the position +1 of the last message that has been read out.

Note that it is not possible to fill the buffer. There is always one message position in buffer unused. Software is responsible for not over-reading the buffer on wrap around (i.e. setting WRITE=READ).

47.11.17Receive Channel Interrupt Register

*Table 600.*0x314 - RXIRQ - Receive Channel Interrupt Register

31 20	19 4	3 0
	IRQ	
	0	
	rw	

19-4: IRQ Interrupt is generated when CanRxWR.WRITE=IRQ, as a consequence of a message reception

All bits are cleared to 0 at reset.

Note that this indicates that a programmed number of messages have been received.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

47.11.18Receive Channel Mask Register

*Table 601.*0x318 - RXMASK - Receive Channel Mask Register

31	30	29	28	0
			AM	
			0x1FFFFFFF	
			rw	

28-0: AM Acceptance Mask, bits set to 1b are taken into account in the comparison between the received message ID and the CanRxCODE.AC field

All bits are set to 1 at reset.

Note that Base ID is bits 28 to 18 and Extended ID is bits 17 to 0.

47.11.19Receive Channel Code Register

*Table 602.*0x31C - RXCODE - Receive Channel Code Register

31	30	29	28	0
			AC	
			0	
			rw	

28-0: AC Acceptance Code, used in comparison with the received message

All bits are cleared to 0at reset.

Note that Base ID is bits 28 to 18 and Extended ID is bits 17 to 0.

A message ID is matched when:

((Received-ID XOR CanRxCODE.AC) AND CanRxMASS.AM) = 0

47.11.20Interrupt registers

The interrupt registers give complete freedom to the software, by providing means to mask interrupts, clear interrupts, force interrupts and read interrupt status.

When an interrupt occurs the corresponding bit in the Pending Interrupt Register is set. The normal sequence to initialize and handle a module interrupt is:

- Set up the software interrupt-handler to accept an interrupt from the module.
- Read the Pending Interrupt Register to clear any spurious interrupts.
- Initialize the Interrupt Mask Register, unmasking each bit that should generate the module interrupt.
- When an interrupt occurs, read the Pending Interrupt Status Register in the software interrupt-handler to determine the causes of the interrupt.
- Handle the interrupt, taking into account all causes of the interrupt.
- Clear the handled interrupt using Pending Interrupt Clear Register.

Masking interrupts: After reset, all interrupt bits are masked, since the Interrupt Mask Register is zero. To enable generation of a module interrupt for an interrupt bit, set the corresponding bit in the Interrupt Mask Register.

Clearing interrupts: All bits of the Pending Interrupt Register are cleared when it is read or when the Pending Interrupt Masked Register is read. Reading the Pending Interrupt Masked Register yields the

contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register. Selected bits can be cleared by writing ones to the bits that shall be cleared to the Pending Interrupt Clear Register.

Forcing interrupts: When the Pending Interrupt Register is written, the resulting value is the original contents of the register logically OR-ed with the write data. This means that writing the register can force (set) an interrupt bit, but never clear it.

Reading interrupt status: Reading the Pending Interrupt Status Register yields the same data as a read of the Pending Interrupt Register, but without clearing the contents.

Reading interrupt status of unmasked bits: Reading the Pending Interrupt Masked Status Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register, but without clearing the contents.

The interrupt registers comprise the following:

- Pending Interrupt Masked Status Register [CanPIMSR] R
- Pending Interrupt Masked Register [CanPIMR] R
- Pending Interrupt Status Register [CanPISR] R
- Pending Interrupt Register [CanPIR] R/W
- Interrupt Mask Register [CanIMR] R/W
- Pending Interrupt Clear Register [CanPICR] W

*Table 603.*Interrupt registers

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
															Tx Loss
															0
															*
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rx Miss	Tx Err Cntr	Rx Err Cntr	Tx Sync	Rx Sync	Tx	Rx	Tx Empty	Rx Full	Tx IRQ	Rx IRQ	Tx AHB Err	Rx AHB Err	OR	Off	Pass
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

16:	TxLoss	Message arbitration lost during transmission (could be caused by communications error, as indicated by other interrupts as well)
15:	RxMiss	Message filtered away during reception
14:	TxErrCntr	Transmission error counter incremented
13:	RxErrCntr	Reception error counter incremented
12:	TxSync	Synchronization message transmitted
11:	RxSync	Synchronization message received
10:	Tx	Successful transmission of message
9:	Rx	Successful reception of message
8:	TxEmpty	Successful transmission of all messages in buffer
7:	RxFull	Successful reception of all messages possible to store in buffer
6:	TxIRQ	Successful transmission of a predefined number of messages
5:	RxIRQ	Successful reception of a predefined number of messages
4:	TxAHBErr	AHB error during transmission
3:	RxAHBErr	AHB error during reception
2:	OR	Over-run during reception
1:	OFF	Bus-off condition

0:	PASS	Error-passive condition

All bits in all interrupt registers are reset to 0b after reset.

Note that the TxAHBErr interrupt is generated in such way that the corresponding read and write pointers are valid for failure analysis. The interrupt generation is independent of the CanCONF.ABORT field setting.

Note that the RxAHBErr interrupt is generated in such way that the corresponding read and write pointers are valid for failure analysis. The interrupt generation is independent of the CanCONF.ABORT field setting.

47.12 Memory mapping

The CAN message is represented in memory as shown in table 604.

*Table 604.*CAN message representation in memory.

AHB addr	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x0	IDE	RTR	-	bID											eID	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	eID															
0x4	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	DLC				-	-	-	-	TxErrCntr							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RxErrCntr								-	-	-	-	AhbErr	OR	Off	Pass
0x8	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Byte 0 (first transmitted)								Byte 1							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Byte 2								Byte 3							
0xC	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Byte 4								Byte 5							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Byte 6								Byte 7 (last transmitted)							

Values:	Levels according to CAN standard:	1b is recessive, 0b is dominant
Legend:	Naming and number in according to CAN standard	
IDE	Identifier Extension:	1b for Extended Format, 0b for Standard Format
RTR	Remote Transmission Request:	1b for Remote Frame, 0b for Data Frame
bID	Base Identifier	
eID	Extended Identifier	
DLC	Data Length Code, according to CAN standard:	
		0000b 0 bytes 0001b 1 byte 0010b 2 bytes 0011b 3 bytes

0100b	4 bytes
0101b	5 bytes
0110b	6 bytes
0111b	7 bytes
1000b	8 bytes
OTHERS	illegal

TxErrCntr	Transmission Error Counter
RxErrCntr	Reception Error Counter
AHBErr	AHB interface blocked due to AHB Error when 1b
OR	Reception Over run when 1b
OFF	Bus Off mode when 1b
PASS	Error Passive mode when 1b
Byte 00 to 07	Transmit/Receive data, Byte 00 first Byte 07 last

47.13 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x03D. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

47.14 Implementation

47.14.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

47.14.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

47.15 Configuration options

Table 605 shows the configuration options of the core (VHDL generics).

*Table 605.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFC#
pirq	Interrupt line used by the GRCAN.	0 - NAHBIRQ-1	0
singleirq	Implement only one common interrupt	0 - 1	0
txchannels	Number of transmit channels	1 - 1	1
rxchannels	Number of receive channels	1 - 1	1
ptrwidth	Width of message pointers	16 - 16	16

47.16 Signal descriptions

Table 606 shows the interface signals of the core (VHDL ports).

Table 606. Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-
CANI	Rx[1:0]	Input	Receive lines	-
CANO	Tx[1:0]	Output	Transmit lines	-
	En[1:0]		Transmit enables	-

* see GRLIB IP Library User's Manual

47.17 Signal definitions and reset values

The signals and their reset values are described in table 607.

Table 607. Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
cantx[]	Output	CAN transmit data	Low	Logical 1
canen[]	Output	CAN transmitter enable	High	Logical 0
canrx[]	Input	CAN receive data	Low	-

47.18 Timing

The timing waveforms and timing parameters are shown in figure 134 and are defined in table 608.

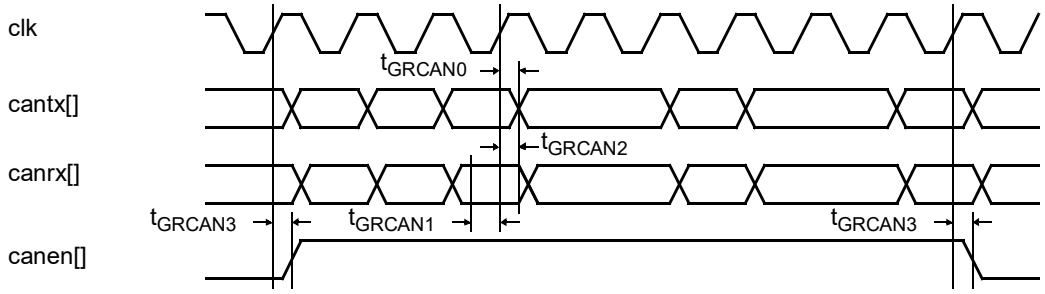


Figure 134. Timing waveforms

Table 608. Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{GRCAN0}	clock to data output delay	rising *clk* edge	-	TBD	ns
t_{GRCAN1}	data input to clock setup	rising *clk* edge	TBD	-	ns
t_{GRCAN2}	data input from clock hold	rising *clk* edge	TBD	-	ns
t_{GRCAN3}	clock to output delay	rising *clk* edge	-	TBD	ns

47.19 Library dependencies

Table 609 shows the libraries used when instantiating the core (VHDL libraries).

*Table 609.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	CAN	Signals, component	GRCAN component and signal declarations.

47.20 Instantiation

This example shows how the core can be instantiated.

```
library  ieee;
use      ieee.std_logic_1164.all;

library  gaisler;
use      gaisler.can.all;

entity example is
   generic (
      padtech:       in    integer := 0);
   port (
      -- CAN interface
      cantx:         out   std_logic_vector(1 downto 0);
      canrx:         in    std_logic_vector(1 downto 0);
      canen:         out   std_logic_vector(1 downto 0);

...

   -- Signal declarations
   signal   rstn:          std_ulogic;
   signal   clk:           std_ulogic;

   signal   ahbmo:         ahb_mst_out_vector := (others => ahbm_none);
   signal   ahbmi:         ahb_mst_in_type;

   signal   apbi:          apb_slv_in_type;
   signal   apbo:          apb_slv_out_vector := (others => apb_none);

   signal   cani0:         can_in_type;
   signal   cano0:         can_out_type;

...

   -- Component instantiation
   grcan0: grcan
      generic map (
         hindex         => 1,
         pindex         => 1,
         paddr          => 16#00C",
         pmask          => 16#FFC",
         pirq           => 1,
         txchannels     => 1,
         rxchannels     => 1,
         ptrwidth       => 16)
      port map (
         rstn           => rstn,
         clk            => clk,
         apbi           => apbi,
         apbo           => apbo(1),
         ahbi           => ahbmi,
         ahbo           => ahbmo(1),
         cani           => cani0,
         cano           => cano0);
```

```
cantx0_pad : outpad
   generic map (tech => padtech) port map (cantx(0), cani0.tx(0));

canrx0_pad : inpad
   generic map (tech => padtech) port map (canrx(0), cani0.rx(0));

canen0_pad : outpad
   generic map (tech => padtech) port map (canen(0), cani0.en(0));

cantx1_pad : outpad
   generic map (tech => padtech) port map (cantx(1), cani0.tx(1));

canrx1_pad : inpad
   generic map (tech => padtech) port map (canrx(1), cani0.rx(1));

canen1_pad : outpad
   generic map (tech => padtech) port map (canen(1), cani0.en(1));
```

48 GRCANFD - CAN Flexible Data-Rate Controller with CANOpen support

48.1 Overview

GRCANFD implements a CAN-FD controller with a top DMA layer handling the configuration of the IP and the communication between the internal CAN-FD controller and a memory external to the IP. It features a generic bus master interface and an AMBA 2.0 APB slave interface. The first one is used for fetching and storing CAN frames from/to external memory. Wrappers adapting this interface to both AMBA 2.0 AHB and AXI4 are available. The APB interface is used for the configuration of the IP.

The internal codec is the CAN-FD controller implementing the MAC and PL sub-layers of the protocol: transmission and reception of frames, error detection and signaling, frame acknowledgment, bit resynchronization, etc. This functionality is compliant with the ISO standard 11898-1:2015 (2nd edition).

Additionally, GRCANFD implements the CANOpen Minimal Set Protocol as per the ECSS-E-ST-50-15C specification, section 9. The IP acts as a slave node (i.e. cannot operate as a CANOpen master). This functionality allows an external CAN node to access the AMBA address space of the device by means of CANOpen PDOs, similarly to RMAP in SpaceWire. The functionality can be configured upon start-up to enable automatic operation in simple applications without need of software intervention. The user can switch between the standard and the CANOpen mode in run-time via the APB interface.

The following block diagram depicts the main components of GRCANFD. The diagram assumes that the AHB wrapper is used, but the functionality is equivalent if the generic bus master or the AXI wrapper are chosen instead.

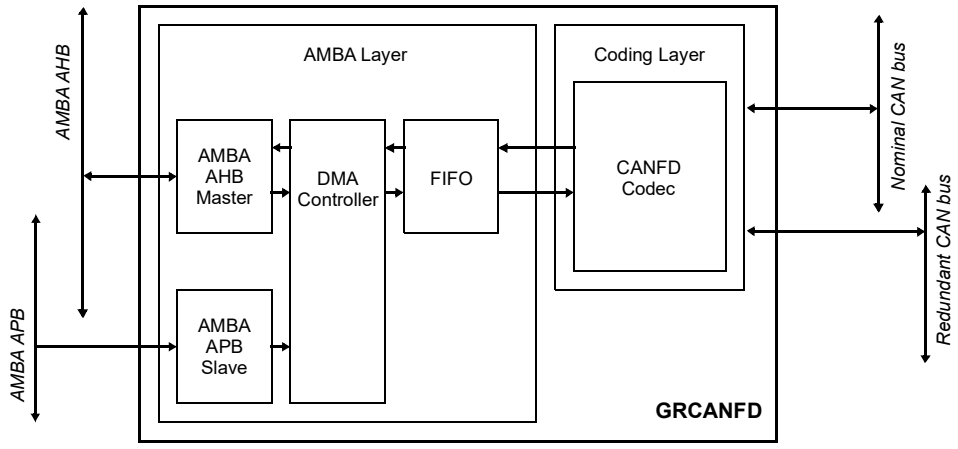


Figure 135. Block diagram

The top AMBA layer contains the memory-mapped registers for the configuration of the IP, accessible through the APB interface. It also controls the global functionality of the IP by fetching frames to be transmitted from the external memory (TX channel) and by storing received frames into the external memory (RX channel) in standard mode, and by processing any input CANOpen commands and providing the reply when applicable in CANOpen mode.

This layer also includes 2 internal SRAMs for the transmit and receive channels. These buffers are used to store the data bytes of a frame, whereas the control bits are stored in registers. The size of each buffer is separately configured via generics, expressed as the number of CAN-FD messages that can be fitted, resulting in memories ranging from 64 to 512 bytes. Note that these buffers are only used in standard mode, no additional memory is required in CANOpen mode.

The external memory where the frames are stored in standard mode is commonly referred to as the circular buffers, and it may be located on-chip or external to the chip. The IP will write and read from this memory through the master interface. There is a separate buffer for the TX and for the RX channels. Although the default topology is circular, straight buffers are also supported. The content of the circular buffers is handled by pointers.

In standard mode, the frames are temporarily buffered in the internal SRAMs. For transmission, the DMA engine fetches frames from the TX circular buffer and the CAN-FD codec transmits them as soon as the SRAM contains a full frame. Likewise, frames received by the codec are temporarily stored in the internal SRAM, and written to the RX circular buffer as soon as the reception is complete. The communication with the circular buffer takes place via the bus master interface.

In CANOpen mode, GRCANFD receives and acknowledges any incoming frame according to the ISO standard, regardless of its format. The frame is then decoded to identify if it is a PDO, SYNC or Heartbeat command, otherwise it is discarded. The CANOpen master node can therefore issue read and write commands, as well as to trigger interrupts in the slave via the CAN interface.

The IP may have up to 3 separate interrupt lines. The first one is the common line, which comprises interrupts for reception and transmission of frames, update on the status of the IP and events on the CAN bus. The other two interrupt lines are used for SYNC messages (for TX and RX channel, respectively) in standard mode. All interrupts are maskable by accessing the APB registers.

GRCANFD implements bus redundancy: its interface includes two RX and two TX lines. The pair of lines to be used can be selected via the AMBA APB interface. The transmission and reception of frames can only occur on one bus at a time. An additional signal is included to enable or disable the output of the external CAN-FD transceiver.

48.1.1 Function

The core implements the following functions:

- CAN-FD core functionality
- TX channel: frame fetching, buffering and transmission
- RX channel: frame reception, filtering, buffering and storage
- CANOpen interpreter supporting PDOs 1-4, SYNC and Heartbeat commands
- SYNC message detection for both TX and RX channels
- Status and monitoring of the IP
- Interrupt generation
- Redundancy selection

48.1.2 Interfaces

The IP core contains the following interfaces:

- 2 CAN interfaces (nominal and redundant)
- Generic bus master interface. If connected to the AMBA AHB wrapper, it includes the sideband signals as per [GRLIB] including:

  - cacheability information

  - interrupt bus

  - configuration information

  - diagnostic information
- AMBA APB slave interface, with sideband signals as per [GRLIB] including:

  - interrupt bus

- configuration information
- diagnostic information

- Configuration interface to set the values of some critical registers after reset.

48.1.3 Hierarchy

GRCANFD can be partitioned in the following main hierarchical elements:

- CAN-FD controller/codec
- CAN bus selector
- Internal SRAM (FIFO)
- Direct Memory Access controller with CANOpen support
- AMBA APB slave
- Generic bus master
- (Optional) Bridges adapting the generic bus master to AMBA 2.0 AHB or AXI4

48.2 CAN Interface

The external interface towards the CAN bus features two redundant pairs of transmit output and receive input (i.e. 0 and 1).

The active pair (i.e. 0 or 1) is selectable by means of a configuration register bit. Note that all reception and transmission is made over the active pair.

For each pair, there is one enable output (i.e. 0 and 1), each being individually programmable. The enable outputs can be used for enabling an external physical driver. Note that both pairs can be enabled simultaneously. The polarity for the enable/inhibit inputs on physical interface drivers may differ, thus the meaning of the enable output is undefined.

Redundancy is implemented by means of Selective Bus Access. Note that the active pair selection above provides means to meet this requirement.

48.3 Protocol compliance

The CAN-FD controller included in GRCANFD is a VHDL core compliant with the ISO standard for CAN with the FD extension: ISO 11898-1:2015 (2nd edition). The minimal CANOpen implementation is compatible with the ECSS-E-ST-50-15C specification.

48.4 Status and monitoring

The GRCANFD register interface provides status and monitoring data, including:

- Operational mode (standard or CANOpen)
- Ongoing transmission
- Bus-off state
- Error-passive mode
- Overrun during reception
- Transmitter error counter (8 bits)
- Receiver error counter (8 bits)

Some of the previous indicators are also written to the circular buffer every time a frame is received and stored in standard mode. This is described later in 48.5.2. Note that this only applies to the RX channel.

48.5 CAN and CAN-FD frames

48.5.1 Frame formats

As defined in the CAN-FD standard, GRCANFD supports the following formats for the transmission and reception of data frames:

- Classical Base Frame Format (CBFF)
- Classical Extended Frame Format (CEFF)
- FD Base Frame Format (FBFF)
- FD Extended Frame Format (FEFF)

Remote frames are compatible with the classical CAN formats (CBFF and CEFF), but not with the new FD formats (FBFF and FEFF), as per the standard. Error Frames (EF) and Overload Frames (OL) are fully supported too.

The following parameters define the format of a frame. A brief description and additional considerations when working with the IP are included. For a complete overview on the formats and their corresponding fields, please refer to the ISO standard for CAN-FD.

- ID: identifier of the frame. It consists of two parts: the base ID (11 bits) and the extended ID (18 bits, optional). Frames with Base format (CBFF and FBFF) only use the base ID, whereas frames with Extended format (CEFF and FEFF) use both.
- IDE: this parameter selects between Base (0b) and Extended Format (1b). It determines if the extended ID shall be appended to the base ID when building a frame, as described above. This parameter does not depend on whether it is an FD frame or not, so the value of other bits such as FDF or BRS is irrelevant.
- RTR: this parameter selects between Data (0b) and Remote frames (1b). Remote frames are only compatible with classical CAN frames. This means that it must be avoided to set both RTR and FDF to 1b when describing a frame, as the behavior is undefined.
- FDF: this parameter determines if the frame has classical (0b: CBFF and CEFF) or FD format (1b: FBFF and FEFF). The FD format allows to transmit larger payloads of up to 64 bytes, and features a more robust CRC algorithm: CRC-17 or CRC-21, depending on the data length. Optionally, it also enables the possibility of switching to a higher bit-rate. This depends on the configuration of the BRS bit, as described below. When setting FDF to 1b, no Remote frames shall be requested (i.e. RTR should be set to 0b).
- BRS: this bit determines whether the bit-rate shall be switched for the data phase of a frame. It is only meaningful for FD frames (FDF set to 1b). Therefore, the bit is ignored for classical CAN frames (FDF to 0b).
- DLC: it selects the data length of the frame. Its encoding varies depending on whether FD format is used or not. For classical frames, the maximum payload is 8 bytes, whereas FD formats may contain up to 64 bytes of data.

It is important to remark that it is optional to enable the BRS bit for FD frames. FD frames with no bit-rate switching still present the advantage of enabling larger data payloads than in classical CAN frames with more robust CRC algorithms, although the full frame would be transmitted at a constant, nominal bit-rate.

48.5.2 Frame memory mapping (standard mode only)

A descriptor is the minimum unit used to represent a frame. It consists of four 32-bit words. The number of descriptors needed to represent a frame varies from 1 to 5. For classical CAN frames (CBFF and CEFF formats) only one descriptor is needed. For CAN-FD frames (FBFF and FEFF formats) the number of descriptors depends on the data length of the frame: whereas frames with up to 8 bytes only

require 1 descriptor, as with classical CAN frames, frames with 64 bytes of data payload require 5 descriptors.

The first descriptor in a frame always includes the control bits describing the frame, as well as information regarding the status of the bus and the IP. This is applicable to both classical CAN and CAN-FD frames. If a frame requires more than one descriptor, the successive descriptors (from 2 up to 5) do not replicate the control and status bits, but only include data bytes. Therefore, each of these descriptors may contain up to 16 bytes of data. Descriptors belonging to the same frame shall always appear consecutively in the circular buffer, taking into account wrap-around conditions.

Each CAN descriptor is aligned to 4 words address boundaries, i.e. the 4 least significant byte address bits are zero for the first word in a CAN descriptor. Note that this frame representation and memory mapping is backwards compatible with GRCAN, which only supports classical CAN format.

Table 610 describes the memory mapping for the first descriptor of a frame, including both control and status bits (first half of the descriptor) and data bytes (second half).

Table 610. CAN message representation in memory.

AHB addr	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x0	IDE	RTR	-	bID											eID	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	eID															
0x4	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	DLC				-	FDF	BRS	-	TxErrCntr							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RxErrCntr								-	-	-	-	BMErr	OR	Off	Pass
0x8	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Byte 0 (first transmitted)								Byte 1							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Byte 2								Byte 3							
0xC	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Byte 4								Byte 5							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Byte 6								Byte 7							

Values: Levels according to CAN standard: 1b is recessive, 0b is dominant

Legend: Naming and numbering according to CAN standard

IDE Identifier Extension: 1b for Extended Format (Base + Extended ID), 0b for Standard Format (Base ID)

RTR Remote Transmission Request: 1b for Remote Frame (only classical CAN), 0b for Data Frame

bID Base Identifier

eID Extended Identifier

DLC Data Length Code, according to CAN standard:

0000b	0 bytes
0001b	1 byte
0010b	2 bytes

		0011b 3 bytes 0100b 4 bytes 0101b 5 bytes 0110b 6 bytes 0111b 7 bytes 1000b 8 bytes 1001b 12 bytes (only CAN-FD) 1010b 16 bytes (only CAN-FD) 1011b 20 bytes (only CAN-FD) 1100b 24 bytes (only CAN-FD) 1101b 32 bytes (only CAN-FD) 1110b 48 bytes (only CAN-FD) 1111b 64 bytes (only CAN-FD)
FDF	Flexible-Data Rate Frame	1b for FD Format, 0b for Classic Format
BRS	Switch Data Bit Rate	1b for bit-rate switch (only CAN-FD frames), 0b for constant bit-rate

TxErrCntr	Transmission Error Counter
RxErrCntr	Reception Error Counter
BMErr	Bus Master Error during previous accesses to the bus when 1b
OR	Reception Overrun when 1b
OFF	Bus-Off mode when 1b
PASS	Error-Passive mode when 1b
Byte 00 to 07	Transmit/Receive data, Byte 00 first Byte 07 last

Status bits (TxErrCntr, RxErrCntr, BMErr, OR, OFF, PASS) only apply to the RX channel. The IP will write these bits to the circular buffer together with the received frame. For transmission, these bits are not used, therefore they can be simply set to 0b when setting up a frame.

Frames with DLC > 1000b will only result in more than 8 bytes of data for FD frames (FDF set to 1b). For classical CAN frames, the maximum data size is limited to 8 bytes, even if DLC is higher than 1000b. Note also that GRCANFD will ignore the BRS bit if FDF is set to 0b (classical CAN format), so that the Nominal Bit Rate would be used for the entire frame.

If an FD frame contains more than 8 data bytes, it requires additional descriptors to be fully represented in the circular buffer. These descriptors only contain data bytes, and its memory representation is the continuation of the second half of the first descriptor:

- Descriptor #2: data bytes 8 to 23.
- Descriptor #3: data bytes 24 to 39.
- Descriptor #4: data bytes 40 to 55.
- Descriptor #5: data bytes 56 to 63 (second half of the descriptor is unused).

For more information regarding the format of the frame and the content of a descriptor, please refer to 48.5.1 and the CAN-FD ISO standard.

48.6 Operational modes

GRCANFD supports two modes of operation: standard and CANOpen. In standard mode, the IP splits the frame in descriptors, as explained in 48.5.2, and the transmission and reception is controlled by hardware pointers handling the TX and RX circular buffers. In CANOpen mode the IP becomes a CANOpen interpreter, decoding any incoming frame and reacting accordingly. This mode interacts with neither the circular buffers nor the internal SRAM memories.

The operational mode shall be selected, i.e. GRCANFD cannot operate in standard mode and CANOpen mode simultaneously. This section describes how to set the default mode after reset and how to switch between modes.

Note that the operational mode only affects the top DMA layer of the IP. The internal CAN-FD controller shall handle all frames in any case, regardless of its format.

48.6.1 Default configuration

The IP features a group of input signals to set the reset value of the most critical registers. This is mainly used in CANOpen mode, since it configures the CAN timing parameters so that communication can be established automatically without software intervention. These reset values are sampled only when the generic *canopen* is set. The parameters that can be configured are:

- Internal CAN-FD controller enable.
- Active CAN bus selection and transceivers output enable.
- CAN bit time parameters (only for classical bit-rate).
- CANOpen mode enable.
- CANOpen node ID.

Note that any changes in these signals are ignored if the IP is not reset.

48.6.2 Switching between standard and CANOpen mode

In order to switch between standard and CANOpen mode, several conditions shall be met before the IP can effectively switch to the requested mode. This prevents the IP from entering any unexpected state if the mode is switched during an ongoing operation.

To switch from standard to CANOpen mode, the following conditions shall be met:

- CANOpen mode enabled: set the Enable bit in the CANOpen Control Register to 1b.
- Transmitter disabled: set the Enable bit in the Transmit Channel Control Register to 0b.
- Receiver disabled: set the Enable bit in the Receive Channel Control Register to 0b.

Any ongoing operation shall finalize (i.e. all FSMs related to the TX and RX channels shall be in idle state) before the request to change to CANOpen mode is granted.

To switch from CANOpen to standard mode, the following conditions shall be met:

- CANOpen mode disabled: set the Enable bit in the CANOpen Control Register to 0b.

Any ongoing operation shall finalize (i.e. the CANOpen FSM shall be in idle state) before the request to change to standard mode is granted.

The status of the IP can be monitored at any time by sampling the Mode bit in the Status Register.

48.7 Standard Mode - Transmission

The circular buffer for the transmit channel is defined by the following parameters:

- base address
- buffer size
- write pointer
- read pointer

The transmit channel can be enabled or disabled.

48.7.1 Circular buffer

The transmit channel operates on a circular buffer located in memory external to GRCANFD. The circular buffer can also be used as a straight buffer. The buffer memory is accessed via the bus master interface (such as AMBA AHB or AXI).

The size of the buffer is defined by the SIZE field in the Transmission Channel Size Register, specifying the number of CAN descriptors * 4 that fit in the buffer.

E.g. CanTxSIZE.SIZE =2 means 8 CAN descriptors fit in the buffer.

Note however that it is not possible to fill the buffer completely, leaving at least one descriptor position in the buffer empty. This is to simplify wrap-around condition checking.

E.g. CanTxSIZE.SIZE =2 means that 7 CAN descriptors fit in the buffer at any given time.

48.7.2 Write and read pointers

The write pointer (WRITE field in the Transmission Channel Write Register) indicates the position +1 of the last CAN descriptor written to the buffer. The write pointer operates on number of CAN descriptors, not on absolute or relative addresses.

The read pointer (READ field in the Transmission Channel Read Register) indicates the position +1 of the last CAN descriptor read from the buffer. The read pointer operates on number of CAN descriptors, not on absolute or relative addresses.

The difference between the write and the read pointers is the number of CAN descriptors available in the buffer for transmission. The difference is calculated using the buffer size, specified by the CanTxSIZE.SIZE field, taking wrap around effects of the circular buffer into account.

Examples:

- There are 2 CAN descriptors available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE=2 and CanTxRD.READ=0.
- There are 2 CAN descriptors available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE =0 and CanTxRD.READ =6.
- There are 2 CAN descriptors available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE =1 and CanTxRD.READ =7.
- There are 2 CAN descriptors available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE =5 and CanTxRD.READ =3.

When a frame has been successfully transmitted, the read pointer (CanTxRD.READ) is automatically updated, taking wrap-around effects of the circular buffer into account. If a frame consists of more than one descriptor, the pointer is not incremented one by one, but it is updated to the next descriptor to be read from the buffer.

Whenever the write pointer CanTxWR.WRITE and read pointer CanTxRD.READ are equal, there are no CAN descriptors available for transmission.

48.7.3 Location

The location of the circular buffer is defined by a base address in Transmission Channel Address Register, which is an absolute address. The location of a circular buffer is aligned on a 1kbyte address boundary.

48.7.4 Transmission procedure

When the channel is enabled (ENABLE bit in the Transmission Channel Control Register equal to 1b), as soon as there is a difference between the write and read pointer, GRCANFD will start fetching the first descriptor of a frame, whose position is indicated by the read pointer. If a frame consists of more than one descriptor, GRCANFD will continue fetching them until the full frame has been read. GRCANFD will decode the DLC field, available in the first descriptor, in order to know how many descriptors shall be fetched in total.

The generic *txbufsize* configures how many frames can be buffered by the core prior to transmission; possible values are between 1 and 8. The data bytes of a frame are stored in the internal TX SRAM,

whereas the control bits are stored in registers. Note that the core will fetch as many frames as the value of *txbufsize* (if available in the circular buffer) regardless of their data length. The core will stop fetching frames if there are no more frames initialized in the circular buffer.

The transmission of a frame does not start until a full frame has been fetched. Buffering the frame minimizes the number of accesses to the circular buffer and allows pre-fetching of additional frames before completing the transmission of the first one, thus increasing the efficiency of the transmitter channel. If *txbufsize* is set to 1, pre-fetching is disabled and the IP will not fetch a new frame until the current transmission has finished.

Note that the TX channel should not be enabled if a potential difference between the write and read pointers could be created, to avoid the descriptor fetching to start prematurely. The TX write pointer shall not be updated until all the descriptors forming the frame are ready to be fetched from the circular buffer.

The read pointer (CanTxRD.READ) is automatically updated after a successful transmission, taking into account wrap around effects of the circular buffer. The content of the local SRAM may then be overwritten, so a new frame would be fetched as soon as the write and read pointers differ.

If the single-shot mode is enabled for the TX channel (CanTxCTRL.SINGLE=1), any frame for which the arbitration is lost, or failed for some other reason such as ACK missing, will lead to the TX channel being automatically disabled (CanTxCTRL.ENABLE=0). The frame will not be put up for re-arbitration and the local SRAM will be emptied, in order not to block future transmissions.

Interrupts are provided to aid the user during transmission, as described in detail later in this section. The main interrupts are the Tx, TxEmpty and TxIrq which are issued respectively after the successful transmission of a frame, when all frames in the circular buffer have been transmitted and when a predefined number of descriptors have been transmitted. The TxLoss interrupt is asserted whenever a transmission does not complete: examples of this are loss of arbitration or communication errors. The TxSync interrupt is issued when a frame matching the SYNC pattern is successfully transmitted. Additional interrupts are provided to signal error conditions on the CAN bus and the AMBA bus.

48.7.5 Straight buffer

It is possible to use the circular buffer as a straight buffer, with a higher granularity than the 1kbyte address boundary limited by the base address in the Transmit Channel Address Register.

While the channel is disabled, the read pointer (READ field in the Transmission Channel Read Register) can be changed to an arbitrary value pointing to the first descriptor to be transmitted, and the write pointer (WRITE field in the Transmission Channel Write Register) can be changed to an arbitrary value.

When the channel is enabled, the transmission will start from the read pointer and continue to the write pointer.

48.7.6 AMBA error

An error response occurring on the AMBA bus while a frame descriptor is being fetched will result in a BmRdErr interrupt. Consequently the bit BMErr of the Status Register is set to 1b, and can only be cleared by reading that register.

If the CanCONF.ABORT bit is set to 0b, GRCANFD will automatically try to fetch the same frame descriptor again.

If the CanCONF.ABORT bit is set to 1b, the TX channel will be disabled (ENABLE bit in the Transmission Channel Control register is cleared automatically to 0b). The read pointer can be used to determine which frame caused the bus master error (but not the specific descriptor, if the frame contains more than one). If the error occurs while the other frame in the local SRAM is being transmitted, the transmission is not interrupted, in order not to disrupt the CAN bus. If the transmission is acknowledged, the pointers are updated accordingly, even if the channel is already disabled.

48.7.7 Enable and disable

When the TX channel is disabled (ENABLE bit in the Transmission Channel Control Register cleared to 0b) during an ongoing transmission, the internal CAN-FD codec will not abort the transmission, but attempt to finish it until the frame is acknowledged, the arbitration is lost or any error is detected. If the frame is sent successfully, the read pointer (CanTxRD.READ) is automatically incremented. Any associated interrupt will be generated.

The progress of any ongoing access can be observed via the ONGOING bit in the Transmission Channel Control Register. The ONGOING bit must be 0b before the channel can be re-configured safely (i.e. changing address, size or read pointer). It is also possible to monitor the Tx and TxLoss interrupts described hereafter.

GRCANFD includes a status bit in the TX channel control register called DisACK. This bit is used to indicate that the TX channel disable request has been acknowledged, and it will take effect as soon as the ongoing transmission finishes or fails to complete. The TX channel is not completely disabled until both ENABLE and DisACK are both 0b.

The channel can be re-enabled again without the need to re-configure the address, size and pointers.

Priority inversion is handled by disabling the transmitting channel, i.e. setting CanTxCTRL.ENABLE=0b as described above, and observing the progress, i.e. reading via the CanTxCTRL.ONGOING and CanTxCTRL.DisACK bits as described above. When the transmit channel is disabled, it can be re-configured and a higher priority frames can be transmitted. Note that the single shot mode does not require the channel to be disabled, but the progress should still be observed as above.

48.7.8 Interrupts

During transmission several interrupts can be generated:

- TxLoss: Frame transmission interrupted due to lost of arbitration, ACK not detected by the transmitter or errors during the transmission.
- TxErrCntr: Increment of the transmitter error counter.
- TxSync: Frame matching the SYNC filter transmitted.
- Tx: Successful transmission of a frame.
- TxEmpty: Successful transmission of all the frames in the circular buffer.
- TxIrq: Successful transmission of a predefined number of frame descriptors. One of the descriptors matches the position programmed by the Transmit Channel Interrupt Register.
- BmRdErr: Bus error while fetching a descriptor via the bus master interface.
- Off: Bus-off condition.
- Pass: Error-passive condition.

The Tx, TxEmpty and TxIrq interrupts are only generated as the result of a successful frame transmission, only once the CanTxRD.READ pointer has been incremented.

48.8 Standard Mode - Reception

The receive channel is defined by the following parameters:

- base address
- buffer size
- write pointer
- read pointer

The receive channel can be enabled or disabled.

48.8.1 Circular buffer

The reception channel operates on a circular buffer located in memory external to GRCANFD. The circular buffer can also be used as a straight buffer. The buffer memory is accessed via the bus master interface (such as AMBA AHB or AXI).

The size of the buffer is defined by the SIZE field in the Reception Channel Size Register, specifying the number of CAN descriptors * 4 that fit in the buffer.

E.g. CanRxSIZE.SIZE =2 means 8 CAN descriptors fit in the buffer.

Note however that it is not possible to fill the buffer completely, leaving at least one descriptor position in the buffer empty. This is to simplify wrap-around condition checking.

E.g. CanRxSIZE.SIZE =2 means that 7 CAN descriptors fit in the buffer at any given time.

48.8.2 Write and read pointers

The write pointer (WRITE field in the Reception Channel Write Register) indicates the position +1 of the last CAN descriptor written to the buffer. The write pointer operates on number of CAN descriptors, not on absolute or relative addresses.

The read pointer (READ field in the Reception Channel Read Register) indicates the position +1 of the last CAN descriptor read from the buffer. The read pointer operates on number of CAN descriptors, not on absolute or relative addresses.

The difference between the write and the read pointers is the number of CAN descriptors available in the buffer for read-out after having received a set of frames. The difference is calculated using the buffer size, specified by the CanRxSIZE.SIZE field, taking wrap around effects of the circular buffer into account.

Examples:

- There are 2 CAN descriptors available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE=2 and CanRxRD.READ=0.
- There are 2 CAN descriptors available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE =0 and CanRxRD.READ=6.
- There are 2 CAN descriptors available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE =1 and CanRxRD.READ=7.
- There are 2 CAN descriptors available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE =5 and CanRxRD.READ=3.

When a frame has been successfully received and all its descriptors have been stored into the circular buffer, the write pointer (CanRxWR.WRITE) is automatically updated, taking wrap-around effects of the circular buffer into account. If a frame consists of more than one descriptor, the pointer is not incremented one by one, but it is updated to the next position to be written to the buffer.

Whenever the read pointer CanRxRD.READ equals (CanRxWR.WRITE+1) modulo (CanRxSIZE.SIZE*4), there is no space available for storing another descriptor. This is signaled by an interrupt (RxFull). If a frame consists of more than one descriptor, GRCANFD will attempt to store each descriptor as long as there is space in the buffer. If there is no space for the next descriptor, the interrupt will be asserted, and GRCANFD will wait until the read pointer is updated to continue storing the frame.

48.8.3 Location

The location of the circular buffer is defined by a base address in the Reception Channel Address Register, which is an absolute address. The location of a circular buffer is aligned on a 1kbyte address boundary.

48.8.4 Reception procedure

When the channel is enabled (ENABLE bit in the Reception Channel Control Register equal to 1b), GRCANFD will evaluate whether any incoming frame shall be stored into the local SRAM. To store a frame, the frame shall match the acceptance filter, which is programmable through the Acceptance Code and Acceptance Mask registers, and there shall be space available in the RX SRAM.

The generic *rxbufsize* configures hoy many frames can be buffered by the core prior to writing them to the circular buffer; possible values are between 1 and 8. The data bytes of a frame are stored in the internal RX SRAM, whereas the control bits are stored in registers. Note that the core will buffer as many frames as the value of *rxbufsize* regardless of their data length.

Once a complete frame is available, GRCANFD will split it into descriptors, as many as needed to represent the full frame. Then the IP will attempt to write each descriptor to the circular buffer, as long as the write and read pointers differ. This process will continue until the full frame is written to the buffer, which is signaled by the Rx interrupt and the write pointer being updated (CanRxWR.WRITE).

By having a local SRAM with capacity for multiple CAN-FD frames, GRCANFD minimizes the risk of missing frames due to conflicts accessing the AHB (or AXI) bus or due to the RX buffer being full. Once a frame is written to the circular buffer and the write pointer is updated, the content of the SRAM can be replaced with a new frame.

Note that the channel should not be enabled until the write and read pointers are configured, to avoid the message reception to start prematurely.

The core supports generation of Overload Frames when the internal buffer is full in order to request other nodes to delay the next transmission. This feature is only available if the size of the RX SRAM is larger than 1 frame (i.e. *rxbufsize* from 2 to 8). If so, the core will transmit Overload Frames if the Overload Frames bit is set in the Receive Channel Control Register and the buffer is full. Up to 2 consecutive Overload Frames will be issued, according to the CAN-FD standard.

If a new frame is received before there being space available in the SRAM, the frame cannot be internally stored and the Overrun condition is detected and signaled via the corresponding interrupt (OR) and the Overrun bit in the Status Register. The OR interrupt is raised only upon the reception of the first frame which causes Overrun. If more frames are received, the OR interrupt is not raised until the Overrun condition is resolved.

Interrupts are provided to aid the user during reception, as described in detail later in this section. The main interrupts are Rx and RxIrq which are issued on the successful reception of a frame and when a predefined number of frame descriptors have been received successfully. The RxMiss interrupt is asserted whenever a frame has been received but does not match any filters (i.e. neither the acceptance nor the SYNC filters) or when a frame is received by the codec but the RX channel is disabled and consequently the frame is not stored in the circular buffer.

The RxFull interrupt is raised when there is no space in the circular buffer for storing the next descriptor. This does not only occur when a complete frame has been received and stored, but also when an individual descriptor has been stored and there is no space in the RX buffer for the following one belonging to the same frame (only applicable to FD frames). In this case, the RX Write Pointer would not represent the position currently being written, as it is only updated after all the descriptors of the frame have been stored.

The RxSync interrupt is issued when a frame matching the SYNC filter has been successfully received. Note that the frame does not need to match the acceptance filter. The interrupt is generated

as soon as the CAN-FD controller finishes receiving the frame. Additional interrupts are provided to signal error conditions on the CAN bus and AMBA bus.

Regardless of the status of the RX channel, the internal CAN-FD codec will check for errors and acknowledge any incoming frames. The codec shall also evaluate frames with any identifier, even if they do not match the Acceptance or SYNC filters. The only way to prevent the internal codec from checking and acknowledging any incoming frames is to disable it via the Control Register (ENABLE bit).

48.8.5 Straight buffer

It is possible to use the circular buffer as a straight buffer, with a higher granularity than the 1kbyte address boundary limited by the base address (CanRxADDR.ADDR) field.

While the channel is disabled, the write pointer (CanRxWR.WRITE) can be changed to an arbitrary value pointing to the first frame to be received, and the read pointer (CanRxRD.READ) can be changed to an arbitrary value.

When the channel is enabled, the reception will start from the write pointer and continue to the read pointer.

48.8.6 AMBA error

An error response occurring on the AMBA bus while a frame descriptor is being stored will result in a BmWrErr interrupt. Consequently the bit BMErr of the Status Register is set to 1b, and it can only be cleared when that register is read.

If the CanCONF.ABORT bit is set to 0b, GRCANFD will retry to write the same descriptor to the circular buffer.

If the CanCONF.ABORT bit is set to 1b, the RX channel will be disabled (CanRxCTRL.ENABLE is cleared automatically to 0b). The write pointer can be used to determine which frame caused the bus master error (but not the specific descriptor, if the frame contains more than one). Any ongoing reception is aborted, and the local SRAM is emptied. The descriptor causing the error is not stored and the write pointer is therefore not updated.

48.8.7 Enable and disable

When the RX channel is disabled (ENABLE bit in the Reception Channel Control Register cleared to 0b) while a frame is being written to the circular buffer, the core will still complete this operation before effectively switching off the channel, thus updating the write pointer (CanRxWR.WRITE) and generating the corresponding interrupts.

When the Receive Channel is disabled, the content of the internal SRAM is emptied once any pending operations are finalized, so that the buffer is fully available once the channel is enabled again.

The progress of an ongoing reception can be observed via the ONGOING bit in the Reception Channel Control Register. The ONGOING bit must be 0b before the channel can be re-configured safely (i.e. changing address, size or write pointer).

GRCANFD includes a status bit in the RX channel control register called DisACK. This bit is used to indicate that the RX channel disable request has been acknowledged, and will take effect as soon as the ongoing reception finishes and the frame is stored into the circular buffer, or until it fails to complete. The RX channel is not completely disabled until both ENABLE and DisACK are both 0b.

The channel can be re-enabled again without the need to re-configure the address, size and pointers.

48.8.8 Interrupts

During reception several interrupts can be generated:

- RxMiss: Frame filtered away during reception

- RxErrCntr: Receive error counter incremented.
- RxSync: Frame matching the SYNC filter received.
- Rx: Successful reception of a frame and storage into the circular buffer.
- RxFull: RX buffer full; no space for the next frame descriptor.
- RxIrq: Successful reception of a predefined number of frame descriptors. One of the descriptors of the frame matches the position programmed by the Receive Channel Interrupt Register.
- BmWrErr: Bus error while storing a frame descriptor via the bus master interface.
- OR: Overrun during reception.
- OFF: Bus-off condition.
- PASS: Error-passive condition.

The Rx and RxIrq interrupts are only generated as the result of a successful frame reception, after the CanRxWR.WRITE pointer has been incremented. RxFull may be generated when a descriptor is stored, regardless of whether it is the last one within a frame or not.

The OR interrupt is generated when a frame is received but the internal SRAM already contains the maximum number of frames yet to be stored into the circular buffer. This may be due to a conflict accessing the bus via the bus master interface, or to the RX circular buffer being full. The assertion of this interrupt implies that one or more frames have already been missed. Once this occurs, the Overrun bit of the Status Register is set to 1b, and will only be cleared by reading the mentioned register.

48.9 CANOpen mode

48.9.1 General

GRCANFD features a minimal implementation of the CANOpen standard. This allows an external CAN node to access the internal address space of the device via specific messages defined in CANOpen.

The IP contains an FSM to handle CANOpen commands. GRCANFD will not transmit any frame spontaneously, it shall be triggered by the master node of the bus. Once a CAN frame is received, the IP will check its format and decode it. The frame shall be a classical CAN data frame, i.e. the bits RTR, FDF and BRS shall be all 0b. Frames not matching this criteria are silently discarded.

A separate bus master interface can be generated for all AMBA accesses performed in CANOpen mode if the generic *sepbus* is set to 1. The AHB index of this second bus is configured via the generic *hindexcopen*.

When decoding the frame, the IP checks the function code (bits 11-8 of the base ID) and the destination node ID (bits 7-0 of the base ID) of the command. If the command is supported by the IP, the corresponding process is triggered, otherwise the command is silently discarded. Supported commands include PDOs 1-4, SYNC and Heartbeat messages. A detailed description of each command and the usage of their data bytes can be found later in this section.

The PDOs 1 and 3 are mapped to write commands, in which the user shall indicate the target AMBA address and the data to write. The IP supports write accesses of 1-4 bytes, depending on the DLC field of the frame. Once the internal write access to the AMBA bus is finalized, the FSM returns to idle state and an interrupt is generated.

The PDOs 2 and 4 are mapped to read commands, in which the user shall indicate the target AMBA address and, optionally, the number of bytes to read. The IP supports read accesses of 0-8 bytes. If no reply length is indicated, the default access is 4 bytes. Once the internal read access to the AMBA bus is finalized, GRCANFD transmits the reply with the requested data. The FSM returns to idle state when the codec has successfully transmitted the frame, and an interrupt is generated.

For all PDOs, the IP supports the reception of RPDOs and, for read commands, the transmission of the corresponding TPDO with the reply.

The Heartbeat command is used to keep alive the bus. It enabled, GRCANFD features an internal counter that is incremented every CAN bit time (i.e. with the Sample Point of the CAN bus). Every time a Heartbeat command is received, the counter is reset. If the counter reaches the programmed timeout, the IP will generate an interrupt and, optionally, switch to the alternate CAN bus, by inverting the LineSel bit in the Configuration Register. This functionality can be enabled in the CANOpen Control Register with the Redundant Control bit.

The SYNC command is used to synchronize all nodes in the network. When GRCANFD receives a SYNC command, an interrupt is generated. Software is required to handle the node synchronization in this case.

If GRCANFD receives a frame which does not pass the preliminary format check (e.g. the frame has FD format or is a Remote Frame) or it does not match any CANOpen object supported, it generates the RxMiss interrupt and the frame is discarded. Then, the IP starts listening to the CAN bus again waiting for the next frame.

48.9.2 Write commands

Write commands are mapped to the CANOpen PDOs 1 and 3, and are used to trigger an internal AMBA write access. Both PDOs will result in the same functionality. The format of the incoming PDOs shall be as follows:

- Frame format: Classical CAN data frame.
- Function code: RPDO1 (0100b) or RPDO3 (1000b).
- Destination node: GRCANFD node ID (node ID field in the CANOpen Control Register).
- DLC: 5 - 8

The DLC field is used to indicate the size of the AMBA write access. The IP supports write accesses of 1 - 4 bytes per PDO. The AMBA address is mapped to the data bytes 1 - 4 of the frame (the first byte being the MSB and the fourth byte the LSB), whereas the data bytes 5 - 8 represent the data to write (the fifth byte being the MSB and the last byte the LSB).

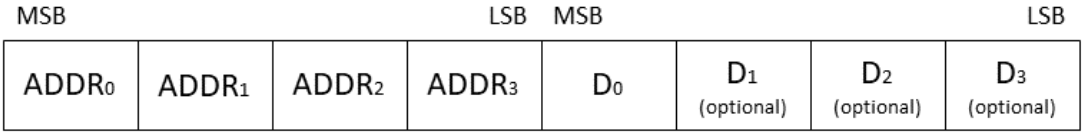


Figure 136. Use of data bytes in write commands (RPDO1 and RPDO3)

Any write command with less than 5 data bytes is simply ignored, as it would not contain enough information to trigger an internal AMBA write access.

48.9.3 Read commands

Read commands are mapped to the CANOpen PDOs 2 and 4, and are used to trigger an internal AMBA read access. Both PDOs will result in the same functionality. The format of the incoming PDOs shall be as follows:

- Frame format: Classical CAN data frame.
- Function code: RPDO2 (0110b) or RPDO4 (1010b).
- Destination node: GRCANFD node ID (node ID field in the CANOpen Control Register).
- DLC: 4 or 5

The size of the AMBA read access is configurable. If the frame contains 4 data bytes (DLC = 4), a standard 32-bit read access is performed. The AMBA address is mapped to these first 4 data bytes of the frame (the first byte being the MSB).

GRCANFD also supports AMBA read accesses of configurable size by means of frames with 5 data bytes (DLC = 5). The four data bytes are still the AMBA address, whereas the fifth byte indicates the size of the read access in number of bytes. Valid sizes range from 0 to 8 bytes, the IP will cap the value to the maximum of 8 bytes.



Figure 137. Use of data bytes in read commands (RPDO2 and RPDO4)

Any read command with DLC other than 4 or 5 is silently discarded by the IP.

Once GRCANFD has completed the internal AMBA read access, it will transmit a TPDO with the data requested as soon as the intermission time (3 bits of nominal CAN bit-rate) is over. The main characteristics of the reply are listed below:

- Frame format: Classical CAN data frame.
- Function code: TPDO2 (0101b) if replying to an RPDO2, or TPDO4 (1001b) for an RPDO4.
- Destination node: GRCANFD node ID (node ID field in the CANOpen Control Register).
- DLC: 0 - 8, according to the size field of the read command (4 by default).

The first data byte of the reply TPDO is the MSB and the last byte, the LSB. Note that it is possible to request the transmission of a TPDO without any data bytes by setting the size of the read access to 0. In this case, no internal read access is performed, and the DLC of the reply is set to 0 accordingly.

48.9.4 SYNC commands

The SYNC commands are employed to synchronize the slave nodes in a CAN bus. Once a frame is interpreted as a SYNC command, GRCANFD triggers an interrupt to be handled by software. The IP does not maintain any internal synchronization timer. The format of the incoming SYNC command is as follows:

- Frame format: Classical CAN data frame.
- Function code: SYNC (0001b).
- Destination node: broadcast (0000000b).

Note that the DLC and the data bytes of a SYNC command are neither checked nor used by the IP.

48.9.5 Heartbeat commands

The Heartbeat command is used as the NMT Error Control mechanism to keep the bus alive. GRCANFD features an internal counter incremented once per CAN bit time if the NMT Error Control functionality is enabled, which is done by setting the CANOpen Heartbeat Timeout Register to a value different from zero. The timeout is expressed in number of CAN bit times.

Every time a Heartbeat command is received, GRCANFD restarts the counter. If the timeout is reached, an interrupt is triggered and the IP may switch to the alternate bus by inverting the LineSel signal of the Configuration Register. This behavior is controlled via the Redundant Control bit in the CANOpen Control Register.

The format of the Heartbeat command shall be as follows:

- Frame format: Classical CAN data frame.

- Function code: NMT Error Control / Heartbeat (1110b).
- Destination node: GRCANFD node ID (node ID field in the CANOpen Control Register).

Note that the DLC and the data bytes of a Heartbeat command are neither checked nor used by the IP.

48.9.6 AMBA errors

As in the standard mode, the IP will raise an interrupt if any access to the internal AMBA bus ends with errors. The CANOpen mode uses the same Bus Master Error interrupt bits as the standard mode. Therefore, if an AMBA write access triggered by a CANOpen write command ends with errors, the IP will raise the BmWrErr interrupt. On the other hand, if the errors are observed during a read access triggered by a CANOpen read command, the corresponding BmRdErr interrupt is raised.

The IP does not use the Abort bit of the Configuration Register, unlike the standard mode. In CANOpen mode, AMBA accesses are never retried (i.e. this corresponds to Abort bit set to 1b) in order to prevent the IP from locking if an external node targeted an invalid AMBA address. Therefore, the internal CANOpen FSM will automatically return to idle after any bus master errors without processing the command. The corresponding interrupt is generated as explained in the paragraph above.

48.9.7 Interrupts

GRCANFD includes four maskable interrupts exclusive to the CANOpen mode:

- CoHbErr: CANOpen Heartbeat timeout error
- CoSync: SYNC command received
- CoWrCmd: CANOpen write command processed
- CoRdCmd: CANOpen read command processed

Note that the IP can still produce other general interrupts in CANOpen mode, such as those related to the CAN error counters, loss of arbitration or bus master errors. More information can be found under the Register section, in 48.14.28.

48.10 Global reset and enable

When the RESET bit in the Control Register is set to 1b, a reset of the whole IP is performed, including the internal CAN-FD codec. The reset clears all the register fields to their default values. Any ongoing frame transfer request will be aborted, potentially violating the CAN protocol.

When the ENABLE bit in the Control Register is cleared to 0b, the CAN-FD codec is reset, so it is safe to modify the configuration registers. When disabled, the CAN-FD controller will be in sleep mode. It will only send recessive bits thus not affecting the CAN bus. It will neither receive nor acknowledge any frame on the bus.

Once the codec is enabled again, it will enter bus integration state before being able to transmit or receive frames. It requires that 11 consecutive recessive bits are detected in the CAN bus prior to starting the normal operation.

48.11 Interrupt outputs

GRCANFD features up to 3 separate output interrupt lines:

Index:	Name:	Description:
0	IRQ	Common output from interrupt handler
1	TxSYNC	Synchronization message transmitted (optional)
2	RxSYNC	Synchronization message received (optional)

The interrupts are configured by means of the *pirq* VHDL generic and the *singleirq* VHDL generic. If *singleirq* is set to 1 there will be only one interrupt line, whose number will be the one set by *pirq*.

Otherwise, if *singleirq* is set to 0, there will be three interrupt lines, whose numbers will be *pirq, pirq+1 and pirq+2*. If *pirq+1 or pirq+2* exceed the maximum number of IRQ lines, the IP will not be able to generate 3 interrupt outputs, even if *singleirq* is set to 0.

Unlike other IPs in the GRLIB IP core library, GRCANFD generates level-based interrupts. This means that the output interrupt, pirq, will be asserted for as long as the core interrupt registers contain unmasked events. It is therefore recommended to clear the content of the interrupt registers after attending the interrupt in order to guarantee that pirq gets properly deasserted.

48.12 Endianness

The core is designed for big-endian and little-endian systems.

GRCANFD automatically detects the endianness of the system by reading a dedicated sideband signal included in the AMBA records. When accessing the generic bus master interface, the data may be realigned and/or swapped, depending on the selected endianness and the type of data involved in the transfer:

- In standard mode, the first 2 words of the first descriptor of a frame contain control and status information, and will always appear at address offsets 0x00 and 0x04, respectively. The core never swaps the bytes of these two words, so they are expected to be written/read in 32-bit accesses. From this point on, the descriptors contain the data bytes of the CAN frame. GRCANFD treats the data bytes as a stream of bytes, i.e. the first data byte of the first descriptor will always appear at offset 0x08, the second byte at offset 0x09 and so on, regardless of the endianness of the system. This implies that GRCANFD will internally swap data bytes in little-endian configurations.
- In CANOpen mode, GRCANFD does not swap the data bytes, so the user must take the endianness of the system into account when accessing the internal AMBA address space of the device via CAN and align the data bytes accordingly.

48.13 Bus master interface

48.13.1 Protocol support

The core features a generic bus master interface to increase the flexibility of the IP. A second bus master interface can be generated for CANOpen transactions by setting the generics *canopen* and *sepbus* to 1. Additionally Frontgrade Gaisler provides wrappers for both AMBA AHB 2.0 and AXI4.

48.13.2 Bus access

In standard mode, GRCANFD uses the generic bus master interface to access the bus in order to fetch and store frame descriptors from/to the corresponding circular buffer. Each access involves a full descriptor. As explained in previous sections, a frame descriptor is formed by four 32-bit words, therefore the access width is always 128 bits for both writing and reading.

When the IP operates in CANOpen mode, the size of the AMBA accesses depend on the write and read commands. Read accesses consist of 1 to 8 bytes, whereas write accesses are of 1 - 4 bytes.

The wrapper for AMBA AHB 2.0 features a bridge to convert the generic bus master into an AHB bus master interface. The 128-bit accesses are split into a burst of 4 incremental 32-bit accesses.

The wrapper for AXI4 features a bridge to convert the generic bus master into an AXI4 bus master interface. For further information regarding the accesses contact Frontgrade Gaisler.

48.14 Registers

The core is programmed through registers mapped into APB address space.

*Table 611.*GRCANFD registers

APB address offset	Register
0x000	Configuration Register
0x004	Status Register
0x008	Control Register
0x00C	Capability Register
0x018	SYNC Mask Filter Register
0x01C	SYNC Code Filter Register
0x040	Nominal Bit-Rate Configuration Register
0x044	Data Bit-Rate Configuration Register
0x048	Transmitter Delay Compensation Register
0x080	CANOpen Control Register
0x084	CANOpen Heartbeat Timeout Register
0x088	CANOpen Heartbeat Count Register
0x08C	CANOpen Status Register
0x100	Pending Interrupt Masked Status Register
0x104	Pending Interrupt Masked Register
0x108	Pending Interrupt Status Register
0x10C	Pending Interrupt Register
0x110	Interrupt Mask Register
0x114	Pending Interrupt Clear Register
0x200	Transmit Channel Control Register
0x204	Transmit Channel Address Register
0x208	Transmit Channel Size Register
0x20C	Transmit Channel Write Register
0x210	Transmit Channel Read Register
0x214	Transmit Channel Interrupt Register
0x300	Receive Channel Control Register
0x304	Receive Channel Address Register
0x308	Receive Channel Size Register
0x30C	Receive Channel Write Register
0x310	Receive Channel Read Register
0x314	Receive Channel Interrupt Register
0x318	Receive Channel Acceptance Mask Register
0x31C	Receive Channel Acceptance Code Register

48.14.1 Configuration Register

*Table 612.*0x000 - CONF - Configuration Register

31 8	7	6	5	4	3	2	1	0
Reserved	LBS	LB	Res	Silent	Select	Enable1	Enable0	Abort
0x000000	0	0	0	0	0*	0*	0*	0
r	rw	rw	r	rw	rw	rw	rw	rw

7: LBS Loop-back selector. Selects between external (0b) and internal (1b) loop-back. Used only if LB is set.
6: LB Loop-back mode. When transmitting a frame, the core will drive the ACK bit and store its own frames.
4: SILENT Listen only to the CAN bus, send recessive bits.
3: SELECT Line selector for transmitter and receiver:
Select receiver input 0 and transmitter output 0 as active when 0b,
Select receiver input 1 and transmitter output 1 as active when 1b.
2: ENABLE1 Set value of output 1 enable. Its polarity depends on the physical transceiver.
1: ENABLE0 Set value of output 0 enable. Its polarity depends on the physical transceiver.
0: ABORT Abort transfer after AMBA bus errors (standard-mode only).

* The reset value of these fields may be overwritten with the input configuration signals (cfg record) if the generic *canopen* is set to 1.

By setting the ABORT bit to 1b, the TX and RX channels are automatically disabled upon detection of an AMBA bus error. Otherwise, the transfer causing the error is issued again. Note that in CANOpen mode, any AMBA accesses resulting in errors are always aborted, regardless of the value of this bit. This prevents any external CAN node from locking the IP if accessing an invalid or protected AMBA address.

When the loop-back mode is enabled, GRCANFD will drive the ACK bit of its own frames. Additionally, these frames will be stored in the RX circular buffer as received frames. The loop-back selector allows to connect the TX and RX bits inside the IP, thus bypassing the external CAN transceiver (internal loop-back), and to have a standard connection to the CAN bus (external loop-back).

The core will only transmit recessive bits over the CAN bus when either the internal loop-back or the Silent mode is activated.

48.14.2 Status Register

*Table 613.*0x004 - STAT - Status Register

31 24	23 16
Reserved	TxErrCntr
0x00	0x00
r	r

15 8	7	6	5	4	3	2	1	0
RxErrCntr	Reserved		MD	Active	BMErr	OR	Bus Off	Err Pass
0x00	0x0		0	0	0	0	0	0
r	r		r	r	r	r	r	r

23-16: TxErrCntr Transmission error counter, 8-bit
15-8: RxErrCntr Reception error counter, 8-bit
5: MD GRCANFD operational mode: standard (0b) or CANOpen (1b)
4: ACTIVE Transmission ongoing
3: BMErr Errors detected during a previous transfer via the bus master interface
2: OR Overrun during reception

1: Bus Off Bus-off condition
0: Err Pass Error-passive condition

The OR bit is set if a frame with an ID matching the acceptance filter cannot be stored within the local SRAM due to lack of space, i.e. the SRAM already is full of CAN messages yet to be stored into the circular buffer (the number of received messages is determined by the generic *rxbufsize*).

The OR and BMErr status bits are cleared when the Status Register is read.

Note that TxErrCntr and RxErrCntr are defined and updated according to the CAN protocol.

Additionally, the fields TxErrCntr, RxErrCntr, BMErr, OR, Bus Off and Error Passive are stored in the circular buffer during reception as part of the first descriptor of every frame.

The MD bit indicates the mode in which GRCANFD operates. This bit only changes once all conditions to switch from one mode to the other are satisfied.

48.14.3 Control Register

Table 614. 0x008 - CTRL - Control Register

31 … 3	2	1	0
Reserved	Dis-Rest	Reset	Enable
0x00000000	0	0	0*
r	rw	w	rw

2: DISREST Disable automatic restart after entering bus-off when 1.
1: RESET Reset complete core when 1. Self-clearing.
0: ENABLE Enable CAN-FD codec when 1. Disable CAN-FD codec when 0.

* The reset value of this field may be overwritten with the input configuration signals (cfg record) if the generic *canopen* is set to 1.

RESET takes effect on the whole IP, including the internal codec. Self-clearing (read back as 0b).

The internal codec shall be disabled by setting ENABLE to 0 before modifying any CAN-related settings, such as the configuration of the bit times. This ensures that the integration with the CAN bus is correctly performed.

When ENABLE is cleared to 0b, the CAN interface is in sleep mode, only outputting recessive bits.

Once the CAN-FD codec is enabled again, it needs to detect 11 consecutive recessive bits on the CAN bus prior to starting the normal operation, i.e. transmit or receive any frame.

48.14.4 Capability Register

Table 615. 0x00C - CAP - Capability Register

31	30	29 … 11	10 … 8	7	6 … 4	3	2	1	0
Rev1	Rev2	Res	TxBufSize	Res	RxBufSize	Res	SepBus	COpen	SingIRQ
-	-	0x00000	-	0	-	0	-	-	-
r	r	r	r	r	r	r	r	r	r

31: Rev1 Revision 1 implemented: support for configurable buffers, FT and scantest, loop-back mode, etc.
30: Rev2 Revision 2 implemented: CANOpen support.
10-8: TxBufSize Size of the internal TX buffer, expressed in number of frames - 1. Determined by the generic *txbufsize*.
6-4: RxBufSize Size of the internal RX buffer, expressed in number of frames - 1. Determined by the generic *rxbufsize*.
2: SepBus Separate bus master interface for CANOpen accesses. Determined by the generic *sepbus*.

1: COpen CANOpen FSM implemented. Determined by the generic *canopen*.
0: SingIRQ The IP has a single IRQ output if SingIRQ is set to 1b, otherwise there are three different interrupts.

The capability register contains general information about the instantiation of the IP, specifically concerning the generics.

If SingIRQ is set to 1b, the IP combines all the interrupts in a single interrupt output. If set to 0b, GRCANFD features up to 3 interrupt outputs: a common line, together with dedicated lines for TxSYNC and RxSYNC interrupts, respectively.

If the bit 31 is set to 1b, the IP implements the Revision 1 of the core, including new generics (txbufsize, rxbufsize, ft, scantest) and control bits (Loop-back mode, Loop-back selector, Overload Frame Enable).

If the bit 30 is set to 1b, the IP implements the Revision 2 of the core, including CANOpen support.

48.14.5 SYNC Mask Filter Register

*Table 616.*0x018 - SYNCMASK - SYNC Mask Filter Register

31 30 29 28	0
Reserved	MASK
0x0	0x1FFFFFFF
r	rw

28-0: MASK Mask for SYNC filter

All bits of the MASK field are set to 1 at reset.

Note that Base ID corresponds to the bits 28 to 18 and Extended ID corresponds to bits 17 to 0.

48.14.6 SYNC Code Filter Register

*Table 617.*0x01C - SYNCCODE - SYNC Code Filter Register

31 30 29 28	0
Reserved	CODE
0x0	0x00000000
r	rw

28-0: CODE Code for SYNC filter

The CAN Base ID corresponds to the bits 28 to 18 and Extended ID corresponds to bits 17 to 0.

The SYNC filter is applied to the transmitted frames as soon as the codec transmits the frame. For the RX channel, the filter is applied as soon as the codec receives the frame, regardless of whether the frame is to be stored into the circular buffer afterwards, so it does not depend on the configuration of the Acceptance filter. Specific interrupts are available for both SYNC filters.

An ID matches the RxSYNC filter when:

((Received-ID) XOR (SYNC CODE)) AND (SYNC MASK) = 0

An ID matches the TxSYNC filter when:

((Transmitted-ID) XOR (SYNC CODE)) AND (SYNC MASK) = 0

48.14.7 Nominal Bit-Rate Configuration Register

Table 618. 0x040 - NOMBR - Nominal Bit-Rate Configuration Register

31	24	23	16
Reserved		SCALER	
0x00		0x00*	
r		rw	

15	10	9	5	4	0
PS1		PS2		SJW	
0x00*		0x00*		0x00*	
rw		rw		rw	

23-16: SCALER Prescaler setting for nominal bit rate, 8-bit: system clock / (SCALER +1)
15-10: PS1 Phase Segment 1 for nominal bit rate, 6-bit
9-4: PS2 Phase Segment 2 for nominal bit rate, 5-bit
3-0: SJW Synchronization Jump Width, 5-bit

* The reset value of these fields may be overwritten with the input configuration signals (cfg record) if the generic *canopen* is set to 1.

The prescaler sets the number of clock cycles per nominal time quantum (plus an offset of 1). PS1, PS2 and SJW define the number of nominal quantum within the Phase Segment 1, Phase Segment 2 and Synchronization Jump Width, respectively.

Certain constraints apply to the previous parameters. Since GRCANFD is an FD enabled implementation with separate prescalers for the nominal and the data bit rate, the valid ranges are as follows:

- Prescaler: 0 - 255
- PS1: 2 - 63
- PS2: 2 - 16
- SJW: 1 - 16

Additional considerations must be taken when defining the parameters:

- PS2 >= SJW
- SJW <= min (PS1, PS2)

For more information regarding the parameters defining the nominal bit time, please refer to the ISO standard 11898-1:2015 (2nd edition).

Therefore, the Nominal time quantum can be obtained as follows:

(system clock period) * (SCALER+1)

whereas the resulting Nominal bit rate is:

(system clock frequency) / ((SCALER+1) * (1+ PS1 + PS2))

48.14.8 Data Bit-Rate Configuration Register

*Table 619.*0x044 - DATABR - Data Bit-Rate Configuration Register

31	24	23	16
Reserved		SCALER	
0x00		0x00	
r		rw	

15	14	13	10	9	8	5	4	3	0
Reserved		PS1		Res.	PS2		Res.	SJW	
00		0x0		0	0x0		0	0x0	
r		rw		r	rw		r	rw	

23-16: SCALER Prescaler setting for data bit rate, 8-bit: system clock / (SCALER +1)
13-10: PS1 Phase Segment 1 for data bit rate, 4-bit
8-5: PS2 Phase Segment 2 for data bit rate, 4-bit
3-0: SJW Synchronization Jump Width, 4-bit

The prescaler sets the number of clock cycles per data time quantum (plus an offset of 1). PS1, PS2 and SJW define the number of data quantum within the Phase Segment 1, Phase Segment 2 and Synchronization Jump Width, respectively.

Certain constraints apply to the previous parameters. Since GRCANFD is an FD enabled implementation with separate prescalers for the nominal and the data bit rate, the valid ranges are as follows:

- Prescaler: 0 - 255
- PS1: 1 - 15
- PS2: 2 - 8
- SJW: 1 - 8

Additional considerations must be taken when defining the parameters:

- SJW <= min (PS1, PS2)
- Data bit-rate >= Nominal bit-rate

For more information regarding the parameters defining the data bit time, please refer to the ISO standard 11898-1:2015 (2nd edition).

Therefore, the Data time quantum can be obtained as follows:

$$(\text{system clock period}) * (\text{SCALER}+1)$$

whereas the resulting Data bit rate is:

$$(\text{system clock frequency}) / ((\text{SCALER}+1) * (1+ \text{PS1} + \text{PS2}))$$

48.14.9 Transmitter Delay Compensation Register

*Table 620.*0x048 - DELCOMP - Transmitter Delay Compensation Register

31	6	5	0
Reserved		TxCompVal	
0x0000000		0x00	
r		rw	

5-0: TxCompVal Number of time quantum for the transmitter delay compensation.

This register configures the delay in terms of number of data quantum to be compensated during the data phase of an FD frame. A maximum of 2 data bit times may be compensated.

If set to 0, the transmitter delay compensation is internally disabled. Otherwise, the register defines the delay between the synchronization segment of a bit and its corresponding secondary sample point, when the signal may be safely read-back.

48.14.10CANOpen Control Register

*Table 621.*0x080 - COCTRL- CANOpen Control Register

31 11	10 4	3	2	1	0
Reserved	ID	Res	RC	SS	EN
0x000000	1111111*	0	0	0	0*
r	rw	r	rw	rw	rw

10-4: ID CANOpen Node ID. Used to determine if the input commands shall be processed by GRCANFD.
2: RC Redundant Control. If set, the IP will switch to the alternate CAN bus after a Heartbeat Timeout error.
1: SS Single-shot mode. If set, the IP will not retry to transmit a reply if it fails due to errors or arbitration.
0: EN Enable CANOpen mode when set to 1b.

* The reset value of these fields may be overwritten with the input configuration signals (cfg record) if the generic *canopen* is set to 1.

48.14.11CANOpen Heartbeat Timeout Register

*Table 622.*0x084 - COHBTO- CANOpen Heartbeat Timeout Register

31 0
TO
0x00000000
rw

31-0: TO Heartbeat Timeout expressed as number of CAN bit times. Once the timeout is reached without receiving a Heartbeat command, GRCANFD will trigger an interrupt.

48.14.12CANOpen Heartbeat Count Register

*Table 623.*0x088 - COHBCT- CANOpen Heartbeat Count Register

31 0
CT
0x00000000
r

31-0: CT Heartbeat internal counter expressed as number of CAN bit times. Restarted every time a Heartbeat command is received. Read-only.

48.14.13CANOpen Status Register

*Table 624.*0x08C - COSTS- CANOpen Status Register

31 4	3 0
Reserved	STS
0x0000000	0x0
r	r

3-0: STS Status of the CANOpen FSM, encoded as follows:
0x0: IDLE. Listening to the CAN bus while waiting for a new RX frame.
0x1: RX_DATABYTES. A frame passing the preliminary format check is being received.
0x2: DECODE. The reception has ended and the IP checks if the frame is a CANOpen object.
0x3: INIT_WRITE. Initializing an AMBA write access after receiving a CANOpen write command.
0x4: END_WRITE. Waiting for the end of the write access.
0x5: INIT_READ. Initializing an AMBA read access after receiving a CANOpen read command.
0x6: END_READ. Waiting for the end of the read access.
0x7: INIT_REPLY. Initializing the TPDO as a reply to a CANOpen read command.
0x8: TX_DATABYTES. Transmitting the TPDO.
0x9: REPLY_NOK. The TPDO failed during transmission due to errors or loss of arbitration.

48.14.14 Transmit Channel Control Register

*Table 625.*0x200 - TXCTRL - Transmit Channel Control Register

31 4	3	2	1	0
Reserved	Dis Ack	Sin-gle	Ong oing	Ena ble
0x0000000	0	0	0	0
r	r	rw	r	rw

3: DisAck Disable request acknowledged
2: SINGLE Single shot mode
1: Ongoing Transmission ongoing
0: ENABLE Enable channel

The TX channel is enabled by setting ENABLE to 1b.

The Ongoing bit indicates whether the codec is transmitting a frame. If the TX channel is disabled while Ongoing is set to 1, the TX channel is disabled, but any ongoing transmission will continue until it finishes (either successfully or with errors). In this case, the DisACK bit is set to 1b to indicate the user that the TX channel will be completely disabled as soon as the ongoing transmission ends.

Changing the configuration of the TX channel (pointers, parameters of the buffer) is not safe when ongoing is set to 1b. The user shall wait until the TX channel is effectively disabled by monitoring both DisACK and ENABLE. When both are set to ‘0’, the codec is no longer transmitting a frame, so the TX channel can be safely reconfigured.

If the single shot mode is enabled, the TX channel will be automatically disabled when a transmission does not complete successfully. This may be due to loss of arbitration, transmission errors, ACK not being sent by the receivers, etc. In this case, the content of the SRAM is removed to avoid blocking any future transmission and GRCANFD does not update the TX read pointer.

The TX channel will also be automatically disabled if the ABORT bit is set to 1b in the Configuration Register, and an AMBA error occurs while fetching a descriptor from the circular buffer. If the codec is transmitting a frame in parallel, the transmission is not interrupted, since that frame did not cause the error accessing the bus (a transmission only starts when all the descriptors describing the frame are fetched).

48.14.15 Transmit Channel Address Register

*Table 626.*0x204 - TXADDR - Transmit Channel Address Register

31 10	9 0
ADDR	Reserved
0	0x000
rw	r

31-10: ADDR Base address for TX circular buffer

48.14.16 Transmit Channel Size Register

*Table 627.*0x208 - TXSIZE - Transmit Channel Size Register

31 21	20 6	5 0
Reserved	SIZE	Reserved
0x000	0x0000	0x00
r	rw	r

20-6: SIZE The size of the TX circular buffer is SIZE*4 descriptors

Valid SIZE values are between 0 and 16384.

Each descriptor occupies four 32-bit words. A frame may consist of 1 to 5 descriptors depending on its format and data length.

Note that the resulting behavior of invalid SIZE values is undefined.

Note that only (SIZE*4)-1 descriptors can be stored simultaneously in the buffer. This is to simplify wrap-around condition checking.

48.14.17 Transmit Channel Write Register

*Table 628.*0x20C - TXWR - Transmit Channel Write Register

31 20	19 4	3 0
Reserved	WRITE	Reserved
0x000	0x0000	0x0
r	rw	r

19-4: WRITE Pointer to last written descriptor+1

The WRITE field is written to in order to initiate a transfer, indicating the position +1 of the last descriptor to transmit.

Note that it is not possible to fill the buffer. There is always one descriptor position in buffer unused. Software is responsible for not over-writing the buffer on wrap around (i.e. setting WRITE=READ).

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

GRCANFD reads this register to know if there are more descriptors to fetch. While operating, it will never modify the write pointer, unless the core is being reset.

48.14.18 Transmit Channel Read Register

*Table 629.*0x210 - TXRD - Transmit Channel Read Register

31 20	19 4	3 0
Reserved	READ	Reserved
0x000	0x0000	0x0
r	rw	r

19-4: READ Pointer to last read descriptor+1

The READ field is written to automatically by the core when a transfer has been completed successfully, indicating the position +1 of the last descriptor transmitted. If a frame consists of more than 1 descriptor, GRCANFD will not increment the pointer one by one, but update it with the final position directly.

Note that the READ field can be use to read out the progress of a transfer of a set of frames.

Note that the READ field can be written to in order to set up the starting point of a transfer. This should only be done while the transmit channel is not enabled and the codec is not transmitting any frame.

Note that the READ field may be incremented even if the transmit channel has been disabled, in case the codec was already transmitting a frame when the ENABLE was set to 0b. As explained previously, this is indicated by both the Ongoing and the DisACK bits of the TX control register.

When the Transmit Channel Read Pointer catches up with the Transmit Channel Write Register, an interrupt is generated (TxEmpty). Note that this implies that all descriptors stored in the buffer have been transmitted.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

48.14.19 Transmit Channel Interrupt Register

*Table 630.*0x214 - TXIRQ - Transmit Channel Interrupt Register

31 20	19 4	3 0
Reserved	IRQ	Reserved
0x000	0x0000	0x0
r	rw	r

19-4: IRQ Interrupt is generated when the value in the Tx Read register becomes equal to IRQ, as a consequence of the transmission of a frame

This register configures the interrupt TxIRQ, which indicates that a programmed number of descriptors have been transmitted.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

Since CAN-FD frames may consist of up to 5 descriptors, the TX read pointer may be incremented internally several times per frame. The content of this register is therefore compared with the TX read pointer after each increment. This means that the interrupt will be asserted if the position of any of the descriptors conforming the frame matches the programmed position. Note that the interrupt is only generated once the transmission of the frame is successfully completed.

48.14.20 Receive Channel Control Register

*Table 631.*0x300 - RXCTRL - Receive Channel Control Register

31 4	3	2	1	0
Reserved	OF	Dis Ack	Ong oing	Ena ble
0x0000000	0	0	0	0
r	rw	r	r	rw

3: OF Overload Frame enable
2: DisAck Disable request acknowledged
1: ONGOING Reception ongoing (read-only)
0: ENABLE Enable channel

The RX channel is enabled by setting ENABLE to 1b.

The Ongoing bit indicates that a reception is taking place. There are two scenarios for this: the codec may be receiving the frame, or the frame has already been received, matches the acceptance filter and it is yet to be written to the circular buffer.

If the user disables the RX channel by setting ENABLE to 0b, any ongoing write access over the AMBA bus will finish. If it was the last descriptor of a frame, the reception is complete and the pointers, status registers and interrupts are updated accordingly. The DisACK bit is then set to 1b to acknowledge the disable request, and is cleared as soon as the RX channel is effectively disabled.

The RX channel may not be configured while Ongoing is 1b, as this may disrupt the status of the CAN bus. The user shall monitor both Ongoing and DisACK bits: when both are 0b and the channel has been disabled, it is safe to modify the configuration of the RX channel.

The RX channel is automatically disabled if the ABORT bit is set to 1b in the Configuration Register and an AMBA error is detected when accessing the bus.

The content of the RX SRAM is emptied whenever the RX channel is effectively disabled.

48.14.21 Receive Channel Address Register

*Table 632.*0x304 - RXADDR - Receive Channel Address Register

31 10	9 0
ADDR	Reserved
0x000000	0x000
rw	r

31-10: ADDR Base address for RX circular buffer

48.14.22 Receive Channel Size Register

*Table 633.*0x308 - RXSIZE - Receive Channel Size Register

31 21	20 6	5 0
Reserved	SIZE	Reserved
0x000	0x0000	0x00
r	rw	r

20-6: SIZE The size of the RX circular buffer is SIZE*4 descriptors

Valid SIZE values are between 0 and 16384.

Note that each descriptor occupies four 32-bit words. A frame may consist of 1 to 5 descriptors depending on its format and data length.

Note that the resulting behavior of invalid SIZE values is undefined.

Note that only (SIZE*4)-1 descriptors can be stored simultaneously in the buffer. This is to simplify wrap-around condition checking.

48.14.23 Receive Channel Write Register

*Table 634.*0x30C - RXWR - Receive Channel Write Register

31 20	19 4	3 0
Reserved	WRITE	Reserved
0x000	0x0000	0x0
r	rw	r

19-4: WRITE Pointer to last written descriptor +1

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

The WRITE field is written to automatically when a transfer has been completed successfully, indicating the position +1 of the last descriptor received. For frames consisting of more than 1 descriptor, GRCANFD will not increment the Write pointer one by one, but update it with the final position directly.

The Write pointer may be updated even after disabling the RX channel. This may be due to the fact that there was a frame being received or already in the local SRAM, and the storage takes place after the disable request.

Note that the WRITE field can be use to read out the progress of a transfer of a set of frames.

Note that the WRITE field can be written to in order to set up the starting point of a transfer. This should only be done while the receive channel is not enabled.

If the RX write pointer catches up with the read pointer, there is no space in the buffer for any additional descriptor. An interrupt is asserted informing that the buffer is full. GRCANFD will detect this and transmit up to 2 consecutive Overload Frames to delay the reception of the next frame. If a new frame arrives under this circumstance, the IP misses it and the Overrun bit in the Status Register is set, together with the Overrun interrupt.

48.14.24 Receive Channel Read Register

*Table 635.*0x310 - RXRD - Receive Channel Read Register

31 20	19 4	3 0
Reserved	READ	Reserved
0x000	0x0000	0x0
r	rw	r

19-4: READ Pointer to last read descriptor +1

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

The READ field is written to in order to release the receive buffer, indicating the position +1 of the last descriptor that has been read out.

Note that it is not possible to fill the buffer. There is always one descriptor position in buffer unused. Software is responsible for not over-reading the buffer on wrap around (i.e. setting WRITE=READ).

GRCANFD will never modify the RX read pointer, unless the IP is being reset.

48.14.25 Receive Channel Interrupt Register

*Table 636.*0x314 - RXIRQ - Receive Channel Interrupt Register

31 20	19 4	3 0
Reserved	IRQ	Reserved
0x000	0x0000	0x0
r	rw	r

19-4: IRQ Interrupt is generated when the value in the Rx Write register becomes equal to IRQ, as a consequence of reception of a message

This register configures the interrupt which indicates that a programmed number of descriptors have been received.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

Since CAN-FD frames may consist of up to 5 descriptors, the RX write pointer may be incremented internally several times per frame. The content of this register is therefore compared with the RX write pointer after each increment. This means that the interrupt will be asserted if the position of any of the descriptors conforming the frame matches the programmed position. Note that the interrupt is only generated once the storage of the frame is successfully completed.

48.14.26 Receive Channel Acceptance Mask Register

*Table 637.*0x318 - RXMASK - Receive Channel Acceptance Mask Register

31 30 29	28 0
Reserved	AM
000	0x1FFFFFFF
r	rw

28-0: AM Acceptance Mask: bits set to 1b are taken into account in the comparison between the received frame ID and the CanRxCODE.AC field

All bits are set to 1 at reset.

Note that Base ID corresponds to the bits 28 to 18 and Extended ID corresponds to the bits 17 to 0.

48.14.27 Receive Channel Acceptance Code Register

*Table 638.*0x31C - RXCODE - Receive Channel Acceptance Code Register

31 30 29	28 0
Reserved	AC
000	0x00000000
r	rw

28-0: AC Acceptance Code, used in comparison with the received frame ID

Note that Base ID corresponds to the bits 28 to 18 and Extended ID corresponds to the bits 17 to 0.

When a frame is received by the internal CAN-FD codec, GRCANFD applies the Acceptance filter to decide whether it should be stored into the circular buffer. A frame matches the filter if the following condition is verified:

((Received-ID) XOR (ACCPT CODE)) AND (ACCPT MASK) = 0

48.14.28 Interrupt registers

The interrupt registers give complete freedom to the software, by providing means to mask interrupts, clear interrupts, force interrupts and read interrupt status.

When an interrupt occurs the corresponding bit in the Pending Interrupt Register is set. The normal sequence to initialize and handle a module interrupt is:

- Set up the software interrupt-handler to accept an interrupt from the module.
- Read the Pending Interrupt Register to clear any spurious interrupts.
- Initialize the Interrupt Mask Register, unmasking each bit that should generate the module interrupt.
- When an interrupt occurs, read the Pending Interrupt Status Register in the software interrupt-handler to determine the causes of the interrupt.
- Handle the interrupt, taking into account all causes of the interrupt.
- Clear the handled interrupt using Pending Interrupt Clear Register.

Masking interrupts: After reset, all interrupt bits are masked, since the Interrupt Mask Register is zero. To enable generation of a module interrupt for an interrupt bit, set the corresponding bit in the Interrupt Mask Register.

Clearing interrupts: All bits of the Pending Interrupt Register are cleared when it is read or when the Pending Interrupt Masked Register is read. Reading the Pending Interrupt Masked Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register. Selected bits can be cleared by writing ones to the bits that shall be cleared to the Pending Interrupt Clear Register.

Forcing interrupts: When the Pending Interrupt Register is written, the resulting value is the original contents of the register logically OR-ed with the write data. This means that writing the register can force (set) an interrupt bit, but never clear it.

Reading interrupt status: Reading the Pending Interrupt Status Register yields the same data as a read of the Pending Interrupt Register, but without clearing the contents.

Reading interrupt status of unmasked bits: Reading the Pending Interrupt Masked Status Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register, but without clearing the contents.

The interrupt registers comprise the following:

- Pending Interrupt Masked Status Register [CanPIMSR] R
- Pending Interrupt Masked Register [CanPIMR] R
- Pending Interrupt Status Register [CanPISR] R
- Pending Interrupt Register [CanPIR] R/W
- Interrupt Mask Register [CanIMR] R/W
- Pending Interrupt Clear Register [CanPICR] W

*Table 639.*Interrupt registers

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
											COHbErr	COSync	COWrCmd	CORdCmd	TxLoss
											0	0	0	0	0
											*	*	*	*	*
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RxMiss	TxErrCntr	RxErrCntr	TxSync	RxSync	Tx	Rx	TxEmpty	RxFull	TxIRQ	RxIRQ	BmRdErr	BmWrErr	OR	Off	Pass
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

20:	COHbErr	CANOpen Heartbeat Timeout Error
19:	COSync	CANOpen SYNC command received
18:	COWrCmd	CANOpen write command processed
17:	CORdCmd	CANOpen read command processed
16:	TxLoss	Unsuccessful transmission (due to loss of arbitration or errors in the frame)
15:	RxMiss	Message filtered away during reception
14:	TxErrCntr	Transmission error counter incremented
13:	RxErrCntr	Reception error counter incremented
12:	TxSync	Synchronization message transmitted by the codec
11:	RxSync	Synchronization message received by the codec
10:	Tx	Successful transmission of message
9:	Rx	Successful reception of message
8:	TxEmpty	Successful transmission of all frames in TX circular buffer
7:	RxFull	Successful reception of all frames possible to store in RX circular buffer
6:	TxIRQ	The Tx Read Pointer is equal to the value stored in Tx IRQ register
5:	RxIRQ	The Rx Write Pointer is equal to the value stored in Rx IRQ register
4:	BmRdErr	Error during AMBA read access
3:	BmWrErr	Error during AMBA write access
2:	OR	Over-run during reception
1:	OFF	Bus-off condition
0:	PASS	Error-passive condition

All bits in all interrupt registers are reset to 0b after reset.

Note that the BmRdErr interrupt is generated in such way that the corresponding read and write pointers are valid for failure analysis in standard mode. The interrupt generation is independent of the CanCONF.ABORT field setting.

Note that the BmWrErr interrupt is generated in such way that the corresponding read and write pointers are valid for failure analysis in standard mode. The interrupt generation is independent of the CanCONF.ABORT field setting.

48.15 Backwards compatibility between GRCAN and GRCANFD

GRCANFD is designed following the same approach as GRCAN, but adapting the register interface to the new frame formats and data lengths. This section summarizes the main differences introduced by GRCANFD from the user's point of view:

- While frames in GRCAN were always represented by a single descriptor, FD frames in GRCANFD may require from 1 to 5 descriptors in the circular buffers due to the larger data pay-

load. Whereas the first descriptor of a frame is still compatible, the descriptors 2 to 5 (if applicable) only contain data bytes and do not replicate the bits describing the format of the frame.

- The first descriptor of a frame includes the FDF and BRS bits. These are both set to 0b for classical CAN frames, so it is fully compatible with the memory representation of CAN frames in GRCAN, as these bits were unused.
- The Configuration Register (APB offset 0x000) does not include the CAN timing parameters anymore. Two new registers are created for this purpose: the Nominal Bit-Rate Configuration Register (0x040) and Data Bit-Rate Configuration Register (0x044).
- BPR (Baud Rate) and SAM (Triple Sampling) not supported by GRCANFD. In GRCAN these features may be enabled in the Configuration Register (0x000).
- New control bits to enable the generation of Overload Frames (Receive Channel Control Register, 0x300) and the internal/external loop-back modes (Configuration Register 0x000).
- New Capability Register (0x00C) providing information about the configuration of the generics when instantiating GRCANFD.
- New register for configuring the Transmitter Delay Compensation (0x048). This only applies to the data phase of FD frames (i.e. when switching the bit-rate).
- Since multiple descriptors may be necessary for describing a frame, the interrupts TxIRQ and RxIRQ are asserted when **any** of the descriptors of a transmitted/received frame matches the position programmed via the registers.
- Different device identifier: 0x03D for GRCAN; 0x0B5 for GRCANFD.
- New CANOpen mode. This mode uses a different set of registers. By default the IP operates in standard mode, which is the same general behavior as in GRCAN.

48.16 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x0B5 (GRCANFD). For description of vendor and device identifiers see GRLIB IP Library User's Manual.

48.17 Implementation

48.17.1 Reset

The core changes its reset behavior depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

GRCANFD will feature a synchronous reset if the parameter *grlib_async_reset_enable* is not set in the GRLIB configuration package. On the contrary, it will implement an asynchronous reset if *grlib_async_reset_enable* is set.

The reset is applied to all the registers in GRCANFD, except for the synchronization registers.

48.17.2 Endianness

The core automatically changes its endianness behavior depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for both big-endian and little-endian systems.

48.18 Configuration options

Table 640 shows the configuration options of the core (VHDL generics).

*Table 640.*Configuration options

Generic name	Function	Allowed range	Default
tech	Technology for the syncram and the sync. registers	0 - NTECH	inferred (0)
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar	0 - 16#FFF#	0
pmask	Mask field of the APB bar	0 - 16#FFF#	16#FFC#
pirq	Interrupt line used by the GRCANFD	0 - NAHBIRQ-1	1
singleirq	Implement only one common interrupt	0 - 1	0
txbufsize	Size of the internal TX buffer expressed in number of frames. The size can be calculated as *txbufsize* * 64 bytes.	1 - 8	2
rxbufsize	Size of the internal RX buffer expressed in number of frames. The size can be calculated as *rxbufsize* * 64 bytes.	1 - 8	2
ft	Enable fault-tolerance against SEU errors. The core can optionally be implemented with fault-tolerance against SEU errors in the FIFO memories by setting this generic to 1, 2, 4 or 5. Possible options are byte parity protection (ft = 1), TMR registers (ft = 2), SECDED BCH (ft = 4) or technology specific protection (ft = 5). If set to 0, no protection is implemented.	0 - 2, 4 - 5	0
scantest	Enable Scan Test support.	0 - 1	0
canopen	Implement a CANOpen target as part of the DMA layer. Enable the CANOpen-specific APB registers and the CANOpen mode. If set, certain registers are automatically configured during reset by sampling the configuration record cfg. This allows to establish communication with the CAN bus automatically without software intervention.	0 - 1	0
sepbus	Generate a separate bus master interface for the AMBA accesses in CANOpen mode. If set to 0, the standard and the CANOpen mode share the same bus master interface. Only valid if the generic *canopen* is also set to 1.	0 - 1	0
hindexcopen	AHB master index of the CANOpen bus. Only valid if both generics *canopen* and *sepbus* are set to 1.	0 - NAHBMST-1	0
ahbbits	Width of AHB read/write data buses and maximum access size	32, 64, 128	AHBDW

If *singleirq* is set to 1 there will be only one interrupt line, whose number will be the one set by *pirq*. Otherwise, if *singleirq* is set to 0, there will be three interrupt lines (Paragraph 38.8), whose numbers will be *pirq, pirq+1 and pirq+2*.

48.19 Signal descriptions

Table 641 shows the interface signals of the core when using the wrapper for AMBA AHB 2.0 (VHDL ports).

*Table 641.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBMI	*	Input	Array of AHB master input signals. The length of the array is 1 + canopen*sepbus.	-
AHBMO	*	Output	Array of AHB master output signals. The length of the array is 1 + canopen*sepbus.	-
CANI	Rx[1:0]	Input	Receive lines	-
CANO	Tx[1:0]	Output	Transmit lines	-
	En[1:0]		Transmit enables	-
CFG	en_codec	Input	Reset value of the ENABLE bit in the Control Register if the generic *canopen* is 1.	High
	line_sel		Reset value of the SELECT bit in the Configuration Register if the generic *canopen* is 1.	-
	en_out1		Reset value of the ENABLE1 bit in the Configuration Register if the generic *canopen* is 1.	-
	en_out0		Reset value of the ENABLE0 bit in the Configuration Register if the generic *canopen* is 1.	-
	nom_presc		Reset value of the SCALER field in the Nominal Bit-Rate Configuration Register if the generic *canopen* is 1.	-
	nom_ph1		Reset value of the PS1 field in the Nominal Bit-Rate Configuration Register if the generic *canopen* is 1.	-
	nom_ph2		Reset value of the PS2 field in the Nominal Bit-Rate Configuration Register if the generic *canopen* is 1.	-
	nom_sjw		Reset value of the SJW field in the Nominal Bit-Rate Configuration Register if the generic *canopen* is 1.	-
	en_canopen		Reset value of the EN bit in the CANOpen Control Register if the generic *canopen* is 1.	High
	node_id		Reset value of the ID field in the CANOpen Control Register if the generic *canopen* is 1.	-
MTESTI**	TXRAM	Input	Memory BIST input signal to TX RAM	-
	RXRAM	Input	Memory BIST input signal to RX RAM	-
MTESTO**	TXRAM	Output	Memory BIST output signal from TX RAM	-
	RXRAM	Output	Memory BIST output signal from RX RAM	-
MTESTCLK**	N/A	Input	Memory BIST clock	-

* see GRLIB IP Library User's Manual

** not available in FPGA releases

48.20 Signal definitions and reset values

The signals and their reset values are described in table 642.

*Table 642.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
cantx[]	Output	CAN transmit data	Low	Logical 1
canen[]	Output	CAN transmitter enable	High	Logical 0
canrx[]	Input	CAN receive data	Low	-

48.21 Library dependencies

Table 643 shows the libraries used when instantiating the core (VHDL libraries).

*Table 643.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GRLIB	GENERIC BM PKG	Signals, components	Package including the generic bus master interface and the bridges for converting to AHB and AXI
GAISLER	CANFD	Signals, component	GRCANFD component and signal declarations

48.22 Instantiation

This example shows how the core can be instantiated using the AMBA 2.0 AHB wrapper.

```
library  ieee;
use      ieee.std_logic_1164.all;
library  gaisler;
use      gaisler.canfd.all;

entity example is
   generic (
      padtech:       in    integer := 0);
   port (
      -- CAN interface
      cantx:         out   std_logic_vector(1 downto 0);
      canrx:         in    std_logic_vector(1 downto 0);
      canen:         out   std_logic_vector(1 downto 0);

...

   -- Signal declarations
   signal   rstn:          std_ulogic;
   signal   clk:           std_ulogic;

   signal   ahbmo:         ahb_mst_out_vector := (others => ahbm_none);
   signal   ahbmi:         ahb_mst_in_vector;

   signal   apbi:          apb_slv_in_type;
   signal   apbo:          apb_slv_out_vector := (others => apb_none);

   signal   cani0:         canfd_in_type;
   signal   cano0:         canfd_out_type;

...

   -- Component instantiation
canfd0 : grcanfd_ahb
    generic map(
      hindex         => 0,
```

```
    pindex          => 0,
    paddr           => 16#00C#,
    pmask           => 16#FFC#,
    pirq            => 1,
    singleirq       => 0)
  port map(
    clk             => clk0,
    rstn            => rstn0,
    ahbmi           => ahbmi0(0 downto 0),
    ahbmo           => ahbmo0(0 downto 0),
    apbi            => apbi0,
    apbo            => apbo0(0),
    cani            => cani0,
    cano            => cano0,
    cfg             => GRCANFD_CFG_NULL  -- Unused in this example
    );

cantx0_pad : outpad
   generic map (tech => padtech) port map (cantx(0), cani0.tx(0));

canrx0_pad : inpad
   generic map (tech => padtech) port map (canrx(0), cani0.rx(0));

canen0_pad : outpad
   generic map (tech => padtech) port map (canen(0), cani0.en(0));

cantx1_pad : outpad
   generic map (tech => padtech) port map (cantx(1), cani0.tx(1));

canrx1_pad : inpad
   generic map (tech => padtech) port map (canrx(1), cani0.rx(1));

canen1_pad : outpad
   generic map (tech => padtech) port map (canen(1), cani0.en(1));
```

49 GRCLKGATE / GRCLKGATE2X - Clock gating unit

49.1 Overview

The clock gating unit provides a means to save power by disabling the clock to unused functional blocks. The core provides a mechanism to automatically disabling the clock to LEON processors in power-down mode, and optionally also to disable the clock for floating-point units.

The core provides a register interface via its APB slave bus interface.

The clock gate unit has two main top-level units, GRCLKGATE and GRCLKGATE2X. GRCLKGATE2X exposes the full functionality and is the recommended top-level for new designs.

49.2 Operation

The operation of the clock gating unit is controlled through four registers: the unlock, clock enable, core reset and CPU/FPU override registers. The clock enable register defines if a clock is enabled or disabled. A '1' in a bit location will enable the corresponding clock, while a '0' will disable the clock. The core reset register allows to generate a reset signal for each generated clock. A reset will be generated as long as the corresponding bit is set to '1'. The bits in clock enable and core reset registers can only be written when the corresponding bit in the unlock register is 1. If a bit in the unlock register is 0, the corresponding bits in the clock enable and core reset registers cannot be written.

To gate the clock for a core, the following procedure should be applied:

1. Disable the core through software to make sure it does not initialize any AHB accesses
2. Write a 1 to the corresponding bit in the unlock register
3. Write a 0 to the corresponding bit in the clock enable register
4. Write a 0 to the corresponding bit in the unlock register

To enable the clock for a core, the following procedure should be applied

1. Write a 1 to the corresponding bit in the unlock register
2. Write a 1 to the corresponding bit in the core reset register
3. Write a 1 to the corresponding bit in the clock enable register
4. Write a 0 to the corresponding bit in the clock enable register
5. Write a 0 to the corresponding bit in the core reset register
6. Write a 1 to the corresponding bit in the clock enable register
7. Write a 0 to the corresponding bit in the unlock register

The clock gating unit also provides gating for the processor core and, optionally, floating-point units. A processor core will be automatically gated off when it enters power-down mode.

With the GRCLKGATE and GRCLKGATE2X units, any shared FPU will be gated off when all processor cores connected to the FPU have floating-point disabled or when all connected processor cores are in power-down mode.

With the GRCLKGATE2X unit it is also possible to support dedicated FPU clock gating. In this case a FPU will be gated off when processor core connected to the FPU has floating-point disabled or when the processor core is in power down mode.

Processor/FPU clock gating can be disabled by writing '1' to bit 0 of the CPU/FPU override register.

49.2.1 Shared FPU

For systems with shared FPU, a processor may be clock gated off while the connected FPU continues to be clocked. The power-down instruction may overtake a previously issued floating-point instruc-

tion and cause the processor to be gated off before the floating-point operation has completed. This can in turn lead to the processor not reacting to the completion of the floating-point operation and to a subsequent processor freeze after the processor wakes up and continues to wait for the completion of the floating-point operation.

In order to avoid this, software must make sure that all floating-point operations have completed before the processor enters power-down. This is generally not a problem in real-world applications as the power-down instruction is typically used in a idle loop and floating-point results have been stored to memory before entering the idle loop. To make sure that there are no floating-point operations pending, software should perform a store of the %fsr register before the power-down instruction.

49.3 Registers

The core's registers are mapped into APB address space.

Table 644. Clock gate unit registers

APB address offset	Register
0x00	Unlock register
0x04	Clock enable register
0x08	Core reset register
0x0C	CPU/FPU override register
0x10 - 0xFF	Reserved

49.3.1 Unlock register

*Table 645.*0x00 - UNLOCK - Unlock register

31 x+1	x 0
RESERVED	UNLOCK
0	0
r	rw

31: x+1 RESERVED

x: 0 Unlock clock enable and reset registers (UNLOCK) - The bits in clock enable and core reset registers can only be written when the corresponding bit in this field is 1.

49.3.2 Clock enable register

*Table 646.*0x04 - CLKEN - Clock enable register

31 x+1	x 0
RESERVED	ENABLE
0	*
r	rw

31: x+1 RESERVED

x: 0 Cock enable (ENABLE) - A ‘1’ in a bit location will enable the corresponding clock, while a ‘0’ will disable the clock.

49.3.3 Core reset register

Table 647. 0x08 - RESET - Reset register

31 x+1	x 0
RESERVED	RESET
0	0
r	rw

31: x+1 RESERVED

x: 0 Reset (RESET) - A reset will be generated as long as the corresponding bit is set to ‘1’.

49.3.4 CPU/FPU override register

Table 648. 0x0c - OVERRIDE - CPU/FPU override register

31 y+1	y 16	15 x+1	x 0
RESERVED	FOVERRIDE	RESERVED	OVERRIDE
0	0	0	0
r	rw	r	rw

31: y+1 RESERVED

y: 16 Override FPU clock gating (FOVERRIDE) - If bit n of this field is set to ’1’ then the clock for FPU n will be active regardless of the value of %PSR.EF. Only available if FPU clock is enabled at implementation.

15: x+1 RESERVED

x: 0 Override CPU clock gating (OVERRIDE) - If bit n of this field is set to ’1’ then the clock for processor n and FPU n will always be active.

49.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x02C. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

49.5 Implementation

49.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset for its internal registers.

49.5.2 Clock gate implementation

The clock gates are implemented using the CLKAND core in the techmap library, that instantiates the appropriate cell for the selected technology.

For ungated clocks, dummy clock gates are instantiated with the same technology but the isdummy generic set to 1. The technology mapping for the technology can decide whether to instantiate real clock gating cells for the technology or to pass the clock through as-is without any gating.

49.5.3 Scan test support

The test-enable signal is taken in through the APB input record and passed through to the techmap layer where it can be connected to the clock gating cell's test enable input or OR:ed into the normal enable if no test-enable input is available. Another alternative is to drive the cell's test enable input with constant 0 and hook it up to test-enable or scan-enable during the DFT implementation. Refer to the synthesis/DFT tools documentation for more details.

A separate ungate active-high input signal that also sets all clock gates to pass-through can be enabled in the core. This is passed in through the functional path.

49.5.4 Simulation

The underlying technology-specific gating in the techmap layer should ensure that all the gated and ungated clocks generated are delay and delta aligned to avoid zero-delay simulation problems. The standard solution is to add a 5 ps delay that gets removed on synthesis, however some technologies may use a different approach.

49.6 Configuration options

Table 649 shows the configuration options of the core (VHDL generics).

*Table 649.*Configuration options

Generic	Function	Allowed range	Default
tech	Clock/fabrication technology	0 to NTECH-1	0
pindex	Selects which APB select signal (PSEL) will be used to access the unit		
paddr	The 12-bit MSB APB address	0 to 16#FFF#	0
pmask	The APB address mask	0 to 16#FFF#	16#FFF#
ncpu	Number of processors that will connect to the unit	-	1
nclks	Number of peripheral units (clock/reset pairs) in addition to any processors and floating-point units that will connect to the unit.	0 - 31	8
emask	Bit mask where bit n (0 is the least significant bit) decides if a unit should be enabled (1) or disabled (0) after system reset.	0 - 16#FFFFFFFF#	0
extemask	If this generic is set to a non-zero value then the after-reset-enable-mask will be taken from the input signal epwen.	0 - 1	0
scantest	Enable scan test support	0 - 1	0

*Table 649.*Configuration options

Generic	Function	Allowed range	Default
edges	Extra clock edges provided by the clock gate unit after reset completes. CPUs get *edges* + 3 rising edges after reset and other cores get *edges* + 1 rising edges after system reset.	-	0
noinv	Do not use inverted clock for clock gate enable register. This generic can be set to one for technologies that have glitch free clock gates.	0 - 1	0
fpush	Selects FPU configuration 0: System has processors without, or with dedicated, FPUs 1: System has one FPU shared between all processors 3: System has one FPU for each parir of processors. (FPU0 is connected to CPU0 and CPU1, FPU1 is connected to CPU2 and CPU3, ...)	0 - 2	0
clk2xen	Enable double clocking. Only available on GRCLKGATE2X entity	0 - 1	1
ungateen	Enable separate ungate input for asynchronous un-gating of all clocks.	0 - 16#FFFFFFFF#	0
fpuclken	Enable separate clocks for FPU. Requires that generic *fpush* is set to 0. Only available on GRCLKGATE2X entity	0 - 1	0
nahbclk	Length of clkahb output vector Only available on GRCLKGATE2X entity	0 - 16#FFFFFFFF#	1
nahbclk2x	Length of clkahb2x output vector Only available on GRCLKGATE2X entity	0 - 16#FFFFFFFF#	1
balance	If balance is set to 1 then an always-enabled clock gate is inserted on each clkahb output. This option is obsolete as the techmap layer can now decide what to do with dummy clock gates, and only the value 1 is supported in the core.	1 - 1	1

49.7 Signal descriptions

Table 650 shows the interface signals of the core (VHDL ports).

*Table 650.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLKIN	N/A	Input	Clock	-
CLKIN2X	N/A	Input	Clock with higher frequency. Only present on GRCLKGATE2X entity.	-
PWD	N/A	Input	Power-down signal from processor cores	High
FPEN	N/A	Input	Floating-point enable signal from processor cores, only used in configurations with shared FPU when using the GRCLKGATE entity. For GRCLKGATE2X this input is also used when VHDL generic *fpuclken* is set to 1.	High
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
GCLK[nclks-1:0]	N/A	Output	Clock(s) to peripheral unit	-
RESET[nclks-1:0]	N/A	Output	Reset(s) to peripheral units	Low
CLKAHB[nahbclk**-1:0]	N/A	Output	Clock to non-gated units	-
CLKAHB2X[nahbclk2x**-1:0]	N/A	Output	2x Clock to non-gated units	-
CLKCPU[ncpu-1:0]	N/A	Output	Clock to processor cores	-
ENABLE[nclks-1:0]	N/A	Output	Enable signal(s) for peripheral units	High
CLKFPU[nfpu***:0]	N/A	Output	Clock to shared floating-point units, only used in configurations with shared FPU.	-
EPWEN	N/A	Input	External enable reset vector	High
UNGATE	N/A	Input	Ungate all clocks for test mode (only used if enabled in configuration)	High

* see GRLIB IP Library User's Manual

** Single output on GRCLKGATE entity, vector on GRCLKGATE2X entity.
*** where nfpu = (fpush/2)*(ncpu/2-1) for GRCLKGATE and (fpush/2+fpuclken)*(ncpu/(2-fpuclken)-1) for GRCLK-GATE2X

49.8 Library dependencies

Table 651 shows libraries used when instantiating the core (VHDL libraries).

*Table 651.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

49.9 Instantiation

This example shows how the core can be instantiated.

```
clkg0: grclkgate
      generic map (
        tech     => fabtech,
        pindex   => 4,
        paddr    => 16#040#,
        pmask    => 16#fff#,
        ncpu     => CFG_NCPU,
```

```
    nclks    => NCLKS,
    emask    => 0,                -- Don't care
    extemask => 1,                -- Reset value defined by input vector (epwen below)
    scantest => scantest,
    edges    => CG_EDGES,
    noinv    => CG_NOINV,
    fpush    => CFG_GRFPUSH)
  port map(
    rst      => rstn,      -- from reset generator
    clkin    => ahb_clk,   -- from clock generator
    pwd      => pwd,  -- from processors, typically dsuo.pwd(CFG_NCPU-1 downto 0)
    fpen     => fpen, -- from processors, if shared FPU is used
    apbi     => apbi,
    apbo     => apbo(4),
    gclk     => gclk,      -- clock to (gated) peripheral cores
    reset    => grst,      -- reset to (gated) peripheral cores
    clkahb   => clkm,      -- clock to AMBA system (not gated)
    clkcpu   => cpuclk,    -- clock to processor cores
    enable   => clkenable, -- enable(n) signals that peripheral n is enabled
    clkfpu   => fpuclk,    -- clock to any shared FPU cores
    epwen    => pwenmask,  -- signal to set enable-after-reset
    ungate   => gnd);
```

50 GRDMAC - DMA Controller with internal AHB/APB bridge

50.1 Overview

The GRDMAC core provides a flexible direct memory access controller. The core can perform burst transfers of data between AHB and APB peripherals at aligned or unaligned memory addresses. The core can be instantiated with one or two AHB master interfaces to perform transfers among different AHB buses.

The core's configuration registers are accessible through an APB interface. Up to 16 DMA channels are supported. Each channel can be configured flexibly by means of two descriptor chains residing in main memory: a Memory to Buffer (M2B) chain and a Buffer to Memory (B2M) chain. Each chain is composed of a linked list of descriptors, where each descriptor specifies an AHB address and the size of the data to read/write, supporting a scatter/gather behavior.

Once enabled, the core will proceed in reading the descriptor chains, then reading memory mapped addresses specified by the M2B chain and filling its internal buffer. It will then write the content of the buffer back to memory mapped addresses by elaborating the B2M descriptor chain.

The core supports a simplified mode of operation, with only one channel. In this mode of operation only one descriptor is present for each of the M2B and B2M chains. These two descriptors are written directly in the core's register via APB.

Note: This IP core is deprecated and should not be used for new designs. Please see the GRDMAC2 IP core instead.

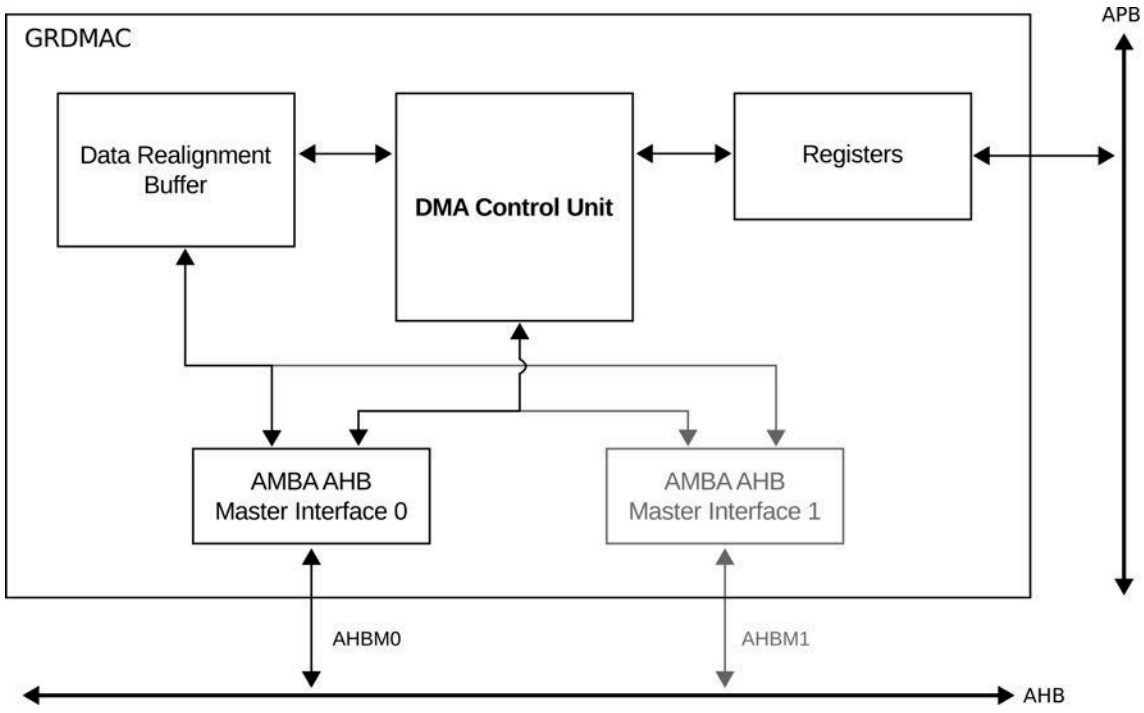


Figure 138. Block diagram

50.2 Configuration

The GRDMAC core consists of four main components: the DMA control unit, the AHB Master interface, the internal buffer with realignment support and an optional second AHB Master interface. The core supports being attached to any AHB bus with a data width of 32-bit, 64-bit or 128-bit. For every DMA channel the core will perform two types of DMA transfers through one of the AMBA AHB

Master interfaces: from memory to the internal buffer (M2B) and from the internal buffer to memory (B2M). The core will read data from memory until its internal buffer is filled or until the M2B descriptor chain is completed. When one of these two events is detected, GRDMAC will start writing the buffer content into memory, by switching to the B2M chain.

The internal buffer size is configurable through the generic *bufsize*. In case the buffer size is smaller that the total size of the M2B chain, the core will switch multiple times from the M2B chain to the B2M chain and vice versa.

The second AHB Master Interface is enabled by setting the *en_ahbm1* generic to 1. If the second interface is not enabled, all settings related to it will be ignored and the core will default to the main AHB Master Interface for all transfers.

50.2.1 Core setup

The GRDMAC core reads its configuration from any AHB mapped address (typically main memory) through its main AMBA AHB Master interface (AHBM0 if instantiated with support for two master interfaces). The core supports up to 16 DMA channels, a number configurable through the *ndmach* generic. For each channel, the M2B and B2M descriptor linked lists must be set up, and a pointer to the first descriptor in the two chains must be provided. These pointers are organized in a structure called Channel Vector. The Channel Vector is organized as in Table 652, below. For each of the GRDMAC channels there are two pointers: one pointer to the M2B descriptor linked list and one pointer to the B2M descriptor linked list. The Channel Vector array must be created at a 128-byte-aligned address. The GRDMAC core will read the Channel Vector entries for each channel up to *ndmach* channels.

*Table 652.*GRDMAC Channel Vector format

Address offset	Field
0x00	Channel 0: M2B descriptor pointer
0x04	Channel 0: B2M descriptor pointer
0x08	Channel 1: M2B descriptor pointer
0x0C	Channel 1: B2M descriptor pointer
...	...
...	...
0x78	Channel 15: M2B descriptor pointer
0x7C	Channel 15: B2M descriptor pointer

50.2.2 Descriptor types

Each descriptor consists of a four-field structure as provided in the tables below and must be created at a 16-byte-aligned address. There are three descriptor types: M2B descriptors, B2M descriptors and conditional descriptors.

The former two descriptors, categorized as data descriptors, are only allowed in the respective descriptor linked lists (M2B descriptor linked list and B2M descriptor linked list).

Conditional descriptors on the other hand, are required to be followed by a data descriptor, to which they bond to, and they can be specified in both the M2B and B2M descriptor linked lists. They are special descriptors that enable conditional behavior in a descriptor linked list and they are described in more detail in paragraph 50.3.2.

50.2.3 Data descriptors

For data descriptors, the first field, the next_descriptor field, is the address of the next descriptor in the chain. The chain ends with a descriptor whose next_descriptor field is all zeroes (NULL pointer).

The second field of an M2B descriptor, the address field, defines the address to read the data from. It can be any address in the system, and there are no alignment requirements. The number of bytes to transfer from memory to the internal buffer is specified in the third field, the control field, as seen in the table below.

*Table 653.*GRDMAC M2B descriptor format

Address offset	Field
0x0	M2B next_descriptor
0x4	M2B address
0x8	M2B control
0xC	M2B status

Table 654. GRDMAC M2B descriptor next_descriptor field (address offset 0x00)

31	4	3	1	0
NEXT_PTR		RESERVED		DT

31: 4 M2B Next descriptor pointer address (NEXT_PTR) - MSb of 16 Byte aligned address of the next descriptor in the M2B descriptor chain or NULL.

0 M2B descriptor type (DT) - Descriptor type field, ‘0’ for data descriptors, ‘1’ for conditional descriptors. Must be set to ‘0’ for this type of descriptor.

Table 655. GRDMAC M2B descriptor address field (address offset 0x04)

31	0
ADDR	

31: 0 M2B Address (ADDR) - Starting address the core will read data from.

Table 656. GRDMAC M2B descriptor control field (address offset 0x08)

31	16	15	5	4	3	2	1	0
SIZE		RESERVED		FA	AN	IE	WB	EN

31: 16 M2B descriptor size (SIZE) - Size in Bytes of the data that will be fetched from the address specified in the M2B address register.

4 M2B descriptor Fixed Address (FA) - If set to ‘1’, the data will be fetched from the same address for the entire size of the descriptor transfer. This is useful when reading from IO peripheral registers in combination with a conditional descriptor. If set to ‘0’, normal operation mode is attained.

3 M2B descriptor AHB Master Interface Number (AN) - If set to ‘0’, the descriptor’s transfer will be performed by the main AHB Master Interface (AHBM0). If set to ‘1’, the descriptor’s transfer will be performed by the second AHB Master Interface (AHBM1). If this interface is not enabled by the configuration generic *en_ahbm1*, then the transfer will fall back to the main AHB Master Interface (AHBM0).

2 M2B descriptor Interrupt Enable (IE) - If set to one, an interrupt will be generated when the M2B descriptor is completed. Descriptor interrupt generation also depends on interrupt mask for channel 0 and global interrupt enable.

Table 656. GRDMAC M2B descriptor control field (address offset 0x08)

1	M2B descriptor write-back (WB) - If set to one, the descriptor's status field will be written back in main memory after completion.
0	M2B descriptor Enable (EN) - If set to one, the descriptor will be enabled, otherwise it will be skipped and the next descriptor fetched from memory.

Table 657. GRDMAC M2B descriptor status field (address offset 0x0C)

31	3	2	1	0
RESERVED		E	S	C

2	M2B descriptor error (E) - If set to one, an error was generated during execution of the M2B descriptor. See error register for more information.
1	M2B descriptor status (S) - If set to one, the descriptor is being executed and running. Otherwise set to zero.
0	M2B descriptor completion (C) - If set to one, the descriptor was completed successfully.

For the B2M chain, the same holds true, with the exception of the address field, which specifies the address in main memory to write to.

*Table 658.*GRDMAC B2M descriptor format

Address offset	Field
0x0	B2M next_descriptor
0x4	B2M address
0x8	B2M control
0xC	B2M status

Table 659. GRDMAC B2M descriptor next_descriptor field (address offset 0x00)

31	4	3	1	0
NEXT_PTR		RESERVED		DT

31: 4	B2M Next descriptor pointer address (NEXT_PTR) - Address of the next descriptor in the B2M descriptor chain or NULL.
0	B2M descriptor type (DT) - Descriptor type field, '0' for data descriptors, '1' for conditional descriptors. Must be set to '0' for this type of descriptor.

Table 660. GRDMAC B2M descriptor address field (address offset 0x04)

31	0
ADDR	

31: 0	B2M Address (ADDR) - Starting address the core will write data to.

Table 661. GRDMAC B2M descriptor control field (address offset 0x08)

31	16	15	5	4	3	2	1	0
SIZE		RESERVED		FA	AN	IE	WB	EN

31: 16	B2M descriptor size (SIZE) - Size in Bytes of the data that will be written to the address specified in the B2M address register.

Table 661. GRDMAC B2M descriptor control field (address offset 0x08)

4	B2M descriptor Fixed Address (FA) - If set to '1', the data will be fetched from the same address for the entire size of the descriptor transfer. This is useful when writing to IO peripheral registers in combination with a conditional descriptor. If set to '0', normal operation mode is attained.
3	B2M descriptor AHB Master Interface Number (AN) - If set to '0', the descriptor's transfer will be performed by the main AHB Master Interface (AHBM0). If set to '1', the descriptor's transfer will be performed by the second AHB Master Interface (AHBM1). If this interface is not enabled by the configuration generic *en_ahbm1*, then the transfer will fall back to the main AHB Master Interface (AHBM0).
2	B2M descriptor Interrupt Enable (IE) - If set to one, an interrupt will be generated when the B2M descriptor is completed. Descriptor interrupt generation also depends on interrupt mask for channel 0 and global interrupt enable.
1	B2M descriptor write-back (WB) - If set to one, the descriptor's status field will be written back in main memory after completion.
0	B2M descriptor Enable (EN) - If set to one, the descriptor will be enabled, otherwise it will be skipped and the next descriptor fetched from memory.

Table 662. GRDMAC B2M descriptor status field (address offset 0x0C)

31	3	2	1	0
RESERVED		E	S	C

2	B2M descriptor error (E) - If set to one, an error was generated during execution of the B2M descriptor. See error register for more information.
1	B2M descriptor status (S) - If set to one, the descriptor is being executed and running. Otherwise set to zero.
0	B2M descriptor completion (C) - If set to one, the descriptor was completed successfully.

If a descriptor's write-back bit in its control field is set to one, the descriptor's status field will be written back to memory after completion. The transfer uses the AMBA AHB Master interface of the core.

50.2.4 Conditional descriptors

A conditional descriptor is a special kind of descriptor which bonds to a data descriptor and provides additional conditional behavior to it. A conditional descriptor can be used to create a DMA channel that retrieves data from IO cores, therefore off loading the CPU from the task. Usually IO cores provide a status register or an interrupt line to notify the CPU of the availability of new data. A conditional descriptor can be set up to poll this status register or to be triggered by an interrupt, signaling for instance, the availability of new data. Once data is available, the bond data descriptor is executed, accumulating the data in the internal buffer of the DMA core, before bursting it to memory for the software to handle it.

There are, hence, two kinds of conditional descriptors: polling conditional descriptors or triggering conditional descriptors. The former kind will continuously poll an address for data, and once a termination condition on the retrieved data is met, will yield to the data descriptor. The latter kind will instead have the core entering a state where it waits for a monitored input signal line to trigger. When the monitored input line is sampled to a value of '1', the data descriptor will be executed.

To set up a triggering conditional descriptor, the IT bit field in the descriptor's control field needs to be set to '1'. Bits 5:0 of the conditional address/triggering line field will specify which of the 64 input lines of the IRQ_TRIG signal will be monitored. During the execution of the triggering conditional descriptor, the triggering line is monitored every clock cycle, and when the value of the line is '1', the conditional execution will terminate and the data descriptor will be yield, fetching COND_SIZE bytes before going back to executing the conditional triggering. The data descriptor will be considered completed when all the bytes from the data descriptor, specified in the SIZE field, have been transfered, in amounts of COND_SIZE at each triggering. If the *timer_en* VHDL configuration generic is set to '1', an optional timeout counter can be enabled during the triggering conditional descriptor execution. By

setting the TE bit field in the core's control register to '1' and by setting the Timer Reset Value Register to the required number of clock cycles, the descriptor execution is halted with a Timeout Error if an interrupt is not received before the timer expires. The error halts the channel execution after eventual descriptor write-back is performed.

To set up a polling conditional descriptor, the IT bit field in the descriptor's control field needs to be set to '0'. Bits 31:0 of the conditional address/triggering line field will point to the address that the DMA core will poll for data until the termination condition is TRUE. The condition is specified as the bitwise AND between the 32-bit word pointed by COND_ADDR and the COND_MASK. This value is compared to 0 according to the following formulas, according to the termination condition type selected in the conditional control field (CT).

Table 663. GRDMAC Conditional descriptor Termination condition type 0

$$(*\text{COND_ADDR} \wedge \text{COND_MASK}) = 0$$

Table 664. GRDMAC Conditional descriptor Termination condition type 1

$$*\text{COND_ADDR} \wedge \text{COND_MASK} \neq 0$$

When the condition is TRUE, the conditional descriptor will stop polling and will proceed with fetching COND_SIZE bytes from the data descriptor pointed by NEXT_PTR. The behavior of conditional descriptors is explained in depth in paragraph 50.3.2.

Also in paragraph 50.3.2 is an example configuration of a conditional DMA channel for UART reading.

*Table 665.*GRDMAC Conditional descriptor format

Address offset	Field
0x0	Conditional next_descriptor
0x4	Conditional address/triggering line
0x8	Conditional control
0xC	Conditional mask

Table 666. GRDMAC Conditional descriptor next_descriptor field (address offset 0x00)

31	4	3	1	0
NEXT_PTR		RESERVED		DT

31: 4	Conditional Next descriptor pointer address (NEXT_PTR) - Address of the data descriptor in the descriptor chain which the conditional descriptor is bond to. Cannot be NULL.
0	Conditional descriptor type (DT) - Descriptor type field, '0' for data descriptors, '1' for conditional descriptors. Must be set to '1' for this type of descriptor.

Table 667. GRDMAC Conditional descriptor address field (address offset 0x04)

31	6	5	0
COND_ADDR[31:6]		COND_ADDR[5:0] / IRQN	

Table 667. GRDMAC Conditional descriptor address field (address offset 0x04)

31: 0	Conditional Address (COND_ADDR) - Address of the 32-bit word the core will read for the conditional termination expression matching.
5: 0	IRQ Trigger Line Number (IRQN) - Index of the IRQ_TRIG signal input vector which is used as the triggering line for triggered conditional descriptors, 0 to 63.

Table 668. GRDMAC Conditional descriptor control field (address offset 0x08)

31 16	15 4	3	2	1	0
COND_SIZE	COUNTER_RST	AN	CT	IT	EN

31: 16	Conditional descriptor total size (COND_SIZE) - Total size in Bytes of the data that will be fetched from the bond data descriptor each time the conditional termination expression matches to true.
15: 4	Conditional descriptor counter reset value (COUNTER_RST) - Reset value of the conditional counter timer that is executed before every polling or triggering. The unit is number of clock cycles and the purpose is to provide a timer between polling requests onto the AMBA AHB bus with enough clock cycles in order not to clog the bus.
3	Conditional descriptor AHB Master Interface Number (AN) - If set to ‘0’, the descriptor’s transfer will be performed by the main AHB Master Interface (AHBM0). If set to ‘1’, the descriptor’s transfer will be performed by the second AHB Master Interface (AHBM1). If this interface is not enabled by the configuration generic *en_ahbm1*, then the transfer will fall back to the main AHB Master Interface (AHBM0).
2	Conditional descriptor Termination Condition type (CT) - If the conditional descriptor is of type “polling”, this bits specifies which type of termination condition is used. If ‘0’, the termination condition is of type 0 as specified in this paragraph. If ‘1’, the termination condition is of type 1.
1	Conditional Descriptor Irq Trigger (IT) - If set to ‘1’, the conditional descriptor will wait for the input interrupt line to go high before executing the bond data descriptor. The selected interrupt line is the one indexed by IRQN in the IRQ_TRIG signal input vector. This bit enables triggering behavior of conditional descriptors. If this bit is set to ‘0’, normal polling behavior with termination condition is enabled.
0	Conditional descriptor Enable (EN) - If set to one, the descriptor will be enabled, otherwise it will be skipped and the next descriptor fetched from memory.

Table 669. GRDMAC Conditional descriptor mask field (address offset 0x0C)

31 0
COND_MASK

31: 0	Conditional Mask (COND_MASK) - Bit mask used in the conditional descriptor termination condition matching.

50.2.5 Register setup

Once the channel vector and the relative descriptor chains are setup in main memory, the GRDMAC register must be also setup. The 128-byte-aligned address, where the Channel Vector resides, must be written in the Channel Vector Pointer register. The control register must also be setup. Once the enable bit of the control register is set to one, the core will start running and will execute all the channels which are enabled.

50.3 Operation

50.3.1 Normal mode of operation

In normal mode of execution, GRDMAC will start executing all the enabled channels until they are complete or an error is generated.

When executing a DMA channel, the core will initially fetch the two descriptor pointers from the address provided in the CVP register which are relative to the channel. It will then fetch the first M2B and B2M descriptors from main memory. The M2B descriptor chain is then executed until either the internal buffer is full, or the M2B chain is completed. If one of this events happen, the core will switch to the B2M descriptor chain. The B2M chain will switch back to the M2B chain when the buffer is empty. The DMA channel is marked complete when the last descriptor in the B2M chain is executed, finally emptying the buffer.

During the execution of a chain, the core will fetch a new descriptor after the successful completion of the previous one, following the pointers in the linked list. When the core reaches a NULL pointer in the M2B chain, it will switch to the B2M chain. When it reaches a NULL pointer in the B2M chain, the core will update the DMA channel status and switch to the next enabled DMA channel, until all the channels are completed.

50.3.2 Operation with conditional descriptors

Conditional descriptors bond to the following data descriptor in the linked list and provide conditional behavior to the execution of the data descriptor. During the execution of a DMA channel, when the core fetches a conditional descriptor from memory, it will proceed and fetch the following descriptor in the chain as well, which must be a data descriptor.

After the descriptors' pair has been fetched, the conditional execution will follow these steps:

a) the core will execute the conditional counter, down counting for COUNTER_RST clock cycles

b) if the conditional descriptor is a polling descriptor, go to step **c1**, if it's a triggering descriptor, go to step **c2**.

c1) the core will fetch a 32-bit word at the COND_ADDR address.

d1) if the conditional termination condition of Table 664 is *false* then the core will go back to step **a,** if the conditional termination condition of Table 664 is *true*, the core will fetch a portion of the data from the data descriptor which is COND_SIZE bytes, then go back to step **a**.

c2) the core will monitor line IRQN of the IRQ_TRIG input signal, indefinitely or until the trigger counter expires.

d2) when the monitored line has a value of '1', the core will fetch a portion of the data from the data descriptor which is COND_SIZE bytes, then go back to step **a**.

The total SIZE of the bond data descriptor will be decremented by COND_SIZE bytes every time the bond data descriptor is executed, and the ADDRESS will be incremented by the same amount (unless the FA flag is set).

The FA (Fixed Address) bit field in the data descriptor control field is useful when accessing data to/from a peripheral data register, i.e. UART data register, when you need to read/write always from/to the same address.

The execution of the descriptor pair (conditional and bond data descriptors) ends when the SIZE field of the data descriptor reaches 0. In other words, the execution ends when SIZE bytes have been fetched in total from the data descriptor, by fetching COND_SIZE byte amounts every time the conditional condition (polling or triggering) is true.

50.3.3 Simplified mode of operation

In Simplified Mode of Operation, the GRDMAC core configuration resides entirely in its configuration registers and the Channel Vector structure is not used. The core will not perform any memory access to fetch configuration data. This mode of operation makes use of only two data descriptors, respectively one descriptor for M2B transfers and one for B2M transfers. Conditional descriptors are not supported in this mode. The descriptors are written directly onto GRDMAC via APB at offsets 0x20 and 0x30. Their next_descriptor field is hardwired to zeroes. Their status is always written-back to their relative descriptor status register.

When the core is configured in Simplified mode of operation, the relative bit (SM) must be set to one in the control register. The core will execute the two internal descriptors on channel zero. Channel zero must therefore be enabled, and the core status can be read on channel zero's status bits in the status register.

50.4 AHB transfers

For every descriptor executed, GRDMAC will perform an AHB data transfer at the address and of the size specified. The AHB accesses that it can perform are up to 128-bit wide and can be at aligned or unaligned memory addresses. The maximum AHB access width depends on the AHB bus width and on the *busw* and *wbmask* generics, as described in paragraph 50.7.

The core will perform unaligned memory access if defined by the descriptors. It will perform byte (8 bit) accesses at byte-aligned addresses, half-word (16 bit) accesses at half-word aligned addresses, and so on. The core will perform burst transfers of the maximum supported width for as long as possible according to the total transfer size. For example, if the maximum supported bus width for one transfer is 64-bit, and a descriptor requests 18 bytes at address 0x40000006, the core will perform one 16-bit half-word access, and one two-beat burst of double words. In some cases, the total transfer size might require GRDMAC to perform additional word, half-word and/or byte accesses at the end of the transfer. The burst accesses performed by GRDMAC are of type incrementing burst of unspecified length. These bursts will never cross a 1KB memory boundary, or a smaller boundary that can be set with the generic *burstbound*. At the specified memory boundary set by *burstbound*, the burst will be interrupted, an idle cycle will be inserted and the incrementing burst of unspecified length will restart from the next address. This generic can be used to limit the maximum burst length performed by the core, making re-arbitration on the AHB bus more frequent.

When the core is configured with the VHDL generic *en_ahbm1* set to '1', a secondary AHB Master interface will be instantiated inside the GRDMAC core. This interface can be connected to a second AHB bus to provide bridging capabilities to the DMA controller. The core will fetch data from this interface when the AN flag in the descriptor's control field is set accordingly. This flag will be ignored in case the core is configured with the *en_ahbm1* VHDL generic set to '0'.

50.5 Data realignment buffer

The realignment buffer is the data buffer used internally by the GRDMAC core. The component allows the core to store the data in a tightly packed way, being optimized to store AMBA AHB transfer data of different size and at different address offsets.

The internal buffer uses RAM implemented using GRLIB parameterizable SYNCRAMBW memories, assuring portability to all supported technologies. Internally two SYNCRAMBW are used, one for even words and one for odd words. The total number of RAMs used depends on the *bufsize* generic, and its minimum size is two words, 8 bytes. To control the implementation technology of the internal RAMs, the technology mapping generic *memtech* may be used. Additionally, the generic *testen* will be propagated to the SYNCRAMBW and is used to enable scan test support.

Fault tolerance can be added to the RAM by setting the *ft* generic to a value different than 0. To obtain byte parity DMR memories, set the *ft* generic to 1. To use TMR set it to 2. Note that the *ft* generic needs to be set to 0 unless the core is used together with the fault tolerant version of GRLIB, which is not available under the terms of the GPL.

50.6 Interrupts

GRDMAC provides fine-grained control of interrupt generation. At the highest level, the global Interrupt Enable bit (IE) in the control register can be set to zero to mask every interrupt setting in the core. If set to one, interrupt generation depends on the following settings.

The Interrupt on Error Enable bit (IEE) in the control register provides a way to generate interrupts in the event of errors. Error generation is discussed further in the next paragraph.

An interrupt can be also generated by the successful completion of a descriptor, if the Interrupt Enable (IE) bit is set to one in the descriptor's control field. The Interrupt Mask bit (Ix) in the Interrupt Mask register can be set to zero to mask all the descriptor completion interrupts. If descriptor write-back is enabled, the interrupt will be generated after writing back the descriptor's status in main memory.

For both interrupts on error and interrupts on descriptor completion events, a flag will be raised in the interrupt flag register at the bit corresponding to the channel where the interrupt event happened (IFx).

As an example of interrupt generation setup, one can enable interrupt on channel completion by performing the following steps. The Interrupt Enable (IE) bit in GRDMAC control register must be set to one, as must be the relevant channel's interrupt mask bit in the Interrupt mask register. Finally the Interrupt Enable (IE) bit in the control field of the last descriptor in the B2M chain of the channel must be set to one, while the same field must be set to zero in every other descriptor in the channel. This way, when the last descriptor in the buffer to memory chain is completed successfully, an interrupt will be generated.

50.7 Wide Data Bus support

The size of AMBA accesses supported through GRDMAC's AHB master interfaces depends on the maximum bus width and if the accessed memory area has been marked as being on the wide bus.

The generic *wbmask* is treated as a 16 bit mask where every bit represents a 256 MiB block of memory, with the least significative bit representing the range 0 - 0x10000000. If the corresponding bit is set to one, GRDMAC with perform wide accesses to that memory area. The size of the accesses is controlled with the *busw* generic. If the generic is set to 0, only 32 bit accesses will be performed.

Furthermore, the size of the AHB accesses can be limited with the Transfer Size Limit (TSL) field in the control register of GRDMAC. If the field is set to 1, the core will limit its maximum AHB transfer size to 32 bits. If it is set to 2, the limit will be 64 bits, and if it is set to 3, the limit will be 128 bits. The field must be interpreted as an upper limit on the transfer size and is subject to the *wbmask* and *busw* generic.

50.8 Errors

Four types of errors can be generated by GRDMAC. Transfer errors, descriptor errors, Channel Vector Pointer errors, conditional errors and timeout errors, as defined in the Error Register.

Transfer errors are generated when the core is accessing DMA data from and to memory and it encounters an AMBA AHB ERROR response. When a transfer error occurs on a descriptor which has the write-back flag enabled, the descriptor status will be written back to main memory with the error field set to one. An eventual interrupt will be generated only after the write back.

Descriptor errors are generated when an ERROR response is received while reading or writing back a descriptor in main memory.

Channel Vector Pointer errors are generated when the core receives an ERROR response when accessing the Channel Vector data structure in main memory.

Conditional errors are generated when a conditional polling descriptor encounters a problem during an AHB polling operation such as an ERROR response.

Finally timeout errors are caused by the timeout counter expiring before receiving an interrupt during triggered conditional descriptor execution. This requires the *timer_en* VHDL configuration generic to be set to '1' and the TE bit field in the control register to be configured to '1' during execution.

The core will enable the corresponding error type bit in the error register in addition to the error flag bit (E). The channel number where the error happened can be also read directly from the channel error field (CHERR) of the error register. Additionally an interrupt will be generated if the Interrupt on Error Enable bit (IEE) and the global Interrupt Enable (IE) bit in GRDMAC control register are set to one, and a flag will be raised in the interrupt flag register bit corresponding to the channel where the error event occurred (IFx).

50.9 Internal Buffer Readout Interface

In case of an error, the execution of the DMA channels will halt and the error will be reported as described in the previous session. It can happen that data that has been accumulated in the internal buffer during the M2B chain transactions, is not written out as part of the B2M chain, due to the channel halting. This internal data can still be read via the APB interface of the GRDMAC core, through the Internal Buffer Readout Interface memory area. The memory area is located at offset 0x800 to 0xFFF of the GRDMAC core memory address, totaling 2 KiB of accessible Internal Buffer space, as seen in Table 670. This area can only be read when the core is in an idle state and bit flag EN of the Control Register is set to '0'. The amount of valid data in the internal buffer can be inferred by reading the read pointer and write pointers to the buffer from the Internal Buffer Pointers Register (offset 0x40).

50.10 Endianness

The core is designed for big-endian systems.

50.11 Registers

The core is programmed through registers mapped into APB address space. The APB address is configured with the *paddr* and *pmask* generics. If the core is instantiated with the internal AHB/APB bridge, the *haddr* and *hmask* generics will configure the APB address space.

*Table 670.*GRDMAC controller registers

APB address offset	Register
0x00	Control register
0x04	Status register
0x08	Interrupt mask register
0x0C	Error register
0x10	Channel Vector Pointer
0x14	Timer Reset Value register
0x18	Capability register
0x1C	Interrupt flag register
0x20	Reserved
0x24	M2B Descriptor Address register*
0x28	M2B Descriptor Control register*
0x2C	M2B Descriptor Status register*
0x30	Reserved
0x34	B2M Descriptor Address register*
0x38	B2M Descriptor Control register*
0x3C	B2M Descriptor Status register*
0x40	Internal Buffer Pointers Register
0x800-0xFFF	Internal Buffer Readout Area

*Only used in Simplified Mode of Operation

50.11.1 Control Register

*Table 671.*GRDMAC control register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15 12	11 6	5	4	3	2	1	0
EF	EE	ED	EC	EB	EA	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0	TSL	RESERVED	TE	SM	IEE	IE	RS	EN
NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR		0	NR	NR	NR	0	0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		*	rw	rw	rw	rw	rw

31: 16 Enable channel x (Ex) - Set to one to enable DMA channel x, from 0 to 15.

15 12 Transfer Size limit (TSL) - If set to 1, the GRDMAC core will limit its maximum transfer size to 32b accesses. If set to 2, it will limit the transfer size to 64 bits. If set to 3, it will limit the maximum transfer size to 128 bit. If set to 0 no limit is imposed. The maximum transfer size is controlled by the *wbmask* and *busw* generics.

5 Timer Enable (TE) - Set to '1' to enable the timeout timer during triggered conditional descriptor execution. If the *timer_en* generic is set to '1', the field is rw, read-only otherwise.

4 Simplified mode (SM) - Set to one to use the core in simplified mode of operation

3 Interrupt enable for Errors (IEE) - Set to one to enable interrupt generation on error. Interrupt generation on error depends on the global Interrupt Enable (IE).

2 Interrupt Enable (IE) - Global Interrupt Enable. If set to zero, no interrupt will be generated. If set to one, interrupts from errors, descriptor completion, won't be masked.

1 Reset (RS) - Resets the core register if set to one.

0 Enable/Run (EN) - When set to one, the core will be enabled and start running.

50.11.2 Status Register

*Table 672.*GRDMAC status register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SF	SE	SD	SC	SB	SA	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0	CF	CE	CD	CC	CB	CA	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

31: 16 Status of channel x (Sx) - Set to one if DMA channel x is running, set to zero otherwise.

15: 0 Completion of channel x(Cx) - Set to one if DMA channel x has completed successfully, zero otherwise.

50.11.3 Interrupt Mask

*Table 673.*GRDMAC Interrupt Mask

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																IF	IE	ID	IC	IB	IA	I9	I8	I7	I6	I5	I4	I3	I2	I1	I0
																NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR
																rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15: 0 Interrupt Mask for channel x (Ix) - Set to 0 to mask descriptor interrupt generation from channel x. Interrupt generation depends on the global Interrupt Enable in the control register.

50.11.4 Error Register

*Table 674.*GRDMAC error register

31 20	19 16	15 6	5	4	3	2	1	0
RESERVED	CHERR	RESERVED	ME	OE	TE	DE	CE	E
			0	0	0	0	0	0
	r		wc	wc	wc	wc	wc	wc

19: 16 Channel error (CHERR) - Channel number where last error was generated.

5 Timeout Error (ME) - One if the last generated error was of type timeout error. This field is cleared by writing a one to it.

4 Conditional Error (OE) - One if the last generated error was of type conditional execution error. This field is cleared by writing a one to it.

3 Transfer Error (TE) - One if the last generated error was of type transfer error. This field is cleared by writing a one to it.

2 Descriptor Error (DE) - One if the last generated error was of type descriptor error. This field is cleared by writing a one to it.

1 CVP Error (CE) - One if the last generated error was of type CVP error. This field is cleared by writing a one to it.

0 Error (E) - If set to one, an error was generated by the entity. This field is cleared by writing a one to it.

50.11.5 Channel Vector Pointer

*Table 675.*GRDMAC Channel Vector Pointer

31 7	6 0
CVP	RESERVED
NR	
rw	

31: 7 Channel Vector Pointer (CVP) - 128 Byte aligned memory address pointing to the vector of up to 16 couples of descriptor chain pointers.

50.11.6 Timer Reset Value Register

*Table 676.*GRDMAC Timer Reset Value Register

31 0
TIMER_RST
0x00000000
*

31: 0 Timer Reset Value (TIMER_RST) - Reset value for the triggered conditional descriptor timeout timer. If the *timer_en* generic is set to '1', the field is rw, read-only otherwise.

50.11.7 Capability Register

*Table 677.*GRDMAC capability register

31 16	15 12	11	10 9	8	7 4	3 0
BUFSZ	RESERVED	TT	FT	H1	NCH	VER
*		*	*	*	*	*
r		r	r	r	r	r

31: 16 Buffer size (BUFSZ) - Binary logarithm of the internal buffer size of the entity.

11 Timer (TT) - If set to '1', the timeout timer is enabled.

Table 677. GRDMAC capability register

10: 9	Fault Tolerant buffer (FT) - These bits indicate if the internal buffers in the core is implemented with fault tolerance. When 0, no fault tolerance, when 1, byte parity DMR, when 2, TMR. Reflects the VHDL generic ft.
8	Second AHB Master (H1) - If set to one, the second AHB master interface (AHBM1) is enabled.
7: 4	Channel Number (NCH) - The maximum number of supported DMA channels in the core is NCH+1.
3: 0	Version (VER) - GRDMAC version number.

50.11.8 Interrupt Flag Register

*Table 678.*GRDMAC interrupt flag register

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED		IFF	IFE	IFD	IFC	IFB	IFA	IF9	IF8	IF7	IF6	IF5	IF4	IF3	IF2	IF1	IF0
		NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15: 0 Interrupt flag for channel x (IFx) - When set to one, an interrupt event (descriptor completion or error) was generated on channel x. This field is cleared by writing a one to it.

50.11.9 M2B Descriptor Address Register*

*Table 679.*GRDMAC M2B descriptor address register*

31	0
ADDR	
NR	
rw	

31: 0 M2B Address (ADDR) - Starting address the core will read data from.

50.11.10M2B Descriptor Control Register*

*Table 680.*GRDMAC M2B descriptor control register*

31	16	15	3	2	1	0
SIZE		RESERVED		IE	R	EN
NR				NR		NR
rw				rw		rw*

31: 16 M2B descriptor size (SIZE) - Size in Bytes of the data that will be fetched from the address specified in the M2B address register.

2 M2B descriptor Interrupt Enable (IE) - If set to one, an interrupt will be generated when the M2B descriptor is completed. Descriptor interrupt generation also depends on interrupt mask for channel 0 and global interrupt enable.

0 M2B descriptor Enable (EN) - Set to one when the descriptor is written the first time. Write value ignored.

50.11.11 M2B Descriptor Status Register*

*Table 681.*GRDMAC M2B descriptor status register*

31	3	2	1	0
RESERVED		E	S	C
		0	0	0
		rw	rw	rw

2 M2B descriptor error - If set to one, an error was generated during execution of the M2B descriptor. See error register for more information.

1 M2B descriptor status (S) - If set to one, the descriptor is being executed and running. Otherwise set to zero.

0 M2B descriptor completion (C) - If set to one, the descriptor was completed successfully.

50.11.12 B2M Descriptor Address Register*

*Table 682.*GRDMAC B2M descriptor address register*

31	0
ADDR	

*Table 682.*GRDMAC B2M descriptor address register*

NR
rw

31: 0 B2M Address (ADDR) - Starting address the core will write data to.

50.11.13 B2M Descriptor Control Register*

*Table 683.*GRDMAC B2M descriptor control register*

31 16	15 3	2	1	0
SIZE	RESERVED	IE	R	EN
NR		NR		NR
rw		rw		rw*

31: 16 B2M descriptor size (SIZE) - Size in Bytes of the data that will be written to the address specified in the B2M address register.

2 B2M descriptor Interrupt Enable (IE) - If set to one, an interrupt will be generated when the B2M descriptor is completed. Descriptor interrupt generation also depends on interrupt mask for channel 0 and global interrupt enable.

0 B2M descriptor Enable (EN) - Set to one when the descriptor is written the first time. Write value ignored.

50.11.14 B2M Descriptor Status Register*

*Table 684.*GRDMAC B2M descriptor status register*

31 3	2	1	0
RESERVED	E	S	C
	0	0	0
	rw	rw	rw

2 B2M descriptor error - If set to one, an error was generated during execution of the B2M descriptor. See error register for more information.

1 B2M descriptor status (S) - If set to one, the descriptor is being executed and running. Otherwise set to zero.

0 B2M descriptor completion (C) - If set to one, the descriptor was completed successfully.

50.11.15 Internal Buffer Pointers Register

*Table 685.*GRDMAC internal buffer pointers register*

31 16	15 0
READ_P	WRITE_P
0	0
r	r

31: 16 B2M Internal Buffer Read Pointer (READ_P)- Points to the last offset in the internal buffer which was correctly read by the core and output on the bus.

15: 0 B2M Internal Buffer Write Pointer (WRITE_P) - Points to the last offset in the internal buffer which was correctly written by the core as an input from the bus.

*Register used only when the core is set to work in Simplified mode of operation.

50.12 Example DMA channel set-up

In this example a single DMA channel will be set-up, using conditional descriptors, to gather data from the UART core (APBUART) and write it into main memory.

The GRDMAC core is configured with its register address-space starting at address 0xCCC00200 and main memory starts at 0x40000000. The APBUART core's register is mapped at 0xCCC00100 and the UART receiver FIFO queue is configured as 4 bytes.

The DMA channel will need two descriptors in the M2B chain: a conditional descriptor bound to a data descriptor. The B2M chain will only need one data descriptor.

The conditional descriptor will poll the UART status register, mapped at 0xCCC00104, and will use the mask 0x00000100 for the termination condition. This mask will be ANDed with the status register, and the result of this operation will only show the value of the "Receiver FIFO half-full" field in the status register. This will enable the conditional register to stop polling when this bit becomes '1'. At this point the data descriptor will be executed for the amount of bytes specified in the conditional descriptor, which in this case is 1 bytes (half of the FIFO size). For the data transfer to read and accumulate correct data, the core must perform a single-byte access. The UART data register contains only one byte of relevant data. The size limit per transfer is therefore 1 byte and the address is marked as fixed, so the core will not increment it after every transfer.

The polling counter for the conditional descriptor is set according to the UART speed. If the UART baud rate is 38.4K and the system frequency is 100 MHz, one can assume that there is going to be 1 Byte available in the UART every 26k clock cycles. Setting the polling period to a value less than 26K will let the DMA get all the characters from the UART without missing any. The conditional counter reset value is set to its maximum, a period of 4095 clock cycles (0xFFF).

The polling will restart after the last read and the transfers will go on until the total size specified in the SIZE field of the data descriptor is reached. At this point the M2B chain is completed and the core will proceed with the B2M chain, emptying the contents of its buffer into memory, at the address specified.

Table 686 shows the memory layout of the system with the required data to set-up this example. Note that the Channel Vector is 128-byte aligned and the descriptors are 16-byte aligned,

Table 686. Memory Content

Address	Data	Description
0x40000080	0x40020010	Channel Vector - Channel 0 M2B descriptor chain pointer
0x40000084	0x40020040	Channel Vector - Channel 0 B2M descriptor chain pointer
...	...	
0x40020010	0x40020031	M2B conditional descriptor 0 - next descriptor pointer (lsb set to 1 for cond. desc.)
0x40020014	0xCCC00104	M2B conditional descriptor 0 - address (UART status register address)
0x40020018	0x0001FFF1	M2B conditional descriptor 0 - control (poll every 4095 cycles, get 1 Byte)
0x4002001C	0x00000080	M2B conditional descriptor 0 - mask (only check “Receiver FIFO half-full”)
...	...	
0x40020030	0x00000000	M2B data descriptor 0 - next descriptor pointer (NULL, end of chain)
0x40020034	0xCCC00100	M2B data descriptor 0 - address (UART data register address)
0x40020038	0x04000011	M2B data descriptor 0 - control (1024 Bytes from fixed address)
0x4002003C	-	M2B data descriptor 0 - status (written by core)
...	...	
0x40020040	0x00000000	B2M data descriptor 0 - next descriptor pointer (NULL, end of chain)
0x40020044	0x40030000	B2M data descriptor 0 - address (DMA write address for UART data)
0x40020048	0x04000001	B2M data descriptor 0 - control (1024 Bytes)
0x4002004C	-	B2M data descriptor 0 - status (written by core)
...	...	
0x40030000	-	UART data written by the DMA core
...	...	
0xCCC00200	0x0001000C	GRDMAC Control register
0xCCC00204	-	GRDMAC Status register (updated by the DMA core)
0xCCC00208	0x00000001	GRDMAC interrupt mask register
0xCCC0020C	-	GRDMAC error register (updated by the DMA core)
0xCCC00200	0x40000080	GRDMAC channel vector pointer
0xCCC00204	-	Reserved
0xCCC00208	0x02000812	GRDMAC capability register
0xCCC0020C	-	GRDMAC interrupt flag register (updated by the DMA core)

as required by the core.

The core is configured with only one DMA channel (channel 0) and one master interface, as can be seen in the capability register. Additionally the internal core’s buffer is 512 Bytes and the time-out timer is available.

The core’s control register is pre-set to enable channel 0 and to enable interrupts and interrupts on errors. To start the execution of the channel the software will write a ‘1’ to the enable bit in the control register, usually by reading the register, performing a logical OR with 0x00000001, and writing the value back to the register. In this case the value that needs to be written to address 0xCCC00200 to correctly start execution is 0x0001000D.

50.13 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x095. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

50.14 Implementation

50.14.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User’s Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

50.14.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

50.15 Configuration options

Table 687 shows the configuration options of the GRDMAC core (VHDL generics). These options are specific to the generic entity *grdmac*. See the chapter Instantiation for more details on specialized versions of GRDMAC.

*Table 687.*Configuration options

Generic	Function	Allowed range	Default
hmindex	AHB master index (AHBM0)	0 - NAHBMST-1	0
hirq	IRQ line used by GRDMAC	0 - NAHBIRQ-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddress	Addr field of the APB bar.	0 - 16#FFF#	1
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
en_ahbm1	Enable second AHB master interface (AHBM1)	0 - NAHBMST-1	0
hmindex1	Second AHB master (AHBM1) index	0 - NAHBMST-1	1
ndmach	Number of available DMA channels.	1 - 16	1
bufsize	Internal buffer size. Must be a power of 2.	8 - 65536	256
burstbound	Boundary that the burst will never cross. Maximum is 1KB as per the AMBA AHB standard. Could be set to smaller values to ease re-arbitration. Must be a multiple of 2.	4 - 1024	512
timer_en	Enables the implementation of the timeout counter which can be set during triggered conditional descriptor execution.	0 - 1	0
memtech	Internal buffer's memory technology selection	0 - NTECH	0
testen	Enable bypass logic for scan testing	0 - 1	0
ft	This generic determines if fault tolerance should be added to the internal data realignment buffer. 0 = no fault tolerance, 1 = byte parity DMR, 2 = TMR. Note that this generic needs to be set to 0 if the core is used together with the GPL version of GRLIB, since that version does not include any fault tolerance capability.	0 - 2	0
wbmask	Wide-bus mask. Indicates which address ranges are 64/128 bit capable. Treated as a 16-bit vector with LSB bit (right-most) indicating address 0 - 0x10000000. See section 50.7 for more information.	0 - 16#FFFF#	0
busw	Bus width of the wide bus area (64 or 128). See section 50.7 for more information.	64, 128	64

50.16 Signal descriptions

Table 688 shows the interface signals of the core (VHDL ports).

*Table 688.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AHB reset	Low
CLK	N/A	Input	AHB clock	-
AHBMI	*	Input	AHB master input	-
AHBMO	*	Output	AHB master output	-
AHBMI1	*	Input	AHB second master input	-
AHBMO1	*	Output	AHB second master output	-
APBI	*	Output	APB slave inputs	-
APBO	*	Input	APB slave outputs	-
IRQ_TRIG[63:0]		Input	Descriptor triggering input	

* see GRLIB IP Library User's Manual

50.17 Library dependencies

Table 689 shows the libraries used when instantiating the core (VHDL libraries).

*Table 689.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	GRDMAC_PKG	Components, signals	GRDMAC internal components and signals.

50.18 Instantiation

In addition to the generic GRDMAC version, *grdmac*, a single-port version of the core is available, *grdmac_1p*, where the *en_ahbm1* generic is preset to '0', and the generics and ports related to the dual-port functionality are removed for convenience.

```
library ieee;
use ieee.std_logic_1164.all;

library gaisler;
use gaisler.misc.all;
use gaisler.grdmac_pkg.all;

library grlib;
use grlib.amba.all;
use grlib.stdlib.all;

entity grdmac_ex is

end entity;

architecture rtl of grdmac_ex is

    -- AMBA signals
    signal apbi : apb_slv_in_type;
    signal apbo : apb_slv_out_vector := (others => apb_none);

    signal ahbsi : ahb_slv_in_type;
    signal ahbso : ahb_slv_out_vector := (others => ahbs_none);

    signal ahbmi : ahb_mst_in_type;
    signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
```

```
begin

    -- ... AHBCTRL

    -- GRDMAC one-AHB-port, AHB master index 1, APB index 0
    -- internal buffer size 1024 bytes, will break bursts at 512 byte boundaries
    -- APB registers at address 0xCCC00200
    dma0 : grdmac_1p
        generic map (
            hmindex     => 1,
            pindex      => 0,
            paddr       => 16#002#,
            hirq        => 1,
            ndmach      => 2,
            bufsize     => 1024, --bytes
            burstbound  => 512 -- bytes
        )
        port map (rstn, clk, ahbmi, ahbmo(1), apbi, apbo(0));

    -- AHB/APB bridge, AHB slave index 2
    apb0: apbctrl
        generic map (hindex => 2, haddr => 16#CCC#)
        port map (rstn,  clk,  ahbsi, ahbso(2), apbi, apbo);

    -- ... APB peripherals

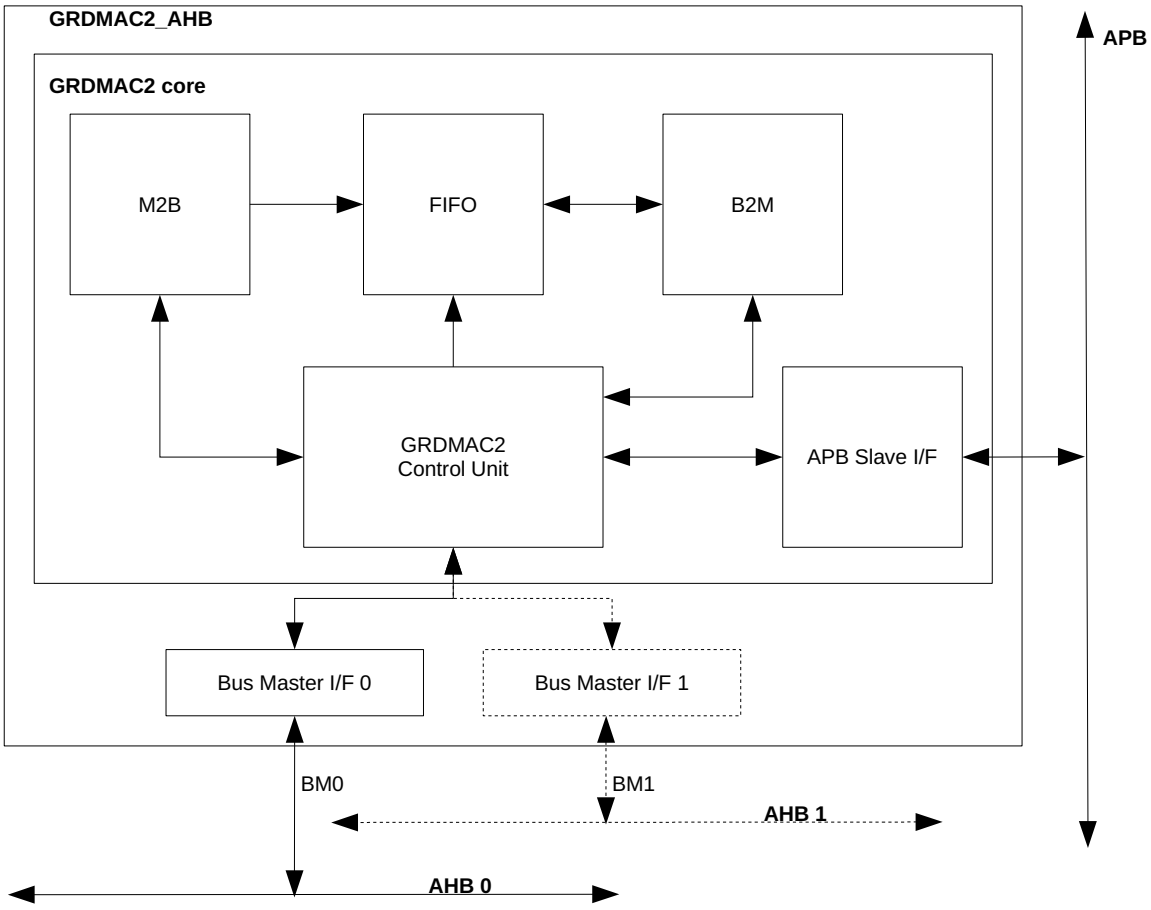
end architecture ; -- rtl
```

51 GRDMAC2 - DMA Controller

51.1 Overview

GRDMAC2 is a flexible direct memory access controller with Bus Master Interface. The basic building blocks of GRDMAC2 are a control module, internal buffer, memory to buffer operation module(M2B), buffer to memory operation module(B2M) and a wrapper layer which provides Bus Master Interface integration and APB register configuration interface. GRDMAC2 supports being attached to any AHB bus with a data width of 32-bit, 64-bit or 128-bit. Wrappers adapting Bus Master interface to both AMBA 2.0 AHB and AXI4 are available. GRDMAC2 can perform burst transfers of data between AHB and APB peripherals at aligned or unaligned memory addresses. GRDMAC2 can be instantiated with one or two Bus Master Interfaces to perform transfers among different AHB buses, therefore GRDMAC2 has a bridging capability.

GRDMAC2 works with self contained descriptors, with all information needed for the DMA transaction. Descriptor configuration allows specification of source and destination addresses which allows a scatter/gather behavior. GRDMAC2 allows multiple types of descriptors which can be broadly classified as data descriptor and conditional descriptor. This enables GRDMAC2 to perform regular data transfer as well as perform an action on occurrence of a specific condition. Conditional descriptors allows an if-else mode of execution based on the condition check outcome. Figure below depicts the basic block diagram of GRDMAC2. The diagram assumes that the AHB wrapper is used, but the functionality is equivalent if the bus master or the AXI wrapper are chosen instead.



With the flexible descriptor queue with conditional descriptors and data descriptors, GRDMAC2 can be used with peripherals like APBUART and other controllers like SPI controller. A typical use case example of GRDMAC2 is described in section 51.8.

GRDMAC2 is equipped with multiple error flags for different kinds of error events and eight individual APB registers solely for debug capability of the current descriptor that is being executed. GRDMAC2 always displays the current execution state in status register and freezes this display in case of an error to make debugging easy.

GRDMAC2 supports the use of an external accelerator core to modify the data in the FIFO between the M2B and the B2M operations. The support to use an accelerator is an extra feature and is not generally included in the core. Currently supported accelerators are:

• AES-256, compatible when using an internal buffer size of 128 bits or wider.

For this accelerator two descriptor types are defined. These are described in section 51.9.

• SHA-256, compatible when using an internal buffer size of 512 bits or wider.

For this accelerator, one descriptor type is defined. This is described in section 51.9.

51.2 Operation

51.2.1 Operation overview

When GRDMAC2 execution is enabled by the user by setting *'EN'* bit in the APB control register, descriptor execution starts from the first descriptor which is pointed by Descriptor Pointer APB Register. GRDMAC2 reads descriptor configurations from any AHB mapped address (typically main memory) through its main Bus Master Interface (BM0 if instantiated with support for two Bus Master Interfaces).
GRDMAC2 decodes the descriptor configuration, identifies the type of descriptor and continues with execution procedure specific to the current descriptor type. On completion of each of the descriptor, GRDMAC2 writes back the status to the descriptor status word, if '*wb*' bit is enabled in the descriptor. Execution continues with the next descriptor pointer by the current descriptor's next descriptor pointer field. In the case of a failed conditional descriptor, DMA controller selects address, pointed by '*fnext.addr'* field, to fetch next descriptor. Fetch-Decode-Execute procedure is performed on each enabled descriptor in the descriptor queue. Any disabled descriptor in the queue is simply skipped and the DMA controller continues execution with the next enabled descriptor in the queue. Descriptor queue should be set up such that the last descriptor in the queue should have '*next.last'* bit value as 1. GRDMAC2 continues execution until it encounters a last descriptor. After completing the last descriptor execution successfully, GRDMAC2 marks the execution status as 'Completed' in the status register and stays idle.
Bus master interface index to be selected for conditional descriptor execution is determined by '*bm*' field in descriptor control word. GRDMAC2 allows the selection of individual Bus Master Interface index for source and destination in data descriptors. The second Bus Master Interface is enabled by setting the VHDL generic *en_bm1* to 1. If the second interface is not enabled, all settings related to it will be ignored and the DMA controller will default to the main Bus Master Interface for all transfers. Descriptor fetches from the memory and write back to the descriptor status word in the memory is performed through default Bus Master Interface, irrespective of the type of descriptor.
The functionalities supported by GRDMAC2 for DMA operation is described in the following sections.

51.2.1.1 Pause and resume

When DMA controller has started processing the descriptors in the current descriptor queue, it is possible to pause the execution in between. Clearing the '*EN*' bit in GRDMAC2 control register(CTRL) will pause the descriptor queue processing. However, execution will pause only after successfully completing the current descriptor execution. GRDMAC2 sets the '*PAU*' bit as 1, and clears the '*ONG*' bit in the status register. Once paused, GRDMAC2 stays idle until '*EN*' and '*KICK*' bit is set in the CTRL register. Once the paused GRDMAC2 is enabled again and kicked by the user, execution resumes from the next descriptor in the queue, if there are enabled descriptors in the remaining part of queue.

51.2.1.2 Restart

GRDMAC2 restarts current descriptor queue execution when '*RT*' bit is set in GRDMAC2 control register (CTRL). Restarting the descriptor queue execution will be performed only after completing the current descriptor execution. DMA controller restarts execution from the first descriptor pointed by Descriptor Pointer APB Register. When '*RT*' bit is set by the user when a descriptor processing is in progress, the request is acknowledged and '*RSP*' bit is set in GRDMAC2 status register. This self clearing bit is cleared by GRDMAC2 when the descriptor queue execution is restarted after the current descriptor is executed completely.

51.2.1.3 Descriptor queue management

GRDMAC2 allows the modification of descriptor queue even after the DMA has started processing the current descriptor queue. Only appending new descriptors are allowed. In order to append new descriptor at the end of descriptor queue, user need to clear '*next.last*' bit of the last descriptor in the current queue and point to the new descriptor as the next descriptor. Mark the new descriptor as the last one. Adding multiple new descriptors also have the same procedure apart from the fact that the last descriptor will be at the end of the newly added part of the queue. Once the descriptor queue is modified, it is mandatory to kick GRDMAC2 to indicate the presence of modified descriptor queue. DMA controller acknowledges kick by setting '*KP*' bit in GRDMAC2 status register. '*KP*' bit is cleared by GRDMAC2 when a new descriptor is taken up for execution. Any type of descriptor queue modification other than appending at the end of the current descriptor queue, is illegal.

51.2.1.4 Reset

Setting '*RST*' bit in GRDMAC2 control register(CTRL) resets the DMA controller, canceling any ongoing descriptor execution. The reset clears all the register fields to their default values.

51.2.2 Data descriptor execution

During a standard data descriptor execution, GRDMAC2 will perform two types of DMA transfers through one of the Bus Master Interfaces: from memory to the internal buffer (M2B) and from the internal buffer to memory (B2M). GRDMAC2 splits the total size of data to be transferred for a data descriptor, in to data chunks of size equal to minimum of the following factors: size of the internal buffer, maximum burst length or a maximum of 1024 bytes. Maximum burst length and size of internal buffer can be set through VHDL generics *max_burst_length, abits* and *dbits*.
After fixing the size of each burst, the DMA controller transfers data chunks from source to destination through M2B and B2M operations. A chunk of data is transferred from memory to the internal buffer as part of the M2B operation and M2B operation is paused. Execution control is switched to B2M operation, where the DMA controller transfers the data chunk from the internal buffer to the destination memory and pauses B2M operation. GRDMAC2 then switches back to the M2B operation which was paused earlier and repeats the same process for the next chunk of data. M2B-B2M cycles are repeated until the total size of data transfer is completed and descriptor is marked as completed. On descriptor completion, based on whether '*wb'* bit is enabled in the descriptor control word, status of the descriptor execution is written back to the descriptor status word.
Data is transferred between source address and destination address which are configured in the descriptor. The offset of source address from which data is fetched is determined by the bit field '*src_fixed_addr*' in the descriptor control word. Similarly the offset of destination address to which data is written is determined by the bit field 'dest_fixed_addr' in the descriptor control word. If src_fixed_addr field is set to 1, GRDMAC2 always fetches data from the same source address. Similarly if '*dest_fixed_addr'* field is set to 1, GRDMAC2 always writes to the same destination address. If any of these fields are not set to 1, GRDMAC2 automatically considers incrementing offsets for source/destination addresses for data read/write. In short, GRDMAC2 can perform burst transactions in the following ways based on the different configurations of '*src_fixed_addr'* and '*dest_fixed_addr'* fields:

1. Data fetched from incrementing source address offsets, written to incrementing destination address offsets.
2. Data fetched from incrementing source address offsets, written to fixed destination address.
3. Data fetched from fixed source address, written to incrementing destination address offsets.
4. Data fetched from fixed source address, written to fixed destination address.

M2B and B2M operation can be performed through the same or different Bus Master Interface index. This selection is performed by configuring *'srcbm'* and *'dstbm'* in descriptor control word provided that second Bus Master Interface is enabled while GRDMAC2 instantiation by VHDL generic *en_bm1*.

- **Data descriptor execution example**

Steps of a data descriptor execution example is given below.

a) GRDMAC2 starts M2B operation.
b) GRDMAC2 checks size of data to be transferred between source and destination for descriptor completion. GRDMAC2 compares the size of internal buffer, maximum burst length and maximum boundary of 1024 bytes and finds out the current burst size.
c) A read burst initiated on the selected Bus Master Interface transfers data from source memory address (incrementing offsets if *'src_fid_addr'* field is not set.) to GRDMAC2 internal buffer. If *'src_fid_addr'* field is set in the descriptor, data is always read from the same source address.
d) GRDMAC2 checks if there is data yet to be fetched from source to complete the descriptor execution. If yes goes to step **e**. Otherwise goes to step **f**.
e) GRDMAC2 pauses M2B operation and switches to B2M operation. Goes to step **g**.
f) GRDMAC2 completes M2B operation and Switches to B2M operation. Goes to step **g**.
g) GRDMAC2 checks size of data to be transferred between source and destination for descriptor completion. GRDMAC2 compares the size of internal buffer, maximum burst length and maximum boundary of 1024 bytes and finds out the current burst size.
h) A write burst is initiated on the selected Bus Master Interface transfers data from GRDMAC2 internal buffer to destination memory address(incrementing offsets if *'dest_fix_addr'* field is not set). If *'dest_fix_addr'* field is set in the descriptor, data is always written to the same destination address.
i) GRDMAC2 checks if there is data yet to be written to destination in order to complete the descriptor execution. If yes goes to step **j**. Otherwise goes to step **k**.
j) GRDMAC2 pauses B2M operation and switches to M2B operation. Goes to step **b**.
k) GRDMAC2 completes B2M operation. Mark descriptor as completed.
l) Writes back the execution status to descriptor status word, if *'wb'* bit is enabled in the descriptor control word.
m) Fetches the next descriptor (pointed by '*next.addr'*) in the queue if there are any. On the other hand, if the current descriptor is the last, marks the execution completion in GRDMAC2 status register.

51.2.3 Conditional descriptor execution

A conditional descriptor is a special kind of descriptor which can be executed to perform a task with conditional behavior. A conditional descriptor can be used to create a DMA transfer that retrieves data from IO cores, therefore off loading the CPU from the task. Usually IO cores provide a status register or an interrupt line to notify the CPU of the availability of new data. A conditional descriptor can be set up to poll this status register or to be triggered by an interrupt, signaling for instance, the availability of new data. This is a typical use case of conditional descriptor execution.

GRDMAC2 identifies a conditional descriptor from the *'type'* field in the descriptor control word. Current version of GRDMAC2 supports 3 types of conditional descriptors. Polling type('*type'* = 1), Triggering type('*type'* = 2) and Poll-on-trigger type('*type*' = 3). The field '*bm*' in descriptor control

word, decides the Bus Master Interface index which performs memory accesses during conditional descriptor execution. This field is relevant only if VHDL generic *en_bm1* is set. Otherwise default Bus Master Interface is used for memory accesses.

Conditional descriptor *'next.addr'* field points to the next descriptor's control word address. Setting *'next.last'* bit marks the current descriptor as the last descriptor in the descriptor queue. Conditional descriptors allows an 'if-else' method of condition check by *'next'* and *'fnext'* fields. On successful execution of a conditional descriptor, GRDMAC2 execution control continues to '*next.addr'* as long as the *'next.last'* bit is not set. On the other hand, if the conditional descriptor execution was failed, execution continues with the next descriptor pointed by *'fnext'*. It is mandatory that *'fnext.last'* bit is never set. Setting this bit will result in a decode descriptor error and *'DE'* will be set in GRDMAC2 status register. This applies to all types of conditional descriptors.
Conditional descriptor fields like *'poll.addr'*, *'expd.data'* and *'cond.mask'* are relevant only for polling or Poll-on-trigger type of descriptors.

51.2.3.1Polling descriptor execution

A polling type descriptor should have *'ctrl.type'* field value 1 in descriptor control word. During a polling descriptor execution, GRDMAC2 polls the address configured in *'poll.addr'* field. Polling is performed in specific number of clock cycle intervals. Interval between polling accesses is decided by *'intrv'* field in descriptor control word. Setting interval between each poll access avoids holding the bus and makes the bus available for other cores in the system.
Polling result data is compared with expected data value configured in *'expd.data'* descriptor field and conditional mask for comparison configured in *'cond.mask'* descriptor field to evaluate the success of the conditional descriptor execution. Condition check is performed as follows.

Table 690. Condition for successful polling descriptor execution

(Content of 'poll.addr') AND 'cond.mask' = 'expd.data' AND 'cond.mask'

In other words, the polling result and expected data is compared only on the bit positions that are set in the conditional mask field. After each polling, condition is checked and this polling-condition check process id repeated for specific number of times based on the value configured in '*count*' field in descriptor control word. If the condition check is not successful even after repeating the polling-condition check process *'count'* times, polling descriptor execution will have a failed status. Otherwise if the condition check was successful before *'count'* times, the polling descriptor execution will be completed successfully.
Apart from the iteration limit on Polling and condition check process, polling descriptor execution is bound by a timeout mechanism if timeout check is enabled in GRDMAC2 control register(CTRL) and during IP instantiation. GRDMAC2 waits for number of clock cycles specified in APB register '*TRST*'(Timer reset value register) before generating a timeout status. Timeout is interpreted either as a failed execution or as an error, based on the field '*errto*' in descriptor control word. Setting '*ctrl.errto*', treats timeout as an error and a failed condition otherwise. On successful condition check, GRDMAC2 marks the descriptor as completed and moves on with the next descriptor execution if there are any present. If the polling descriptor execution is failed, GRDMAC2 fetches next descriptor from address pointed by *'fnext.addr'* descriptor field and continues execution.

- **Polling descriptor execution example**

Steps of a polling descriptor example is given below. A counter is decremented from reset value as soon as the polling descriptor execution has started (If timeout mechanism is enabled by VHDL generic *en_timer*). Reset value of this counter can be configured through GRDMAC2 '*TRST'* register.

a) GRDMAC2 checks if timeout counter value is zero. If yes, Goes to step **i**. Goes to step **c** otherwise

b) GRDMAC2 polls memory address configured in *'poll.addr'*. Starts incrementing counter(C0) which increments every clock cycle. Goes to step **d**.

c) Clears the counter C0. Goes to step **b**.

d) Increment a polling iteration count by 1. Goes to step **e**.

e) GRDMAC2 compares the read data with *'expd.data'* and *'cond.mask'* based on condition in table 690. If condition check is successful, Goes to step **h**. If the condition check is unsuccessful goes to step **f**.
f) GRDMAC2 checks if polling iteration count is equal to *'count*' configured in descriptor control word. If yes goes to step **k**, else goes to step **g**.
g) Checks if the counter (C0) value is equal to *'intrv'* configured in descriptor control word. If yes goes to step **a**, else goes to step **g**.
h) Marks descriptor as completed. Writes back the execution status to descriptor status word, if '*wb'* bit is enabled in the descriptor control word. Goes to step **j**.
i) Marks timeout. If *'errto'* field is set in the descriptor control word, marks error bit *'ERR'* and '*PE'* bit in GRDMAC2 status register and stop ongoing execution. If *'errto'* field is not set in the descriptor control word, goes to step **k**.
j) Fetches the next descriptor(pointed by *'next.addr'*) in the queue if there are any. On the other hand, if the current descriptor is the last, marks the execution completion in GRDMAC2 status register.
k) Writes back descriptor status if *'wb'* bit is enabled in the descriptor control word. Fetches the next descriptor pointed by *'fnext.addr'* and continue execution.

51.2.3.2 Triggering descriptor execution

A triggering type descriptor must have *'type'* field value 2 in descriptor control word. Triggering descriptor monitors the input trigger on the interrupt line specified in the field *'irqn'* in
the descriptor control word. If the timeout mechanism is enabled, the descriptor execution continues the process of monitoring for input trigger, either till the expected trigger is received or till timeout. Based on the *'errto'* bit field in the control word of a triggering descriptor, timeout is treated either as an error or as a failed descriptor execution status.
On successful execution of triggering descriptor, GRDMAC2 marks the descriptor as completed and moves
on with the next descriptor execution if there are any present. If triggering descriptor execution is failed, GRDMAC2 fetches next descriptor from address pointed by *'fnext.addr'* descriptor field and continues execution.
The expected trigger event type is configured by *'ctrl.trtp'* field and *'poll.trig_val'* field in the descriptor. Setting *'ctrl.trtp'* field to 1 in the descriptor will configure the expected event as level and clearing *'ctrl.trtp'* field will configure the expected event as edge. Setting *'poll.trig_val'* field to 1 in the descriptor will configure the expected event as a negative edge/
low level(based on the *'ctrl.trtp'* value). Clearing *'poll.trig_val'* field in the descriptor will configure the expected event as a positive edge/high level(based on the *'ctrl.trtp'* value).

- **Triggering descriptor execution example**

Steps of a triggering descriptor example is given below. A counter is decremented from reset value as soon as the triggering descriptor execution has started(If timeout mechanism is enabled by VHDL generic *en_timer*). Reset value of this counter can be configured through GRDMAC2 '*TRST'* register.

a) GRDMAC2 checks if timeout counter value is zero. If yes, Goes to step **e**. Goes to step **b** otherwise.
b) GRDMAC2 monitors the interrupt line number *'irqn'* configured in descriptor control word for expected event.
c) If expected event occurs, goes to step **d**, else goes to step **a**.
d) Marks descriptor as completed. Writes back the execution status to descriptor status word, if *'wb'* bit is enabled in the descriptor control word. Goes to step **f**.

e) Marks timeout. If *'errto'* field is set in the descriptor control word, Marks error bit *'ERR'* and *'PE'* bit in GRDMAC2 status register and stop ongoing execution. If *'errto'* field is not set in the descriptor control word, goes to step **k**.
f) Fetches the next descriptor (pointed by *next.addr*) in the queue if there are any. On the other hand, if the current descriptor is the last, marks the execution completion in GRDMAC2 status register.
g) Writes back descriptor status if '*wb*' bit is enabled in the descriptor control word. Fetches the next descriptor pointed by *fnext.addr* and continue execution.

51.2.3.3 Poll-on-trigger descriptor execution

A poll-on-trigger type descriptor must have '*ctrl.type*' field value 3 in descriptor control word. Poll-on-trigger type descriptor execution combines the functionality of a triggering descriptor and a polling descriptor. Poll-on-trigger type of descriptor execution monitors the input trigger on the interrupt line specified in the field '*irqn*' in the descriptor control word. On reception of the expected trigger on IRQ line of interest, GRDMAC2 polls the address configured in '*poll.addr*' field. Each polling access is separated by '*intrv*' number of clock cycle as configured in descriptor control word. After each polling the result(content of '*poll.addr'* address) is compared with expected data('*expd.data'*) and mask(*'cond.mask'*) using the condition described in table 690. Polling and condition check process is repeated for *'count'* times as configured in the descriptor control word. Both trigger monitoring and polling are individually bound by timeout if timeout mechanism is enabled.

On successful execution of poll-on-trigger descriptor, GRDMAC2 marks the descriptor as completed and moves on with the next descriptor execution if there are any present. If poll-on-trigger descriptor execution is failed, GRDMAC2 fetches next descriptor from address pointed by '*fnext.addr'* descriptor field and continues execution.

51.3 Configuration

GRDMAC2 can be configured flexibly by means of different VHDL generics. VHDL generics support and their valid values are explained in section .

The DMA controller and descriptors should be configured properly for successful operation of the controller. The following sections describes the DMA controller and descriptor configuration in detail.

51.3.1 GRDMAC2 configuration

GRDMAC2 can be configured through APB register interface. The following APB registers should be used to configure DMA controller.

- **Control register (CTRL)**

Control register allows the configuration of interrupt generation on error and descriptor completion. Control register also allows the user to enable timeout mechanism for conditional descriptors. Timeout mechanism for conditional descriptors can be enabled by setting *'TE*' bit in control register provided that VHDL generic *en_timer* is set on IP instantiation. Conditional descriptor execution is bound by timeout if timeout mechanism is enabled. Conditional descriptor execution will continue for ever or until the successful condition check result if timeout is disabled.

There are designated bits which allows different functionalities like kicking, enabling, resetting, pausing and restarting GRDMAC2. These functionalities are explained in section 51.2.1, in detail.

- **Timer reset value register (TRST)**

Configuring TRST register has an impact only on the conditional descriptor execution, provided that timeout mechanism is enabled. During a conditional descriptor execution, GRDMAC2 will wait for, number of clock cycles configured in Timer reset value register, before generating a timeout status.

- **First descriptor pointer (FPTR)**

First descriptor pointer register should be configured to point to the first descriptor in a descriptor queue.

51.3.2 Descriptor configuration

Descriptor types supported by GRDMAC2 can be broadly classified as data and conditional descriptors. Conditional descriptors are further classified based on the type of condition check they are trying to satisfy. GRDMAC2 demands pre-defined descriptor configuration for each of these types, as described in sections 51.3.2.1 and 51.3.2.2. It is mandatory that descriptors are written to two byte aligned addresses in the memory. By default, Bus Master Interface index 0 is selected for all descriptor reading and execution status write back.

51.3.2.1Data descriptor format

Data descriptor format and description of individual fields are displayed below.

Table 691. Data descriptor format

Address offset	Field
0x00	Control word
0x04	Next descriptor pointer
0x08	Destination base address
0x0C	Source base address
0x10	Status word

Table 692. 0x00 - ctrl - Data descriptor control word

31 11	10	9	8	7	6	5	4 1	0
size	destfix	srcfix	irqe	dstbm	srcbm	wb	type	en

31 : 11	size	Total size of data, in bytes, to be transferred from source to destination.
10	destfix	All data is to be written to the same(fixed) destination address 0- Non fixed destination address(incrementing address offsets) 1- Fixed destination address
9	srcfix	All data is to be read from the same(fixed) source address 0- Non fixed source address(incrementing address offsets) 1- Fixed source address
8	irqe	Enable interrupt on descriptor completion 0- Disable interrupt 1 – Enable interrupt
7	dstbm*	Bus master interface index through which data is to be written to the destination. 0 - BM0 1 - BM1
6	srcbm*	Bus master interface index through which data is to be read from the source. 0 - BM0 1 - BM1
5	wb	Write back status to current descriptor status word, after execution. 0- disable write back 1- enable write back
4 : 1	type	Descriptor type. 0- data descriptor
0	en	Enabled data descriptor. 0- Disabled 1- Enabled

**Field value is relevant if VHDL generic en_bm1 is set to one. Else default Bus Master Interface is selected*

*Table 693.*0x04 - next - Next descriptor pointer

31 1	0
addr	last

31 : 1	addr	MSB of 2 byte aligned next descriptor start address.
0	last	Last descriptor in the descriptor queue. 0 - Not last descriptor 1 - Last descriptor

*Table 694.*0x08 - dest - Destination base address

31 0
addr

31 : 0	addr	Destination base address to which data is to be written.

Table 695. 0x0C - src - Source base address

31			0
addr			
31 : 0	addr	Source base address from which data is to be read out.	

Table 696. 0x10 - sts - Descriptor status word

31			2	1	0
Reserved				err	done
31 : 2		Reserved			
1	err	Descriptor execution error status. 0 - No error. 1 - Error during execution			
0	done	Descriptor completion without error. 0 - Not completed 1 - Completed			

51.3.2.2 Conditional descriptor format

Conditional descriptor format and description of individual fields are displayed below.

Table 697. Conditional descriptor format

Address offset	Field
0x00	Control word
0x04	Next descriptor pointer
0x08	Next descriptor pointer on failure
0x0C	Polling address
0x10	Status word
0x14	Expected data
0x18	Conditional mask

Table 698. 0x00 - ctrl - Conditional descriptor control word

31 24	23 16	15	14	13	12 7	6	5	4 1	0
count	intrv	trtp	irqe	errto	irqn	bm	wb	type	en

31 : 24	count	Number of time the polling and condition check to be repeated before assuming a failed execution status
23 : 16	intrv	Number of clock cycle interval between consecutive polling accesses.
15	trtp	Type of input trigger event expected. 0- Edge 1- Level
14	irqe	Enable interrupt on descriptor completion 0- Disable interrupt 1 – Enable interrupt
13	errto	Timeout is to be treated as an error or a failed conditional descriptor execution. 0 - Timeout is a failed conditional descriptor execution, 1 - Timeout is an error.
12 : 7	irqn	Interrupt line number to be monitored for input trigger, if the current descriptor is a triggering/poll-on-trigger descriptor.
6	bm*	Bus master interface index through which polling accesses are to be performed, if the current descriptor is a polling descriptor. 0 – BM0 1 - BM1
5	wb	Write back status to current descriptor status word, after execution. 0- disable write back 1- enable write back
4 : 1	type	Descriptor type. 1- Polling 2- Triggering 3- Poll-on-trigger
0	en	Enabled data descriptor. 0- Disabled 1- Enabled

**Field value is relevant if VHDL generic en_bm1 is set to one. Else default Bus Master Interface is selected.*

Table 699. 0x04 - next - Next descriptor pointer

31 1	0
addr	last

31 : 1	addr	MSB of 2 byte aligned next descriptor start address. This pointer will be considered for execution when current conditional descriptor has executed successfully.
0	last	Last descriptor in the descriptor queue. 0 - Not last descriptor 1 - Last descriptor

Table 700. 0x08 - fnext - Next descriptor pointer on failure

31			1	0
addr				last
31 : 1	addr	MSB of 2 byte aligned next descriptor start address. This pointer will be considered for execution when current conditional descriptor has failed.		
0	last*	Last descriptor in the descriptor queue. 0 - Not last descriptor		

* *This field should not be set ever. Setting fnext.last field will generate a decode descriptor* error

Table 701. 0x0C - poll - Polling address

31			1	0
addr				trig_ val
31 : 1	addr	MSB of 2 byte aligned address to be polled during descriptor execution if the current descriptor is a polling descriptor.		
0	trig_val	Expected input trigger value if the current descriptor is a triggering descriptor 0- Positive edge/high level based on the ‘ctrl.trtp’ field value of the descriptor. 1 - Negative edge/low level based on the ‘ctrl.trtp’ field value of the descriptor.		

Table 702. 0x10 - sts - Descriptor status word

31			3	2	1	0
Reserved				cpass	err	done
31 : 3	-	Reserved				
2	cpass	Status of condition check. 1 - Conditional descriptor executed successfully. 0 - Conditional descriptor execution failed.				
1	err	Descriptor execution error status. 0 - No error. 1 – Error during execution				
0	done	Descriptor completion without error. 0 - Not completed 1 - Completed				

Table 703. 0x14 - expd - Expected data

31			0
data			
31 : 0	data	Expected data for polling type of descriptor execution. GRDMAC2 compares polling result with this data during descriptor execution.	

Table 704. 0x18 - cond - Conditional Mask

31		0
mask		

31 : 0	mask	Mask for polling type of descriptor execution. GRDMAC2 compares polling result with this mask during descriptor execution. Polling result data bits corresponding to mask bits which are set to one, is only considered for condition check comparison with expected data (‘expd.data’).

51.4 Bus Master Interface

While executing data descriptors, GRDMAC2 performs AHB data transfer at the address and of the size specified. The AHB accesses in GRDMAC2 is performed through a Bus Master Inter ace. GRDMAC2 supports single or double Bus Master Interface implementation by VHDL generic *en_bm1* configuration. When GRDMAC2 is configured with the VHDL generic *en_bm1* set to ‘1’, a secondary Bus Master Interface will be enabled in GRDMAC2 core. This interface can be connected to a second AHB bus to provide bridging capabilities to the DMA controller. GRDMAC2 will perform data accesses though this interface when the *‘bm’*(*‘srcbm’/’dstbm’* in case of data descriptors) field in the descriptor’s control word is set to 1. This bit field will be ignored and default Bus Master Interface is used in case GRDMAC2 is configured with the VHDL generic *en_bm1* set to ‘0’. Bus master interface front end data width can be configured to 32, 64 or 128 based on VHDL generic *dbits.* Bus master interface brings in the flexibility of aligned or unaligned accessed in GRDMAC2. Burst accesses are always limited by 1KB boundary. However this can be further limited by setting VHDL generic *max_burst_length.* At the specified memory boundary the burst will be interrupted, an idle cycle will be inserted making re-arbitration on the AHB bus more frequent.

Additionally Frontgrade Gaisler provides Bus Master interface wrappers for both AMBA AHB 2.0 and AXI4. The wrapper for AXI4 features a bridge to convert the Bus Master into an AXI4 Bus Master Interface. For further information regarding the accesses contact Frontgrade Gaisler.

51.5 Internal buffer

For any data descriptor execution, data is fetched from the source memory and stored in an internal buffer in GRDMAC2. This data is later moved from the internal buffer and written to the destination memory. The internal buffer uses RAM implemented using GRLIB parameterizable SYNCRAMBW memory, assuring portability to all supported technologies. Depth of the RAM depends on the VHDL generic *abits*. Width of the internal buffer is configured identical to data width of Bus Master Interface. Fault tolerance can be added to the RAM by setting the VHDL generic *ft* to a valid value different than 0. Note that the VHDL generic *ft* needs to be set to 0 unless GRDMAC2 is used together with the fault tolerant version of GRLIB, which is not available under the terms of the GPL. Technology for SYNCRAM can be selected by VHDL generic *tech*.

51.6 Interrupts

GRDMAC2 provides fine-grained control of interrupt generation. At the highest level, the global Interrupt Enable bit (*‘IE’*) in GRDMAC2 control register(CTRL) can be set to zero to mask every interrupt setting in GRDMAC2. If set to one, interrupt generation depends on the following settings. The Interrupt on Error Enable bit (*‘IER’*) in CTRL register provides a way to generate interrupts in the event of errors. Error generation is discussed further in section 51.7. An interrupt can be also generated by the successful completion of a descriptor, if the Interrupt Enable (*‘irqe’*) bit is set to one in the descriptor’s control field. The Interrupt Mask bit (*‘IM’*) in the CTRL register can be set to one to mask the descriptor completion interrupts. Descriptor completion interrupt will be generated as soon as the successful completion of the descriptor execution. Even before writing back the descriptor’s status in main memory, if the write back is enabled. For both interrupts on error and interrupts on descriptor

completion events, a flag will be raised in the interrupt flag bit(*'IF'*) in GRDMAC2 status register. Interrupt generated on descriptor completion can be masked by configuring *'IM'* bit even though *'IE'* bit in CTRL register and '*irqe*' bit in descriptor control word is set to one. However it is not possible to mask the interrupt generated on error when *'IE'* bit in CTRL register is set to one. As an example of interrupt generation setup, one can perform the following steps. The Interrupt Enable (*'IE'*) bit in CTRL register must be set to one. The Interrupt Mask (*'IM'*) bit in CTRL register must be configured as zero. Set up a descriptor queue with a single data descriptor. Configure Interrupt Enable (*'irqe'*) bit in the control field of the descriptor as one and mark the descriptor as the last descriptor. When the descriptor queue is completed successfully, an interrupt will be generated.

51.7 Status and Errors

GRDMAC2 provides a very clear status display by an APB status register and 8 separate debug capability registers which displays details of the current descriptor that is being executed.

There is a general error flag *'ERR'* which will be set in any case of error and an individual flag which says the type of error that has occurred. A decode descriptor error(*'DE'*) can occur during the decoding of a descriptor if the type of descriptor is not a valid value(0 to 3 is only valid) for current version of GRDMAC2. Another situation which can generate this error is when a conditional descriptor has it's *'fnext.last'* bit set to one. A read descriptor error(*'RE'*) will be flagged in GRDMAC2 status register in case when the Bus Master Interface receives an error response during the descriptor read burst memory access. A polling error flag(*'PE'*) can be set in two cases. One case is when a Bus Master Interface receives an error response during the read access on the address being polled.

Another case is when timeout occurs during a polling descriptor execution and the descriptor is configured to treat timeout as an error. A triggering error(*'TRE'*) will be generated when a triggering type descriptor encounters a timeout and the descriptor is configured to treat timeout as an error. When the Bus Master Interface receives an error response during the write access on descriptor status word in the memory, a write back error(*'WBE'*) is flagged in GRDMAC2 status register. If the Bus Master Interface receives an error response during any part of the read accesses performed as part of M2B operation, M2B read data error flag(*'RDE'*) will be set status register. Similarly, in case the Bus Master Interface receives an error response during any part of the write accesses performed as part of B2M operation, B2M write data error

flag(*'WDE'*) will be set in status register. A FIFO error(*'FE'*) is flagged in GRDMAC2 status register only when an uncorrectable error status is reported from the internal FIFO when VHDL generic *ft* is enabled. In such a situation the *'FP'* field will be pointing to the offset in FIFO where the error has been detected. When GRDMAC2 receives a kick request, it reads the next descriptor pointer of the current descriptor again. If the Bus Master Interface receives an error response while this read access, a *'NPE'* flag will be set in APB status register. Timeout status flag *'TO'* will be set in case of a timeout during the conditional descriptor execution. All error flags are cleared on writing 1 to them. However the timeout flag *'TO'* is self clearing. In case of an error, GRDMAC2 pauses the execution and stays idle. Along with appropriate error flags, *'PAU'* flag will be set in GRDMAC2 status register. GRDMAC2 allows the user to clear the errors and kick GRDMAC2 to continue execution from the next descriptor in the queue.

GEDMAC2 has a 5 bit *'ST'* field which always displays the current state of descriptor execution. In case of any error, the *'ST'* field will freeze at the execution state where the error occurred. When GRDMAC2 is performing descriptor execution in a descriptor queue, the '*ONG'* bit in GRDMAC2 status register will be set to one. In case of an error or if GRDMAC2 has completed the execution of the entire queue, GRDMAC2 stays idle and clears the *'ONG'* bit, indicating that no operation is currently in progress. *'CMP'* flag is set to one, only when the descriptor queue is executed completely.

There are two self clearing flags *'KP'* and *'RSP'* which shows the existence of a pending kick request and a pending restart request. *'KP'* flag will be cleared whenever a new descriptor is taken up for execution. In other words, GRDMAC2 handles kick by reading the *'next'* field of the current descriptor again from the memory and processing it. *'RSP'* bit is cleared when the current descriptor execution is

completed and DMA operation restarted from the first descriptor. The debug capability registers displays all the fields of the current descriptor that is being executed.

The descriptor pointer debug capability register shows the base address where the current descriptor was read from.

51.8 GRDMAC2 Use case Example

A simple example showing how GRDMAC2 can be used with APBUART is explained below. In this example GRDMAC2 is used to read data, whenever a peripheral APBUART receives data from a host. GRDMAC2 transfers data from the APBUART receiver register to main memory. Assume that the system has SRAM memory at 0x30000000 and UART configuration registers are located at 0x80300000. Assume that IP is instantiated with VHDL generic *en_bm1* set to 1 and Bus Master Interface index 1 is connected to APB bus on which UART is also connected. SRAM is connected on the main bus to which default Bus Master Interface index 0 is connected.

1. Set up a descriptor queue with 3 descriptors. A polling descriptor followed by a data descriptor and a disabled data descriptor. Descriptor queue should be written to a memory location (assuming 0x30000000). The descriptor configuration is listed below.

a) Polling descriptor:

Table 705. GRDMAC2 use case example- Conditional descriptor configuration

Field	Value	Comment
ctrl.en	1	Enabled descriptor
ctrl.type	1	Polling descriptor
ctrl.wb	1	Execution status write back is enabled
ctrl.irqn	-	Irrelevant for polling descriptors
ctrl.trtp	-	Irrelevant for polling descriptors
ctrl.intrv	0xFF	0xFF clock cycles between two polling accesses
ctrl.count	0xFF	Repeat polling condition check for 0xFF iterations before assuming failed execution status
ctrl.bm	1	Bus Master Interface 1 is to be used for UART status register polling
ctrl.errto	0	Timeout is not an error
ctrl.irqe	1	Interrupt enabled on descriptor completion
next.addr	0x3000001C	Pointer to next descriptor
nxt_des.last	0	Not last descriptor
f_nxt_des.addr	0x30000030	Pointer to next descriptor if the current descriptor execution failed.
f_nxt_des.last	0	Mandatory value
poll.trg_val	-	Irrelevant for polling descriptors
poll.addr	0x80300004	APBUART status register address.
expd.data	0x00000001	Expects data ready to be 1. Break, Framing error, Parity error and Overrun bits to be 0.
cond.mask	0x00000079	Consider bit 0, 3, 4, 5 and 6 for comparison using the condition check in table 690

b) Data descriptor 1

This descriptor is written at 0x3000001C

Table 706. GRDMAC2 use case example- Data descriptor configuration

Field	Value	Comment
ctrl.en	1	Enabled descriptor
ctrl.type	0	Data descriptor
ctrl.wb	1	Execution status write back is enabled
ctrl.dstbm	0	Bus master interface 0 is to be used for data write to SRAM area.
ctrl.srcbm	1	Bus master interface 1 is to be used for data fetch from UART RX register.
ctrl.destfix	0	Write data to incrementing destination address offsets
ctrl.srcfix	1	Fetch data always from the same source address
ctrl.irqe	1	Interrupt enabled on descriptor completion
ctrl.size	64	Transfer 64 bytes of data
next.addr	0x30000000	Pointer to next descriptor. This essentially creates a loop of polling- data transfer process
nxt_des.last	0	Not last descriptor
dest.addr	0x30000080	SRAM area where data read from UART is to be written
src.addr	0x80300000	APBUART data register from where data is to be read.

2. Write the same data descriptor described in step **b** at address 0x30000030 with *'ctrl.en'* bit as zero and *'nxt_des.last'* bit as 1. This disabled data descriptor is pointed as the descriptor to be executed on failure for the first polling descriptor.

3. Configure GRDMAC2 '*FPTR'* register as 0x30000000, control register bit field *'TE'* as zero.

4. After configuring descriptor queue and GRDMAC2 registers, start GRDMAC2 operation by setting GRDMAC2 control register(CTRL) *'EN'* bit field as one.

GRDMAC2 should transfer data from UART data register whenever the data ready bit in APBUART status register is set. Transferred data can be read from the destination SRAM area and verified.

51.9 Accelerator functionality

There is an option to include accelerator functionality to the core. Whether or not this functionality is added is decided in the configuration options (VHDL generics). This accelerator can be used to replace data in the FIFO between the M2B and B2M operations.

Currently the following accelerators are supported:

• AES-256 (when using an internal buffer size of 128 bits or wider)

• SHA-256 (when using an internal buffer size of 512 bits or wider)

Currently the following descriptor types are supported:

• 4 - Encryption descriptor for AES-256

• 5 - Accelerator update descriptor for AES-256

• 6 - Authentication descriptor for SHA-256

51.9.1 AES-256 Accelerator description

The Advanced Encryption Standard (AES) is a symmetric encryption algorithm. The accelerator implements the AES algorithm with 256-bit key length using CTR mode of operation. The AES algorithm is specified in the "Advanced Encryption Standard (AES)" document, Federal Information Processing Standards (FIPS) Publication 197. The document is established by the National Institute of

Standards and Technology (NIST). Encryption and decryption are equal using this operation mode, meaning that if one encrypts already encrypted data using the same KEY and Initialization Vector (IV) the result will be the decrypted data.

To use the AES-256 accelerator the KEY is updated using an accelerator update descriptor with a length of 32 bytes and the IV is updated using and accelerator update descriptor with a length of 16 bytes. This descriptor specifies where the KEY and IV data is stored and the GRDMAC2 fetches the data and transfers it to the register in the AES-256 accelerator. The KEY can be reused by several encryption/decryption descriptors but the IV needs to be updated between every new descriptor. **It is very important that the IV always is a new unique value if it is ever used with the same key for a different descriptor.**

After updating the IV (and the KEY if needed) the encryption descriptor is used. The encryption works for lengths of any number of bytes, same as for regular data descriptors.

51.9.2 SHA-256 Accelerator description

The Secure Hash Algorithm (SHA) is a cryptographic hash function. The accelerator implements the SHA algorithm with a 256 bit hash output. The SHA algorithm is specified in the “Secure Hash Standard (SHS)” document, Federal Information Processing Standards (FIPS) Publication 180-4. The document is established by the National Institute of Standards and Technology (NIST).

To use the SHA-256 accelerator the authentication descriptor is used. The authentication works for lengths of any number of bytes, same as for regular data descriptors.

51.9.3 Encryption descriptor format

Encryption descriptor format and description of individual fields are displayed below.

Table 707. Encryption descriptor format

Address offset	Field
0x00	Control word
0x04	Next descriptor pointer
0x08	Destination base address
0x0C	Source base address
0x10	Status word

Table 708. 0x00 - ctrl - Encryption descriptor control word

31 11	10	9	8	7	6	5	4 1	0
size	destfix	srcfix	irqe	dstbm	srcbm	wb	type	en

31 : 11	size	Total size of data, in bytes, to be transferred from source to destination.
10	destfix	All data is to be written to the same(fixed) destination address 0- Non fixed destination address(incrementing address offsets) 1- Fixed destination address
9	srcfix	All data is to be read from the same(fixed) source address 0- Non fixed source address(incrementing address offsets) 1- Fixed source address
8	irqe	Enable interrupt on descriptor completion 0- Disable interrupt 1 – Enable interrupt
7	dstbm*	Bus master interface index through which data is to be written to the destination. 0 - BM0 1 - BM1
6	srcbm*	Bus master interface index through which data is to be read from the source. 0 - BM0 1 - BM1
5	wb	Write back status to current descriptor status word, after execution. 0- disable write back 1- enable write back
4 : 1	type	Descriptor type. 4- encryption descriptor
0	en	Enabled descriptor. 0- Disabled 1- Enabled

**Field value is relevant if VHDL generic en_bm1 is set to one. Else default Bus Master Interface is selected*

Table 709. 0x04 - next - Next descriptor pointer

31 1	0
addr	last

31 : 1	addr	MSB of 2 byte aligned next descriptor start address.
0	last	Last descriptor in the descriptor queue. 0 - Not last descriptor 1 - Last descriptor

Table 710. 0x08 - dest - Destination base adress

31	0
addr	

31 : 0	addr	Destination base address to which data is to be written.

Table 711. 0x0C - src - Source base adress

31			0
addr			
31 : 0	addr	Source base address from which data is to be read out.	

Table 712. 0x10 - sts - Descriptor status word

31			2	1	0
Reserved				err	done
31 : 2		Reserved			
1	err	Descriptor execution error status. 0 - No error. 1 - Error during execution			
0	done	Descriptor completion without error. 0 - Not completed 1 - Completed			

- **Encryption descriptor execution example**

Steps of an encryption descriptor execution example is given below.

a) GRDMAC2 starts M2B operation.

b) GRDMAC2 checks size of data to be transferred between source and destination for descriptor completion. GRDMAC2 compares the size of internal buffer, maximum burst length and maximum boundary of 1024 bytes and finds out the current burst size.

c) A read burst initiated on the selected Bus Master Interface transfers data from source memory address (incrementing offsets if 'src_fid_addr' field is not set.) to GRDMAC2 internal buffer. If 'src_fid_addr' field is set in the descriptor, data is always read from the same source address.

d) GRDMAC2 checks if there is data yet to be fetched from source to complete the descriptor execution. If yes goes to step e. Otherwise goes to step f.

e) GRDMAC2 pauses M2B operation and switches to ACC operation. Goes to step g.

f) GRDMAC2 completes M2B operation and Switches to ACC operation. Goes to step g.

g) GRDMAC2 executes encryption on the data currently in the internal buffer and replaces it with the encrypted data. GRDMAC2 checks if M2B is paused. If yes goes to step **h**. Otherwise goes to step **i**.

h) GRDMAC2 pauses ACC operation and switches to B2M operation. Goes to step j.

i) GRDMAC2 completes ACC operation and switches to B2M operation. Goes to step j.

j) GRDMAC2 checks size of data to be transferred between source and destination for descriptor completion. GRDMAC2 compares the size of internal buffer, maximum burst length and maximum boundary of 1024 bytes and finds out the current burst size.

k) A write burst is initiated on the selected Bus Master Interface transfers data from GRDMAC2 internal buffer to destination memory address(incrementing offsets if 'dest_fix_addr' field is not set). If 'dest_fix_addr' field is set in the descriptor, data is always written to the same destination address.

l) GRDMAC2 checks if there is data yet to be written to destination in order to complete the descriptor execution. If yes goes to step m. Otherwise goes to step n.

m) GRDMAC2 pauses B2M operation and switches to M2B operation. Goes to step b.

n) GRDMAC2 completes B2M operation. Mark descriptor as completed.

o) Writes back the execution status to descriptor status word, if ‘wb’ bit is enabled in the descriptor control word.

p) Fetches the next descriptor (pointed by ‘next.addr’) in the queue if there are any. On the other hand, if the current descriptor is the last, marks the execution completion in GRDMAC2 status register.

51.9.4 Accelerator update descriptor format

Table 713. Accelerator update descriptor format

Address offset	**Field**
0x00	Control word
0x04	Next descriptor pointer
0x08	Source base address
0x0C	Status word

Table 714. 0x00 - ctrl - Accelerator update descriptor control word

31 9	8	7	6	5	4 1	0
size	srcfix	irqe	srcbm	wb	type	en

31 : 9	size	Total size of data, in bytes, to be transferred from source to destination.
8	srcfix	All data is to be read from the same(fixed) source address 0- Non fixed source address(incrementing address offsets) 1- Fixed source address
7	irqe	Enable interrupt on descriptor completion 0- Disable interrupt 1 – Enable interrupt
6	srcbm*	Bus master interface index through which data is to be read from the source. 0 - BM0 1 - BM1
5	wb	Write back status to current descriptor status word, after execution. 0- disable write back 1- enable write back
4 : 1	type	Descriptor type. 5- accelerator update descriptor
0	en	Enabled descriptor. 0- Disabled 1- Enabled

**Field value is relevant if VHDL generic en_bm1 is set to one. Else default Bus Master Interface is selected*

Table 715. 0x04- next - Next descriptor pointer

31 1	0
addr	last

31 : 1	addr	MSB of 2 byte aligned next descriptor start address.
0	last	Last descriptor in the descriptor queue. 0 - Not last descriptor 1 - Last descriptor

Table 716. 0x08 - src - Source base adress

31			0
addr			
31 : 0	addr	Source base address from which data is to be read out.	

Table 717. 0x0C - sts - Descriptor status word

31			2	1	0
Reserved				err	done
31 : 2		Reserved			
1	err	Descriptor execution error status. 0 - No error. 1 - Error during execution			
0	done	Descriptor completion without error. 0 - Not completed 1 - Completed			

- **Accelerator update descriptor execution example**

Steps of an accelerator update descriptor execution example is given below.

a) GRDMAC2 starts M2B operation.

b) GRDMAC2 checks size of data to be transferred between source and destination for descriptor completion. GRDMAC2 compares the size of internal buffer, maximum burst length and maximum boundary of 1024 bytes and finds out the current burst size.

c) A read burst initiated on the selected Bus Master Interface transfers data from source memory address (incrementing offsets if 'src_fid_addr' field is not set.) to GRDMAC2 internal buffer. If 'src_fid_addr' field is set in the descriptor, data is always read from the same source address.

d) GRDMAC2 checks if there is data yet to be fetched from source to complete the descriptor execution. If yes goes to step e. Otherwise goes to step f.

e) GRDMAC2 pauses M2B operation and switches to ACC operation. Goes to step g.

f) GRDMAC2 completes M2B operation and Switches to ACC operation. Goes to step g.

g) GRDMAC2 executes encryption on the data currently in the internal buffer and replaces it with the encrypted data.

h) GRDMAC2 pauses ACC operation and switches to M2B operation. Goes to step b.

i) GRDMAC2 completes ACC operation. Mark descriptor as completed.

j) Writes back the execution status to descriptor status word, if 'wb' bit is enabled in the descriptor control word.

k) Fetches the next descriptor (pointed by 'next.addr') in the queue if there are any. On the other hand, if the current descriptor is the last, marks the execution completion in GRDMAC2 status register.

51.9.5 Authentication descriptor format

Authentication descriptor format and description of individual fields are displayed below.

Table 718. Authentication descriptor format

Address offset	Field
0x00	Control word
0x04	Next descriptor pointer
0x08	Destination base address
0x0C	Source base address
0x10	Status word

Table 719. 0x00 - ctrl - Authentication descriptor control word

31 11	10	9	8	7	6	5	4 1	0
size	destfix	srcfix	irqe	dstbm	srcbm	wb	type	en

31 : 11	size	Total size of data to be read from source
10	destfix	All data is to be written to the same(fixed) destination address 0- Non fixed destination address(incrementing address offsets) 1- Fixed destination address
9	srcfix	All data is to be read from the same(fixed) source address 0- Non fixed source address(incrementing address offsets) 1- Fixed source address
8	irqe	Enable interrupt on descriptor completion 0- Disable interrupt 1 – Enable interrupt
7	dstbm*	Bus master interface index through which data is to be written to the destination. 0 - BM0 1 - BM1
6	srcbm*	Bus master interface index through which data is to be read from the source. 0 - BM0 1 - BM1
5	wb	Write back status to current descriptor status word, after execution. 0- disable write back 1- enable write back
4 : 1	type	Descriptor type. 6- authentication descriptor
0	en	Enabled descriptor. 0- Disabled 1- Enabled

**Field value is relevant if VHDL generic en_bm1 is set to one. Else default Bus Master Interface is selected*

Table 720. 0x04 - next - Next descriptor pointer

31 1	0
addr	last

31 : 1	addr	MSB of 2 byte aligned next descriptor start address.
0	last	Last descriptor in the descriptor queue. 0 - Not last descriptor 1 - Last descriptor

Table 721. 0x08 - dest - Destination base adress

31			0
addr			

31 : 0	addr	Destination base address to which data is to be written.

Table 722. 0x0C - src - Source base adress

31			0
addr			

31 : 0	addr	Source base address from which data is to be read out.

Table 723. 0x10 - sts - Descriptor status word

31			2	1	0
Reserved				err	done

31 : 2		Reserved
1	err	Descriptor execution error status. 0 - No error. 1 - Error during execution
0	done	Descriptor completion without error. 0 - Not completed 1 - Completed

- **Authentication descriptor execution example**

Steps of an authentication descriptor execution example is given below.

a) GRDMAC2 starts M2B operation.

b) GRDMAC2 checks size of data to be read and processed before descriptor execution completed. GRDMAC2 compares the size of internal buffer, maximum burst length and maximum boundary of 1024 bytes and finds out the current burst size.

c) A read burst initiated on the selected Bus Master Interface transfers data from source memory address (incrementing offsets if 'src_fid_addr' field is not set.) to GRDMAC2 internal buffer. If 'src_-fid_addr' field is set in the descriptor, data is always read from the same source address.

d) GRDMAC2 checks if there is data yet to be fetched from source to complete the descriptor execution. If yes goes to step e. Otherwise goes to step f.

e) GRDMAC2 pauses M2B operation and switches to ACC operation. Goes to step g.

f) GRDMAC2 completes M2B operation and Switches to ACC operation. Goes to step g.

g) GRDMAC2 executes authentication on the data currently in the internal buffer and clears the internal buffer. GRDMAC2 checks if M2B is paused. If yes goes to step h. Otherwise goes to step i.

h) GRDMAC2 pauses ACC operation and switches to M2B operation. Goes to step b.

i) GRDMAC2 completes ACC operation and switches to B2M operation. Goes to step j.

j) GRDMAC2 sets size of data to be transferred to destination for descriptor completion to 256 bits. GRDMAC2 compares the size of internal buffer, maximum burst length and maximum boundary of 1024 bytes and finds out the current burst size.

k) A write burst is initiated on the selected Bus Master Interface transfers data from GRDMAC2 internal buffer to destination memory address(incrementing offsets if 'dest_fix_addr' field is not set). If 'dest_fix_addr' field is set in the descriptor, data is always written to the same destination address.

l) GRDMAC2 completes B2M operation. Mark descriptor as completed.

m) Writes back the execution status to descriptor status word, if 'wb' bit is enabled in the descriptor control word.

n) Fetches the next descriptor (pointed by 'next.addr') in the queue if there are any. On the other hand, if the current descriptor is the last, marks the execution completion in GRDMAC2 status register.

51.10 Endianness

GRDMAC2 is designed for both big-endian and little-endian systems.

51.11 Registers

GRDMAC2 is programmed through registers mapped into APB address space. The APB address configured with VHDL generics *paddr* and *pmask*.

Table 724. GRDMAC2 APB registers

APB address offset	Register
0x00	Control register
0x04	Status register
0x08	Timer reset value register
0x0C	Capability register
0x10	First descriptor pointer
0x14	Descriptor control word for debug capability
0x18	Next Descriptor pointer for debug capability
0x1C	fnext/dest.addr for debug capability*
0x20	poll.addr/src.addr for debug capability*
0x24	Descriptor status for debug capability
0x28	expd.data/Null for debug capability*
0x2C	cond.mask /Null for debug capability*
0x30	Current descriptor pointer for debug capability

*Debug capability register field according to the current descriptor type is conditional/data

51.11.1 Control register

Table 725. 0x00 - CTRL - Control register

31	8	7	6	5	4	3	2	1	0
Reserved		TE*	IER	IM	IE	RT	KICK	RST	EN
0		0	0	0	0	0	0	0	0
r		*	rw	rw	rw	rw	rw	rw	rw

31 : 8	-	Reserved
7	TE*	Timeout check enable during conditional descriptor execution.
6	IER	Enable interrupt generation on error events
5	IM	Mask interrupt generation on descriptor completion.
4	IE	Enable interrupt generation. 0 - Disable interrupt all type of generation 1- Enable interrupt generation
3	RT	Restart current descriptor queue execution.
2	KICK	Kick GRDMAC2. GRDMAC2 reads the next descriptor pointer word of the current descriptor again when there is a kick request.
1	RST	Reset GRDMAC2. Setting this bit to one resets the core completely.
0	EN	Enable DMA controller.

** Field value is relevant only when VHDL generic 'en_timer' is set to 1.*

51.11.2 Status register

Table 726. 0x04 - STS - Status Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NPE	WDE	RDE	TO	WBE	TRE	PE	RE	DE	KP	RSP	IF	PAU	ONG	ERR	CMP
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
wc	wc	wc	wc	wc	wc	wc	wc	wc	r	r	wc	r	r	wc	r

Table 727.

31 : 27	26 : 17	16
ST	FP	FE
0	0*	0
r	r	wc

31 : 27	ST	Current GRDMAC2 operation state
26 :17	FP*	FIFO offset at which error encountered.
16	FE*	FIFO error. Error reported from GRDMAC2 internal FIFO.
15	NPE	Error during reading next descriptor pointer register, on a kick request
14	WDE	Error while writing data during B2M operation.
13	RDE	Error while reading data during M2B operation.
12	TO	Timeout during conditional descriptor execution.
11	WBE	Write back error.
10	TRE	Triggering descriptor error.
9	PE	Polling error.
8	RE	Read descriptor error.
7	DE	Decode descriptor error.
6	KP	Pending kick request.
5	RSP	Pending restart request.
4	IF	Interrupt flag.
3	PAU	Paused descriptor queue execution.
2	ONG	Ongoing descriptor queue execution.
1	ERR	Error during descriptor queue execution.
0	CMP	Completed execution of the descriptor queue.

**Field value relevant only if VHDL generic 'ft' is set to non zero value.*

51.11.3 Timer reset value register

Table 728. 0x08 - TRST - Timer reset value register

31	0
VAL	
0	
rw	

31 : 0	VAL	Reset value of the timeout counter for timeout check mechanism. *

**Considered for conditional descriptor execution, only if VHDL generic 'en_timer' and APB control register 'TE' field is set to one.*

51.11.4 Capability Register

Table 729. 0x0C - CAP - Capability Register

31 28	27 17	16 12	11 9	8	7 5	4	3 0
BFDP	Reserved	ACC	DW	TE	FT	BM1	VER
*	0	*	*	*	*	*	*
r	r	r	r	r	r	r	r

31 : 28	BFDP	Address bits of internal FIFO. FIFO depth is calculated as 2^(abits). Determined by VHDL generic 'a*bits*'.
27 : 16	-	Reserved
16 : 12	ACC	Value of VHDL generic *"en_acc"* configured during IP Instantiation.
11 : 9	DW	Bus master interface front end data width. Determined by VHDL generic '*dbits*'.
8	TE	Timeout mechanism support. Determined by VHDL generic '*en_timer*'.
7 : 5	FT	Value of VHDL generic '*ft*' configured during IP Instantiation.
4	BM1	Second Bus Master Interface support. Determined by VHDL generic '*en_bm1*'.
3 : 0	VER	Revision of GRDMAC2

51.11.5 First descriptor pointer

Table 730. 0x10 - FPTR - First descriptor pointer

31	0
addr	
0	
rw	

31 : 0	addr	First descriptor pointer register. Points to first descriptor of the descriptor queue.

51.11.6 Descriptor control word for debug capability

Table 731. 0x14 - DCTR - Descriptor control word for debug capability

31	0
ctrl	
0	
r	

31 : 0	ctrl	Current descriptor's control field for debug capability.

51.11.7 Next Descriptor pointer for debug capability

*Table 732.*0x18 - DNXT - Next Descriptor pointer for debug capability

31	0
addr	
0	
r	

31 : 0	addr	Current descriptor's next descriptor field for debug capability

51.11.8 fnext/dest.addr for debug capability

*Table 733.*0x1C - DFNX - fnext/dest.addr for debug capability

31	0
addr	
0	
r	

31 : 0	addr	Current descriptor's next descriptor field on failure for debug capability/ Current descriptor's destination address field for debug capability*

* Debug capability field depends on the descriptor type.

51.11.9 poll.addr/src.addr for debug capability

Table 734. 0x20 - DPOL- poll.addr/src.addr for debug capability

31	0
addr	
0	
r	

31 : 0	addr	Current descriptor's 'poll.addr' for debug capability/Current descriptor's src.addr field for debug capability*

* Debug capability field depends on the descriptor type.

51.11.10 Descriptor status for debug capability

Table 735. 0x24 - DSTS - Descriptor status for debug capability

31	0
sts	
0	
r	

31 : 0	sts	Current descriptor's status word for debug capability

51.11.11 Expected data for debug capability

*Table 736.*0x28 - DDTA- Expected data for debug capability

31	0
data	
0	
r	

31 : 0	data	Current descriptor's expected data(expd.data) for debug capability*

* This field value is null for a data descriptor.

51.11.12 Conditional mask for debug capability

Table 737. 0x2C - DMSK - Conditional mask for debug capability

31	0
mask	
0	
r	

31 : 0	mask	Current descriptor's conditional mask for debug capability*

* This field value is null for a data descriptor.

51.11.13 Current descriptor pointer for debug capability

Table 738. 0x30 - DPTR - Current descriptor pointer for debug capability

31	0
addr	
0	
r	

31 : 0	addr	Pointer from which the current descriptor is read, for debug capability

51.12 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x0C0 (GRDMAC2). For description of vendor and device identifiers see GRLIB IP Library User's Manual.

51.13 Implementation

51.13.1 Reset

GRDMAC2 changes its reset behavior depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

GRDMAC2 will feature a synchronous reset if the parameter grlib_async_reset_enable is not set in the GRLIB configuration package. On the contrary, it will implement an asynchronous reset if grlib_async_reset_enable is set.

The reset is applied to all the registers in GRDMAC2.

51.14 Configuration options

Table 739 shows the configuration options of GRDMAC2 (VHDL generics).

Table 739. Configuration options

Generic name	Function	Allowed range	Default
tech	Technology for the internal FIFO	0 to NTECH	inferred
hindex0	AHB master index(BM0)	0 - NAHBMST-1	0
hindex1	AHB master index(BM1)	0 - NAHBMST-1	1
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar	0 - 16#FFF#	0
pmask	Mask field of the APB bar	0 - 16#FFF#	16#FF8#
pirq	Interrupt line used by GRDMAC2	0 - NAHBIRQ-1	1
ft	Enable fault-tolerance against SEU errors. GRDMAC2 can optionally be implemented with fault-tolerance against SEU errors in the FIFO memories by setting this generic to 1, 2,3,4 or 5. Possible options are byte parity protection (ft = 1), TMR registers (ft = 2), Parity, no DMR (ft = 3, detect only), SECDED BCH (ft = 4, max 57 data bits)) or technology specific protection (ft = 5, max 32 data bits). If set to 0, no protection is implemented.	0 - 5	0
abits	Address bits for the internal FIFO. FIFO depth is calculated as 2^abits	0 -10	4
dbits	Bus master front end data width and Internal FIFO data width	32 - 128	32
max_burst_length	Maximum burst length	2 - 256	256
en_bm1	Enable Bus Master Interface 1	0 - 1	0
en_timer	Enable timeout timer	0 - 1	0
en_acc	Enable accelerator. Possible options are no accelerator enabled (en_acc = 0), AES256 encryption accelerator enabled (en_acc = 1), SHA256 authentication accelerator enabled (en_acc = 2) or AES256+SHA256 encryption+authentication accelerators enabled (en_acc = 3).	0 - 4	0

51.15 Signal descriptions

Table 740 shows the interface signals of GRDMAC2(VHDL ports).

Table 740. Signal description

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBMI0	*	Input	AHB master input signals	-
AHBMO0	*	Output	AHB master output signals	-
AHBMI1	*	Input	AHB master input signals	-
AHBMO1	*	Output	AHB master output signals	-
TRIGGER[63:0]	-	Input	Trigger input	-

* See GRLIB IP Library User's Manual

51.16 Library dependencies

Table 741 shows the libraries used when instantiating GRDMAC2 (VHDL libraries).

Table 741. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GRLIB	GENERIC BM PKG	Signals, components	Package including the Bus Master Interface and the bridge for converting to AHB
GAISLER	GRDMAC2	Signals, components	GRDMAC2 component and signal declarations

51.17 Instantiation

This example shows how GRDMAC2 can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.stdlib.all;
library gaisler;
use gaisler.grdmac2_pkg.all;

entity grdmac2_example is
generic (
  tech: in integer := 0
);
port (
-- Input and output Ports
);
end entity grdmac2_example;

architecture rtl of grdmac2_example is

  -- Signal declarations
  -- Reset and clock signals
  signal clk : std_ulogic;
  signal rstn : std_ulogic;

  -- APB interface signals
```

```
  signal apbi : apb_slv_in_type;
  signal apbo : apb_slv_out_type := (others => apb_none);

  -- AHB master interface signals
  signal ahbmi0 : ahb_mst_in_type;
  signal ahbmo0 : ahb_mst_out_type := (others => ahbm_none);
  signal ahbmi1 : ahb_mst_in_type;
  signal ahbmo1 : ahb_mst_out_type := (others => ahbm_none);

  -- Input trigger signal
  signal trigger : std_logic_vector(63 downto 0);

begin
  -- AMBA components are instantiated here
  ...

  -- GRDMAC2
  u2 : grdmac2_ahb
  generic map (
     tech => tech,
     pindex => 0,
     paddr => 16#010#,
     pmask => 16#FFE#,
     pirq => 0,
     dbits => 32,
     en_bm1 => 1,
     hindex0 => 0,
     hindex1 => 1,
     ft => 0,
     abits => 4,
     en_timer => 1
     )
  port map (
     rstn => rstn,
     clk => clk,
     apbi => apbi,
     apbo => apbo,
     ahbmi0 => ahbmi0,
     ahbmo0 => ahbmo0,
     ahbmi1 => ahbmi1,
     ahbmo1 => ahbmo1,
     trigger => trigger(63 downto 0)
     );
  end architecture rtl;
```

52 GRECC - Elliptic Curve Cryptography

52.1 Overview

Elliptic Curve Cryptography (ECC) is used as a public key mechanism. The computational burden that is inhibited by ECC is less than the one of RSA. ECC provides the same level of security as RSA but with a significantly shorter key length. ECC is well suited for application in mobile communication.

The GRECC core implements encryption and decryption for an elliptic curve based on 233-bit key and point lengths. The implemented curve is denoted as *sect233r1* or *B-233.*

The *sect233r1* elliptic curve domain parameters are specified in the “Standards for Efficient Cryptography (SEC) - SEC2: Recommended Elliptic Curve Domain Parameters” document. The document is established by the Standards for Efficient Cryptography Group (SECG).

The *B-233* elliptic curve domain parameters are specified in the “Digital Signature Standard (DSS)” document, Federal Information Processing Standards (FIPS) Publication 186-2. The document is established by the National Institute of Standards and Technology (NIST).

The GRECC can be used with algorithms such as:

- Elliptic Curve Digital Signature Algorithm DSA (ECDSA), which appears in FIPS 186-2, IEEE 1363-2000 and ISO/IEC 15946-2
- Elliptic Curve El Gamal Method (key exchange protocol)
- Elliptic Curve Diffie-Hellman (ECDH) (key agreement protocol)

The core provides the following internal AMBA APB slave interface, with sideband signals as per [GRLIB] including:

- interrupt bus
- configuration information
- diagnostic information

The core can be partition in the following hierarchical elements:

- Elliptic Curve Cryptography (ECC) core
- AMBA APB slave
- GRLIB plug&play wrapper

Note that the core can also be used without the GRLIB plug&play information.

52.2 Operation

Elliptic Curve Cryptography (ECC) is an asymmetric cryptographic approach (also known as public key cryptography) that applies different keys for encryption and decryption. The most expensive operation during both encryption and decryption is the elliptic curve point multiplication. Hereby, a point on the elliptic curve is multiplied with a long integer ($k*P$ multiplication). The bit sizes of the coordinates of the point $P=(x, y)$ and the factor k have a length of hundreds of bits.

In this implementation the key and the point lengths are 233 bit, so that for every key there are 8 write cycles necessary and for every point (consisting of x and y) there are 16 write cycles necessary. After at least 16700 clock cycles the result can be read out.

The key is input via eight registers. The input point $P_{in}=(x, y)$ is written via eight registers for x and eight registers for y. After the last y input register is written, the encryption or decryption is started. The progress can be observed via the status register. When the operation is completed, an interrupt is generated. The output point $P_{out}=(x, y)$ is then read out via eight registers for x and eight registers for y.

52.3 Advantages

The main operation in ECC is the k*P multiplication. One k*P multiplication requires about 1500 field multiplications in the base field, which is the most expensive base operation. The complexity of a field multiplication can be reduced by applying the Karatsuba method. Normally the Karatsuba approach is applied recursively. The GRECC core includes an iterative implementation of the Karatsuba method which allows to realize area efficient hardware accelerators for the *k*P* multiplication. Hardware accelerators which are realized applying an iterative approach need up to 60 per cent less area and about 30 per cent less energy per multiplication than the recursive variants.

52.4 Background

The Standards for Efficient Cryptography Group (SECG) was initiated by Certicom Corporation to address the difficulty vendors and users face when building and deploying interoperable security solutions. The SECG is a broad international coalition comprised of leading technology companies and key industry players in the information security industry. One of the goals is to enable the effective incorporation of Elliptic Curve Cryptographic (ECC) technology into these various cryptographic solutions.

The Standards for Efficient Cryptography Group (SECG) has develop two sets of documents. The first set, under the name SEC, specifies interoperable cryptographic technologies and solutions. The second set, Guidelines for Efficient Cryptography (GEC), provides background information on elliptic curve cryptography and recommendations for ECC parameter and curve selection.

The Federal Information Processing Standards Publication Series of the National Institute of Standards and Technology (NIST) is the official series of publications relating to standards and guidelines adopted under the provisions of the Information Technology Management Reform Act.

This Digital Signature Standard (DSS) specifies a suite of algorithms which can be used to generate a digital signature. Digital signatures are used to detect unauthorized modifications to data and to authenticate the identity of the signatory. In addition, the recipient of signed data can use a digital signature in proving to a third party that the signature was in fact generated by the signatory. This is known as nonrepudiation since the signatory cannot, at a later time, repudiate the signature.

52.5 233-bit elliptic curve domain parameters

The core implements the 233-bit elliptic curve domain parameters *sect233r1,* or the equivalent *B-233*, which are verifiably random parameters. The following specification is established in “Standards for Efficient Cryptography (SEC) - SEC 2: Recommended Elliptic Curve Domain Parameters”. The verifiably random elliptic curve domain parameters over F_{2m} are specified by the septuple T = (m; f (x); a; b; G; n; h) where m = 233 and the representation of $F_{2}233$ is defined by:

$f(x) = x^{233}+x^{74}+1$

The curve E: $y^2+xy = x^3+ax^2+b$ over F_{2m} is defined by:

a = 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000001

b = 0066 647EDE6C 332C7F8C 0923BB58 213B333B 20E9CE42 81FE115F 7D8F90AD

The base point G in compressed form is:

G = 0300FA C9DFCBAC 8313BB21 39F1BB75 5FEF65BC 391F8B36 F8F8EB73 71FD558B

and in uncompressed form is:

G = 04 00FAC9DF CBAC8313 BB2139F1 BB755FEF 65BC391F 8B36F8F8
EB7371FD 558B0100 6A08A419 03350678 E58528BE BF8A0BEF F867A7CA
36716F7E 01F81052

Finally the order n of G and the cofactor are:

n = 0100 00000000 00000000 00000000 0013E974 E72F8A69 22031D26 03CFE0D7

h = 02

52.6 Throughput

The data throughput for the GRECC core is around 233/16700 bits per clock cycle, i.e. approximately 13.9 kbits per MHz.

The underlaying EEC core has been implemented in a dual crypto chip on 250 nm technology as depicted in the figure below. The throughput at 33 MHz operating frequency was 850 kbit/s, the power consumption was 56,8 mW, and the size was 48,5 kgates.

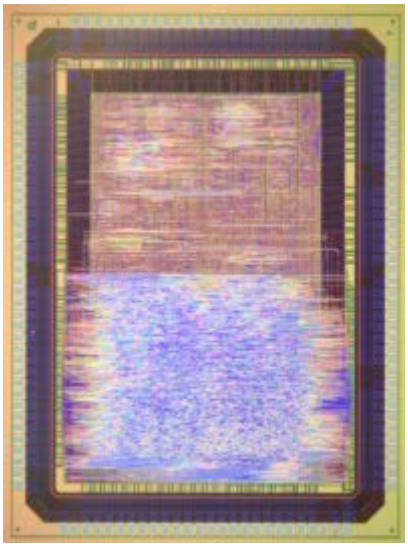


Figure 139. Dual Crypto Chip

52.7 Characteristics

The GRECC core has been synthesized for a Xilinx Virtex-2 XC2V6000-4 devices with the following results:

- LUTs: 12850 (19%)
- Frequency:93 MHz

52.8 Registers

The core is programmed through registers mapped into APB address space.

Table 742. GRECC registers

APB address offset	Register
0x20	Key 0 Register
0x24	Key 1 Register
0x28	Key 2 Register
2C	Key 3 Register
0x30	Key 4 Register
0x34	Key 5 Register
0x38	Key 6 Register
0x3C	Key 7 Register
0x40	Point X Input 0 Register
0x044	Point X Input 1 Register
0x048	Point X Input 2 Register
0x04C	Point X Input 3 Register
0x050	Point X Input 4 Register
0x054	Point X Input 5 Register
0x58	Point X Input 6 Register
0x5C	Point X Input 7 Register
0x60	Point Y Input 0 Register
0x64	Point Y Input 1 Register
0x68	Point Y Input 2 Register
0x6C	Point Y Input 3 Register
0x70	Point Y Input 4 Register
0x74	Point Y Input 5 Register
0x78	Point Y Input 6 Register
0x7C	Point Y Input 7 Register
0xA0	Point X Output 0 Register
0xA4	Point X Output 1 Register
0xA8	Point X Output 2 Register
0xAC	Point X Output 3 Register
0xB0	Point X Output 4 Register
0xB4	Point X Output 5 Register
0xB8	Point X Output 6 Register
0xBC	Point X Output 7 Register
0xC0	Point Y Output 0 Register
0xC4	Point Y Output 1 Register
0xC8	Point Y Output 2 Register
0xCC	Point Y Output 3 Register
0xD0	Point Y Output 4 Register
0xD4	Point Y Output 5 Register
0xD8	Point Y Output 6 Register
0xDC	Point Y Output 7 Register
0xFC	Status Register

52.8.1 Key 0 to 7 Registers

*Table 743.*0x20 - KEY0 - Key 0 Register (least significant)

31	0
KEY(31 downto 0)	
0	
w	

*Table 744.*0x24 - KEY1 - Key 1 Register

31	0
KEY(63 downto32)	
0	
w	

*Table 745.*0x28 - KEY2 - Key 2 Register

31	0
KEY(95 downto 64)	
0	
w	

*Table 746.*0x2C - KEY3 - Key 3 Register

31	0
KEY(127 downto 96)	
0	
w	

*Table 747.*0x30 - KEY4 - Key 4 Register

31	0
KEY(159 downto 128)	
0	
w	

*Table 748.*0x34 - KEY5 - Key 5 Register

31	0
KEY(191 downto 160)	
0	
w	

*Table 749.*0x38 - KEY6 - Key 6 Register

31	0
KEY(223 downto 192)	
0	
w	

Table 750. 0x3C - KEY7 - Key 7 Register (most significant)

31 9	8 0
RESERVED	KEY(232 downto 224)
0	0
-	w

52.8.2 Point X Input 0 to 7 Registers

Table 751. 0x40 - PXI0 - Point X Input 0 Register (least significant)

31	0
X(31 downto 0)	
0	
w	

Table 752. 0x44 - PXI1 - Point X Input 1 Register

31	0
X(63 downto32)	
0	
w	

Table 753. 0x48 - PXI2 - Point X Input 2 Register

31	0
X(95 downto 64)	
0	
w	

Table 754. 0x4C - PXI3 - Point X Input 3 Register

31	0
X(127 downto 96)	
0	
w	

Table 755. 0x50 - PXI4 - Point X Input 4 Register

31	0
X(159 downto 128)	
0	
w	

Table 756. 0x54 - PXI5 - Point X Input 5 Register

31	0
X(191 downto 160)	
0	
w	

Table 757. 0x58 - PXI6 - Point X Input 6 Register

31	0
X(223 downto 192)	
0	
w	

Table 758. 0x5C - PXI7 - Point X Input 7 Register (most significant)

31 9	8 0
RESERVED	X(232 downto 224)
0	0
-	w

52.8.3 Point Y Input 0 to 7 Registers (W)

*Table 759.*0x60 - PYI0 - Point Y Input 0 Register (least significant)

31	0
Y(31 downto 0)	

*Table 760.*0x64 - PYI1 - Point Y Input 1 Register

31	0
Y(63 downto32)	

*Table 761.*0x68 - PYI2 - Point Y Input 2 Register

31	0
Y(95 downto 64)	

*Table 762.*0x6C - PYI3 - Point Y Input 3 Register

31	0
Y(127 downto 96)	

*Table 763.*0x70 - PYI4 - Point Y Input 4 Register

31	0
Y(159 downto 128)	

*Table 764.*0x74 - PYI5 - Point Y Input 5 Register

31	0
Y(191 downto 160)	

*Table 765.*0x78 - PYI6 - Point Y Input 6 Register

31	0
Y(223 downto 192)	

*Table 766.*0x7C - PYI7 - Point Y Input 7 Register (most significant)

31 9	8 0
RESERVED	Y(232 downto 224)
	0
	w

The encryption or decryption operation is started when the Point Y Input 7 Register is written.

52.8.4 Point X Output 0 to 7 Registers (R)

Table 767. 0xA0 - PXO0 - Point X Output 0 Register (least significant)

31	0
X(31 downto 0)	
NR	
r	

Table 768. 0xA4 - PXO1 - Point X Output 1 Register

31	0
X(63 downto32)	
NR	
r	

Table 769. 0xA8 - PXO2 - Point X Output 2 Register

31	0
X(95 downto 64)	
NR	
r	

Table 770. 0xAC - PXO3 - Point X Output 3 Register

31	0
X(127 downto 96)	
NR	
r	

Table 771. 0xB0 - PXO4 - Point X Output 4 Register

31	0
X(159 downto 128)	
NR	
r	

Table 772. 0xB4 - PXO5 - Point X Output 5 Register

31	0
X(191 downto 160)	
NR	
r	

Table 773. 0xB8 - PXO6 - Point X Output 6 Register

31	0
X(223 downto 192)	
NR	
r	

Table 774. 0xBC - PXO7 - Point X Output 7 Register (most significant)

31 9	8 0
RESERVED	X(232 downto 224)
-	NR
r	r

52.8.5 Point Y Output 0 to 7 Registers (R)

*Table 775.*0xC0 - PYO0 - Point Y Output 0 Register (least significant)

31	0
Y(31 downto 0)	
NR	
r	

*Table 776.*0xC4 - PYO1 - Point Y Output 1 Register

31	0
Y(63 downto32)	
NR	
r	

*Table 777.*0xC8 - PYO2 - Point Y Output 2 Register

31	0
Y(95 downto 64)	
NR	
r	

*Table 778.*0xCC - PYO3 - Point Y Output 3 Register

31	0
Y(127 downto 96)	
NR	
r	

*Table 779.*0xD0 - PYO4 - Point Y Output 4 Register

31	0
Y(159 downto 128)	
NR	
r	

*Table 780.*0xD4 - PYO5 - Point Y Output 5 Register

31	0
Y(191 downto 160)	
NR	
r	

*Table 781.*0xD8 - PYO6 - Point Y Output 6 Register

31	0
Y(223 downto 192)	
NR	
r	

*Table 782.*0xDC - PYO7 - Point Y Output 7 Register (most significant)

31 9	8 0
RESERVED	Y(232 downto 224)
-	NR
r	r

52.8.6 Status Register (R)

*Table 783.*0xFC - STAT - Status Register

31 1	0
.	FSM
0	1
r	r

31-1: - Unused
0: FSM 0 when ongoing, 1 when idle or ready

52.9 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x074. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

52.10 Configuration options

Table 784 shows the configuration options of the core (VHDL generics).

*Table 784.*Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB BAR	0 - 16#FFF#	0
pmask	Mask field of the APB BAR	0 - 16#FFF#	16#FFC#
pirq	Interrupt line used by the GRECC	0 - NAHBIRQ-1	0

52.11 Signal descriptions

Table 785 shows the interface signals of the core (VHDL ports).

*Table 785.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
DEBUG[10:0]	N/A	Output	Debug information	-

* see GRLIB IP Library User's Manual

Note that the ECC core can also be used without the GRLIB plug&play information. The AMBA APB signals are then provided as IEEE Std_Logic_1164 compatible scalars and vectors.

52.12 Library dependencies

Table 786 shows libraries used when instantiating the core (VHDL libraries).

*Table 786.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	CRYPTO	Component	GRECC component declarations

52.13 Instantiation

This example shows how the core can be instantiated.

```
library  ieee;
use      ieee.std_logic_1164.all;

library  grlib;
use      grlib.amba.all;

library  gaisler;
use      gaisler.crypto.all;
...
...
   signal debug: std_logic_vector(10 downto 0);
..
..
   grecc0: grecc
      generic map (
         pindex         => pindex,
         paddr          => paddr,
         pmask          => pmask,
         pirq           => pirq)
      port map (
         rstn           => rstn,
         clk            => clk,
         apbi           => apbi,
         apbo           => apbo(pindex),
         debug          => debug);
```

53 GRETH - Ethernet Media Access Controller (MAC) with EDCL support

53.1 Overview

Frontgrade Gaisler's Ethernet Media Access Controller (GRETH) provides an interface between an AMBA-AHB bus and an Ethernet network. It supports 10/100 Mbit speed in both full- and half-duplex. The AMBA interface consists of an APB interface for configuration and control and an AHB master interface which handles the dataflow. The dataflow is handled through DMA channels. There is one DMA engine for the transmitter and one for the receiver. Both share the same AHB master interface. The ethernet interface supports both the MII and RMII interfaces which should be connected to an external PHY. The GRETH also provides access to the MII Management interface which is used to configure the PHY.

Optional hardware support for the Ethernet Debug Communication Link (EDCL) protocol is also provided. This is an UDP/IP based protocol used for remote debugging.

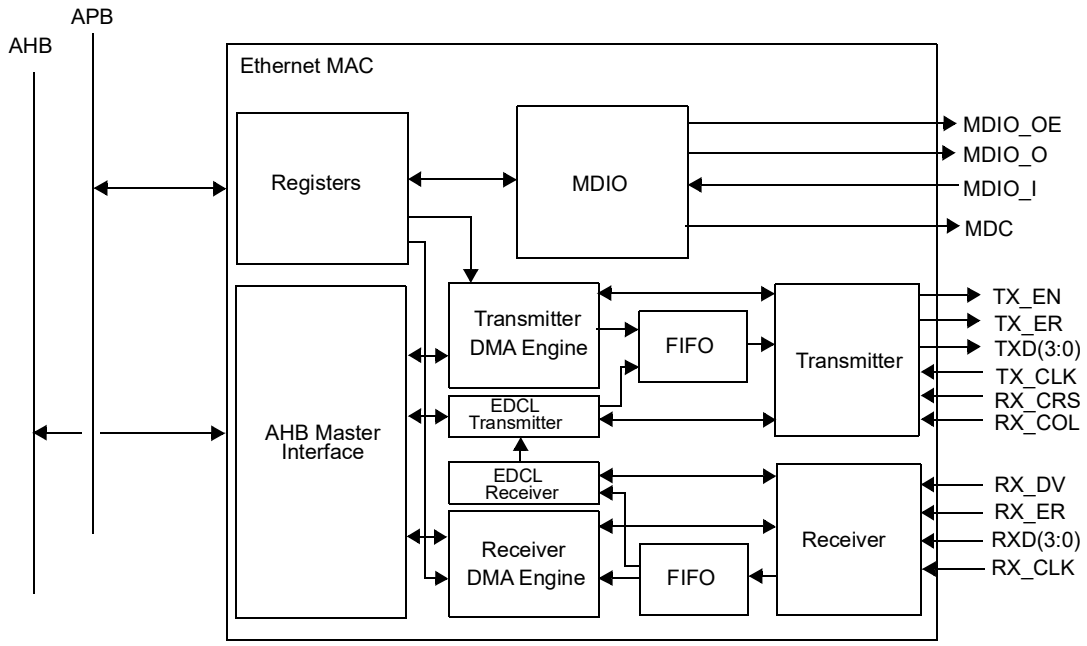


Figure 140. Block diagram of the internal structure of the GRETH.

53.2 Operation

53.2.1 System overview

The GRETH consists of 3 functional units: The DMA channels, MDIO interface and the optional Ethernet Debug Communication Link (EDCL).

The main functionality consists of the DMA channels which are used to transfer data between an AHB bus and an Ethernet network. There is one transmitter DMA channel and one Receiver DMA channel. The operation of the DMA channels is controlled through registers accessible through the APB interface.

The MDIO interface is used for accessing configuration and status registers in one or more PHYs connected to the MAC. The operation of this interface is also controlled through the APB interface.

The optional EDCL provides read and write access to an AHB bus through Ethernet. It uses the UDP, IP, ARP protocols together with a custom application layer protocol to accomplish this. The EDCL contains no user accessible registers and always runs in parallel with the DMA channels.

The Media Independent Interface (MII) is used for communicating with the PHY. There is an Ethernet transmitter which sends all data from the AHB domain on the Ethernet using the MII interface. Correspondingly, there is an Ethernet receiver which stores all data from the Ethernet on the AHB bus. Both of these interfaces use FIFOs when transferring the data streams. The GRETH also supports the RMII which uses a subset of the MII signals.

The EDCL and the DMA channels share the Ethernet receiver and transmitter.

53.2.2 Protocol support

The GRETH is implemented according to IEEE standard 802.3-2002 and IEEE standard 802.3Q-2003. There is no support for the optional control sublayer. This means that packets with type 0x8808 (the only currently defined ctrl packets) are discarded. The support for 802.3Q is optional and need to be enabled via generics.

53.2.3 Clocking

GRETH has three clock domains: The AHB clock, Ethernet receiver clock and the Ethernet transmitter clock. The ethernet transmitter and receiver clocks are generated by the external ethernet PHY, and are inputs to the core through the MII interface. The three clock domains are unrelated to each other and all signals crossing the clock regions are fully synchronized inside the core.

Both full-duplex and half-duplex operating modes are supported and both can be run in either 10 or 100 Mbit. The minimum AHB clock for 10 Mbit operation is 2.5 MHz, while 18 MHz is needed for 100 Mbit. Using a lower AHB clock than specified will lead to excessive packet loss.

53.2.4 RAM debug support

Support for debug accesses the core's internal RAM blocks can be optionally enabled using the ramdebug VHDL generic. Setting it to 1 enables accesses to the transmitter and receiver RAM buffers and setting it to 2 enables accesses to the EDCL buffer in addition to the previous two buffers.

The transmitter RAM buffer is accessed starting from APB address offset 0x10000 which corresponds to location 0 in the RAM. There are 512 32-bit wide locations in the RAM which results in the last address being 0x107FC corresponding to RAM location 511 (byte addressing used on the APB bus).

Correspondingly the receiver RAM buffer is accessed starting from APB address offset 0x20000. The addresses, width and depth is the same.

The EDCL buffers are accessed starting from address 0x30000. The number of locations depend on the configuration and can be from 256 to 16384. Each location is 32-bits wide so the maximum address is 0x3FC and 0xFFFC correspondingly.

Before any debug accesses can be made the ramdebugen bit in the control register has to be set. During this time the debug interface controls the RAM blocks and normal operations is stopped. EDCL packets are not received. The MAC transmitter and receiver could still operate if enabled but the RAM buffers would be corrupt if debug accces are made simultaneously. Thus they MUST be disabled before the RAM debug mode is enabled.

53.2.5 Multibus version

There is a version of the core which has an additional master interface that can be used for the EDCL. Otherwise this version is identical to the basic version. The additional master interface is enabled with the edclsepahb VHDL generic. Then the ethi.edclsepahb signal control whether EDCL accesses are done on the standard master interface or the additional interface. Setting the signal to '0' makes the EDCL use the standard master interface while '1' selects the additional master. This signal is only sampled at reset and changes to this signal have no effect until the next reset.

53.2.6 Endianness

The core is designed for big-endian systems.

53.3 Tx DMA interface

The transmitter DMA interface is used for transmitting data on an Ethernet network. The transmission is done using descriptors located in memory.

53.3.1 Setting up a descriptor.

A single descriptor is shown in table 787 and 788. The number of bytes to be sent should be set in the length field and the address field should point to the data. The address must be word-aligned. If the interrupt enable (IE) bit is set, an interrupt will be generated when the packet has been sent (this requires that the transmitter interrupt bit in the control register is also set). The interrupt will be generated regardless of whether the packet was transmitted successfully or not. The Wrap (WR) bit is also a control bit that should be set before transmission and it will be explained later in this section.

*Table 787.*GRETH transmit descriptor word 0 (address offset 0x0)

31 16	15	14	13	12	11	10 0
RESERVED	AL	UE	IE	WR	EN	LENGTH

31: 16	RESERVED
15	Attempt Limit Error (AL) - The packet was not transmitted because the maximum number of attempts was reached.
14	Underrun Error (UE) - The packet was incorrectly transmitted due to a FIFO underrun error.
13	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when the packet from this descriptor has been sent provided that the transmitter interrupt enable bit in the control register is set. The interrupt is generated regardless if the packet was transmitted successfully or if it terminated with an error.
12	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 8. The pointer automatically wraps to zero when the 1 kB boundary of the descriptor table is reached.
11	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
10: 0	LENGTH - The number of bytes to be transmitted.

*Table 788.*GRETH transmit descriptor word 1 (address offset 0x4)

31 2	1 0
ADDRESS	RES

31: 2	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.
1: 0	RESERVED

To enable a descriptor the enable (EN) bit should be set and after this is done, the descriptor should not be touched until the enable bit has been cleared by the GRETH.

53.3.2 Starting transmissions

Enabling a descriptor is not enough to start reception. A pointer to the memory area holding the descriptors must first be set in the GRETH. This is done in the receiver descriptor pointer register. The the beginning of the area and must start on an address that is aligned to the size of the descriptor table. The size of the descriptor table can be determined from the formula: STS.NRD*8. The STS.NRD field shows the number of entries in the descriptor table, and each descriptor size is 8 bytes.

Bits 31 to STS.NRD+10 hold the base address of descriptor area while bits STS.NRD+9 to 3 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the GRETH the pointer field is incremented by 8 to point at the next descriptor. The pointer will automatically wrap back to zero when the upper boundary has been reached. The WR bit in the descriptors can be set to make the pointer wrap back to zero before the upper boundary.The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when a transmission is active.

The final step to activate the transmission is to set the transmit enable bit in the control register. This tells the GRETH that there are more active descriptors in the descriptor table. This bit should always be set when new descriptors are enabled, even if transmissions are already active. The descriptors must always be enabled before the transmit enable bit is set.

53.3.3 Descriptor handling after transmission

When a transmission of a packet has finished, status is written to the first word in the corresponding descriptor. The Underrun Error bit is set if the FIFO became empty before the packet was completely transmitted while the Attempt Limit Error bit is set if more collisions occurred than allowed. The packet was successfully transmitted only if both of these bits are zero. The other bits in the first descriptor word are set to zero after transmission while the second word is left untouched.

The enable bit should be used as the indicator when a descriptor can be used again, which is when it has been cleared by the GRETH. There are three bits in the GRETH status register that hold transmission status. The Transmitter Error (TE) bit is set each time an transmission ended with an error (when at least one of the two status bits in the transmit descriptor has been set). The Transmitter Interrupt (TI) is set each time a transmission ended successfully.

The transmitter AHB error (TA) bit is set when an AHB error was encountered either when reading a descriptor or when reading packet data. Any active transmissions were aborted and the transmitter was disabled. The transmitter can be activated again by setting the transmit enable register.

53.3.4 Setting up the data for transmission

The data to be transmitted should be placed beginning at the address pointed by the descriptor address field. The GRETH does not add the Ethernet address and type fields so they must also be stored in the data buffer. The 4 B Ethernet CRC is automatically appended at the end of each packet. Each descriptor will be sent as a single Ethernet packet. If the size field in a descriptor is greater than defined by maxsize generic + header size bytes, the packet will not be sent.

53.4 Rx DMA interface

The receiver DMA interface is used for receiving data from an Ethernet network. The reception is done using descriptors located in memory.

53.4.1 Setting up descriptors

A single descriptor is shown in table 789 and 790. The address field should point to a word-aligned buffer where the received data should be stored. The GRETH will never store more than defined by the maxisize generic + header size bytes to the buffer. If the interrupt enable (IE) bit is set, an interrupt will be generated when a packet has been received to this buffer (this requires that the receiver interrupt bit in the control register is also set). The interrupt will be generated regardless of whether the packet was received successfully or not. The Wrap (WR) bit is also a control bit that should be set before the descriptor is enabled and it will be explained later in this section.

*Table 789.*GRETH receive descriptor word 0 (address offset 0x0)

31 27	26	25 19	18	17	16	15	14	13	12	11	10 0
RESERVED	MC	RESERVED	LE	OE	CE	FT	AE	IE	WR	EN	LENGTH

Table 789. GRETH receive descriptor word 0 (address offset 0x0)

31: 27	RESERVED
26	Multicast address (MC) - The destination address of the packet was a multicast address (not broadcast).
25: 19	RESERVED
18	Length error (LE) - The length/type field of the packet did not match the actual number of received bytes.
17	Overrun error (OE) - The frame was incorrectly received due to a FIFO overrun.
16	CRC error (CE) - A CRC error was detected in this frame.
15	Frame too long (FT) - A frame larger than the maximum size was received. The excessive part was truncated.
14	Alignment error (AE) - An odd number of nibbles were received.
13	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when a packet has been received to this descriptor provided that the receiver interrupt enable bit in the control register is set. The interrupt is generated regardless if the packet was received successfully or if it terminated with an error.
12	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 8. The pointer automatically wraps to zero when the 1 kB boundary of the descriptor table is reached.
11	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
10: 0	LENGTH - The number of bytes received to this descriptor.

Table 790. GRETH receive descriptor word 1 (address offset 0x4)

31	2	1	0
ADDRESS			RES

31: 2	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.
1: 0	RESERVED

53.4.2 Starting reception

Enabling a descriptor is not enough to start reception. A pointer to the memory area holding the descriptors must first be set in the GRETH. This is done in the receiver descriptor pointer register. The the beginning of the area and must start on an address that is aligned to the size of the descriptor table. The size of the descriptor table can be determined from the formula: STS.NRD*8. The STS.NRD field shows the number of entries in the descriptor table, and each descriptor size is 8 bytes.

Bits 31 to STS.NRD+10 hold the base address of descriptor area while bits STS.NRD+9 to 3 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the GRETH the pointer field is incremented by 8 to point at the next descriptor. The pointer will automatically wrap back to zero when the upper boundary has been reached. The WR bit in the descriptors can be set to make the pointer wrap back to zero before the upper boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when reception is active.

The final step to activate reception is to set the receiver enable bit in the control register. This will make the GRETH read the first descriptor and wait for an incoming packet.

53.4.3 Descriptor handling after reception

The GRETH indicates a completed reception by clearing the descriptor enable bit. The other control bits (WR, IE) are also cleared. The number of received bytes is shown in the length field. The parts of the Ethernet frame stored are the destination address, source address, type and data fields. Bits 17-14 in the first descriptor word are status bits indicating different receive errors. All four bits are zero after a reception without errors. The status bits are described in table 789.

Packets arriving that are smaller than the minimum Ethernet size of 64 B are not considered as a reception and are discarded. The current receive descriptor will be left untouched an used for the first packet arriving with an accepted size. The TS bit in the status register is set each time this event occurs.

If a packet is received with an address not accepted by the MAC, the IA status register bit will be set.

Packets larger than maximum size cause the FT bit in the receive descriptor to be set. The length field is not guaranteed to hold the correct value of received bytes. The counting stops after the word containing the last byte up to the maximum size limit has been written to memory.

The address word of the descriptor is never touched by the GRETH.

53.4.4 Reception with AHB errors

If an AHB error occurs during a descriptor read or data store, the Receiver AHB Error (RA) bit in the status register will be set and the receiver is disabled. The current reception is aborted. The receiver can be enabled again by setting the Receive Enable bit in the control register.

53.4.5 Accepted MAC addresses

In the default configuration the core receives packets with either the unicast address set in the MAC address register or the broadcast address. Multicast support can also be enabled and in that case a hash function is used to filter received multicast packets. A 64-bit register, which is accessible through the APB interface, determines which addresses should be received. Each address is mapped to one of the 64 bits using the hash function and if the bit is set to one the packet will be received. The address is mapped to the table by taking the 6 least significant bits of the 32-bit Ethernet crc calculated over the destination address of the MAC frame. A bit in the receive descriptor is set if a packet with a multicast address has been received to it.

53.5 MDIO Interface

The MDIO interface provides access to PHY configuration and status registers through a two-wire interface which is included in the MII interface. The GRETH provided full support for the MDIO interface. If it is not needed in a design it can be removed with a VHDL generic.

The MDIO interface can be used to access from 1 to 32 PHY containing 1 to 32 16-bit registers. A read transfer i set up by writing the PHY and register addresses to the MDIO Control register and setting the read bit. This caused the Busy bit to be set and the operation is finished when the Busy bit is cleared. If the operation was successful the Linkfail bit is zero and the data field contains the read data. An unsuccessful operation is indicated by the Linkfail bit being set. The data field is undefined in this case.

A write operation is started by writing the 16-bit data, PHY address and register address to the MDIO Control register and setting the write bit. The operation is finished when the busy bit is cleared and it was successful if the Linkfail bit is zero.

53.5.1 PHY interrupts

The core also supports status change interrupts from the PHY. A level sensitive interrupt signal can be connected on the mdint input. The mdint_pol vhdl generic can be used to select the polarity. The PHY status change bit in the status register is set each time an event is detected in this signal. If the PHY status interrupt enable bit is set at the time of the event the core will also generate an interrupt on the AHB bus.

53.6 Ethernet Debug Communication Link (EDCL)

The EDCL provides access to an on-chip AHB bus through Ethernet. It uses the UDP, IP and ARP protocols together with a custom application layer protocol. The application layer protocol uses an

ARQ algorithm to provide reliable AHB instruction transfers. Through this link, a read or write transfer can be generated to any address on the AHB bus. The EDCL is optional and must be enabled with a generic.

53.6.1 Operation

The EDCL receives packets in parallel with the MAC receive DMA channel. It uses a separate MAC address which is used for distinguishing EDCL packets from packets destined to the MAC DMA channel. The EDCL also has an IP address which is set through generics. Since ARP packets use the Ethernet broadcast address, the IP-address must be used in this case to distinguish between EDCL ARP packets and those that should go to the DMA-channel. Packets that are determined to be EDCL packets are not processed by the receive DMA channel.

When the packets are checked to be correct, the AHB operation is performed. The operation is performed with the same AHB master interface that the DMA-engines use. The replies are automatically sent by the EDCL transmitter when the operation is finished. It shares the Ethernet transmitter with the transmitter DMA-engine but has higher priority.

53.6.2 EDCL protocols

The EDCL accepts Ethernet frames containing IP or ARP data. ARP is handled according to the protocol specification with no exceptions.

IP packets carry the actual AHB commands. The EDCL expects an Ethernet frame containing IP, UDP and the EDCL specific application layer parts. Table 791 shows the IP packet required by the EDCL. The contents of the different protocol headers can be found in TCP/IP literature.

Table 791. The IP packet expected by the EDCL.

Ethernet Header	IP Header	UDP Header	2 B Offset	4 B Control word	4 B Address	Data 0 - 242 4B Words	Ethernet CRC

The following is required for successful communication with the EDCL: A correct destination MAC address as set by the generics, an Ethernet type field containing 0x0806 (ARP) or 0x0800 (IP). The IP-address is then compared with the value determined by the generics for a match. The IP-header checksum and identification fields are not checked. There are a few restrictions on the IP-header fields. The version must be four and the header size must be 5 B (no options). The protocol field must always be 0x11 indicating a UDP packet. The length and checksum are the only IP fields changed for the reply.

The EDCL only provides one service at the moment and it is therefore not required to check the UDP port number. The reply will have the original source port number in both the source and destination fields. UDP checksum are not used and the checksum field is set to zero in the replies.

The UDP data field contains the EDCL application protocol fields. Table 792 shows the application protocol fields (data field excluded) in packets received by the EDCL. The 16-bit offset is used to align the rest of the application layer data to word boundaries in memory and can thus be set to any value. The R/W field determines whether a read (0) or a write(1) should be performed. The length

Table 792. The EDCL application layer fields in received frames.

16-bit Offset	14-bit Sequence number	1-bit R/W	10-bit Length	7-bit Unused

field contains the number of bytes to be read or written. If R/W is one the data field shown in table 791 contains the data to be written. If R/W is zero the data field is empty in the received packets. Table 793 shows the application layer fields of the replies from the EDCL. The length field is always zero for replies to write requests. For read requests it contains the number of bytes of data contained in the data field.

*Table 793.*The EDCL application layer fields in transmitted frames.

16-bit Offset	14-bit sequence number	1-bit ACK/NAK	10-bit Length	7-bit Unused

The EDCL implements a Go-Back-N algorithm providing reliable transfers. The 14-bit sequence number in received packets are checked against an internal counter for a match. If they do not match, no operation is performed and the ACK/NAK field is set to 1 in the reply frame. The reply frame contains the internal counter value in the sequence number field. If the sequence number matches, the operation is performed, the internal counter value is stored in the sequence number field, the ACK/NAK field is set to 0 in the reply and the internal counter is incremented, . The length field is always set to 0 for ACK/NAK=1 frames. The unused field is not checked and is copied to the reply. It can thus be set to hold for example some extra identifier bits if needed.

53.6.3 EDCL IP and Ethernet address settings

The default value of the EDCL IP and MAC addresses are set by `ipaddrh, ipaddrl, macaddrh and macaddrl` generics. The IP address and MAC address can later be changed by software. To allow several EDCL enabled GRETH controllers on the same sub-net, the 4 LSB bits of the IP and MAC address can optionally be set by an input signal. This is enabled by setting the `edcl` generic = 2, and driving the 4-bit LSB value on ethi.edcladdr.

53.6.4 EDCL buffer size

The EDCL has a dedicated internal buffer memory which stores the received packets during processing. The size of this buffer is configurable with a VHDL generic to be able to obtain a suitable compromise between throughput and resource utilization in the hardware. Table 794 lists the different buffer configurations. For each size the table shows how many concurrent packets the EDCL can handle, the maximum size of each packet including headers and the maximum size of the data payload. Sending more packets before receiving a reply than specified for the selected buffer size will lead to dropped packets. The behavior is unspecified if sending larger packets than the maximum allowed.

*Table 794.*EDCL buffer sizes

Total buffer size (kB)	Number of packet buffers	Packet buffer size (B)	Maximum data payload (B)
1	4	256	200
2	4	512	456
4	8	512	456
8	8	1024	968
16	16	1024	968
32	32	1024	968
64	64	1024	968

53.7 Media Independent Interfaces

There are several interfaces defined between the MAC sublayer and the Physical layer. The GRETH supports two of them: The Media Independent Interface (MII) and the Reduced Media Independent Interface (RMII).

The MII was defined in the 802.3 standard and is most commonly supported. The ethernet interface have been implemented according to this specification. It uses 16 signals.

To support lower speed where the operation and clock frequency of the core and phy remains unchanged i.e. running at 10Mb/s when the IP is configured for 100Mb/s speed enable signals should be created to mimic the desired bit rate.

When operating at 10Mb/s, every byte of the MAC frame is repeated 10 clock periods to achieve the correct bit rate. The GRETH_GBIT core does not take care of this operation and enable signals with toggling frequency of the correct bit rate needs to be created.

The RMII was developed to meet the need for an interface allowing Ethernet controllers with smaller pin counts. It uses 6 (7) signals which are a subset of the MII signals. Table 795 shows the mapping between the RMII signals and the GRLIB MII interface.

*Table 795.*Signal mappings between RMII and the GRLIB MII interface.

RMII	MII
txd[1:0]	txd[1:0]
tx_en	tx_en
crs_dv	rx_crs
rxd[1:0]	rxd[1:0]
ref_clk	rmii_clk
rx_er	not used

53.8 AXI support

The core is designed for an AMBA system but can be adapted for AXI using the AHBM2AXI adapter.

53.9 Registers

The core is programmed through registers mapped into APB address space.

*Table 796.*GRETH registers

APB address offset	**Register**
0x0	Control register
0x4	Status/Interrupt-source register
0x8	MAC Address MSB
0xC	MAC Address LSB
0x10	MDIO Control/Status
0x14	Transmit descriptor pointer
0x18	Receiver descriptor pointer
0x1C	EDCL IP
0x20	Hash table msb
0x24	Hash table lsb
0x28	EDCL MAC address MSB
0x2C	EDCL MAC address LSB
0x10000 - 0x107FC	Transmit RAM buffer debug access
0x20000 - 0x207FC	Receiver RAM buffer debug access
0x30000 - 0x3FFFC	EDCL buffer debug access

53.9.1 Control Register

*Table 797.*0x00 - CTRL - GRETH control register

31	30 28	27	26	25	24 15	14	13	12	11	10	9 8	7	6	5	4	3	2	1	0
EA	BS		MA	MC	RESERVED	ED	RD	DD	ME	PI	RES	SP	RS	PM	FD	RI	TI	RE	TE
*	*	*	*	*	0	*	*	0	0	0	0	1	0	0	0	0	0	0	0
r	r	r	r	r	r	rw	rw	rw	rw	rw	r	rw	wc	rw	rw	rw	rw	rw	rw

31 EDCL available (EA) - Set to one if the EDCL is available.

30: 28 EDCL buffer size (BS) - Shows the amount of memory used for EDCL buffers. 0 = 1 kB, 1 = 2 kB, ..., 6 = 64 kB.

27 RESERVED

26 MDIO interrupts available (MA) - Set to one when the core supports mdio interrupts. Read only.

25 Multicast available (MC) - Set to one when the core supports multicast address reception. Read only.

24: 15 RESERVED

14 EDCL Disable (ED) - Set to one to disable the EDCL and zero to enable it. Reset value taken from the ethi.edcldisable signal. Only available if the EDCL hardware is present in the core.

13 RAM debug enable (RD) - Set to one to enable the RAM debug mode. Reset value: '0'. Only available if the VHDL generic ramdebug is nonzero.

12 Disable duplex detection (DD) - Disable the EDCL speed/duplex detection FSM. If the FSM cannot complete the detection the MDIO interface will be locked in busy mode. If software needs to access the MDIO the FSM can be disabled here and as soon as the MDIO busy bit is 0 the interface is available. Note that the FSM cannot be reenabled again.

11 Multicast enable (ME) - Enable reception of multicast addresses. Reset value: '0'.

10 PHY status change interrupt enable (PI) - Enables interrupts for detected PHY status changes.

9: 8 RESERVED

7 Speed (SP) - Sets the current speed mode. 0 = 10 Mbit, 1 = 100 Mbit. Only used in RMII mode (rmii = 1). A default value is automatically read from the PHY after reset. Reset value: '1'.

6 Reset (RS) - A one written to this bit resets the GRETH core. Self clearing. No other accesses should be done .to the slave interface other than polling this bit until it is cleared.

5 Promiscuous mode (PM) - If set, the GRETH operates in promiscuous mode which means it will receive all packets regardless of the destination address. Reset value: '0'.

4 Full duplex (FD) - If set, the GRETH operates in full-duplex mode otherwise it operates in half-duplex. Reset value: '0'.

3 Receiver interrupt (RI) - Enable Receiver Interrupts. An interrupt will be generated each time a packet is received when this bit is set. The interrupt is generated regardless if the packet was received successfully or if it terminated with an error. Reset value: '0'.

2 Transmitter interrupt (TI) - Enable Transmitter Interrupts. An interrupt will be generated each time a packet is transmitted when this bit is set. The interrupt is generated regardless if the packet was transmitted successfully or if it terminated with an error. Reset value: '0'.

1 Receive enable (RE) - Should be written with a one each time new descriptors are enabled. As long as this bit is one the GRETH will read new descriptors and as soon as it encounters a disabled descriptor it will stop until RE is set again. This bit should be written with a one after the new descriptors have been enabled. Reset value: '0'.

0 Transmit enable (TE) - Should be written with a one each time new descriptors are enabled. As long as this bit is one the GRETH will read new descriptors and as soon as it encounters a disabled descriptor it will stop until TE is set again. This bit should be written with a one after the new descriptors have been enabled. Reset value: '0'.

53.9.2 Status Register

*Table 798.*0x04 - STAT - GRETH status register

31 28	27 24	23 9	8	7	6	5	4	3	2	1	0
RESERVED	NRD	RESERVED	PS	IA	TS	TA	RA	TI	RI	TE	RE
0	*	0	0	0	0	NR	NR	NR	NR	NR	NR
r	r	r	wc	wc	wc	wc	wc	wc	wc	wc	wc

*Table 798.*0x04 - STAT - GRETH status register

27: 24	Number of receive descriptors (NRD) - Shows the size of the DMA descriptor table. 0b0000 = 128 0b0001 = 256, 0b0010 = 512, 0b0010 = 1024, 0b0100 = 2048, 0b0101 = 4096, 0b0110 = 8192, 0b0111 = 16384, 0b1000 = 32768, 0b1001 = 65536
8	PHY status changes (PS) - Set each time a PHY status change is detected.
7	Invalid address (IA) - A packet with an address not accepted by the MAC was received. Cleared when written with a one. Reset value: ‘0’.
6	Too small (TS) - A packet smaller than the minimum size was received. Cleared when written with a one. Reset value: ‘0’.
5	Transmitter AHB error (TA) - An AHB error was encountered in transmitter DMA engine. Cleared when written with a one. Not Reset.
4	Receiver AHB error (RA) - An AHB error was encountered in receiver DMA engine. Cleared when written with a one. Not Reset.
3	Transmitter interrupt (TI) - A packet was transmitted without errors. Cleared when written with a one. Not Reset.
2	Receiver interrupt (RI) - A packet was received without errors. Cleared when written with a one. Not Reset.
1	Transmitter error (TE) - A packet was transmitted which terminated with an error. Cleared when written with a one. Not Reset.
0	Receiver error (RE) - A packet has been received which terminated with an error. Cleared when written with a one. Not Reset.

53.9.3 MAC Address MSB

*Table 799.*0x08 - MACMSB - GRETH MAC address MSB.

31 16	15 0
RESERVED	Bit 47 downto 32 of the MAC address
	NR
	rw

31: 16 RESERVED

15: 0 The two most significant bytes of the MAC Address. Not Reset.

53.9.4 MAC Address LSB

*Table 800.*0x0C - MACLSB - GRETH MAC address LSB.

31 0
Bit 31 downto 0 of the MAC address
NR
rw

31: 0 The four least significant bytes of the MAC Address. Not Reset.

53.9.5 MDIO ctrl/status Register

*Table 801.*0x10 - MDIO - GRETH MDIO ctrl/status register.

31 16	15 11	10 6	5 4	3	2	1	0
DATA	PHYADDR	REGADDR	RES	BU	LF	RD	WR
0	*	0		0	1	0	0
rw	rw	rw		r	r	rw	rw

31: 16 Data (DATA) - Contains data read during a read operation and data that is transmitted is taken from this field. Reset value: 0x0000.

15: 11 PHY address (PHYADDR) - This field contains the address of the PHY that should be accessed during a write or read operation. Reset value: “00000”.

10: 6 Register address (REGADDR) - This field contains the address of the register that should be accessed during a write or read operation. Reset value: “00000”.

5:4 RESERVED

3 Busy (BU) - When an operation is performed this bit is set to one. As soon as the operation is finished and the management link is idle this bit is cleared. Reset value: ‘0’.

2 Linkfail (LF) - When an operation completes (BUSY = 0) this bit is set if a functional management link was not detected. Reset value: ‘1’.

1 Read (RD) - Start a read operation on the management interface. Data is stored in the data field. Reset value: ‘0’.

0 Write (WR) - Start a write operation on the management interface. Data is taken from the Data field. Reset value: ‘0’.

53.9.6 Transmitter Descriptor Table Base Address Register

*Table 802.*0x14 - TXBASE - GRETH transmitter descriptor table base address register.

31 X+1	X 3	2 0
BASEADDR	DESCPNT	RES
NR	0	0
rw	rw	r

31: X+1 Transmitter descriptor table base address (BASEADDR) - Base address to the transmitter descriptor table.Not Reset.The value of x is given by the formula: 9 + STS.NXD

X: 3 Descriptor pointer (DESCPNT) - Pointer to individual descriptors. Automatically incremented by the Ethernet MAC. The value of x is given by the formula: 9 + STS.NXD.

2: 0 RESERVED

53.9.7 Receiver Descriptor Table Base Address Register

*Table 803.*0x18 - RXBASE - GRETH receiver descriptor table base address register.

31 X+1	X 3	2 0
BASEADDR	DESCPNT	RES
NR	0	0
rw	rw	r

31: X+1 Receiver descriptor table base address (BASEADDR) - Base address to the receiver descriptor table.Not Reset. The value of x is given by the formula: 9 + STS.NXD

X: 3 Descriptor pointer (DESCPNT) - Pointer to individual descriptors. Automatically incremented by the Ethernet MAC. The value of x is given by the formula: 9 + STS.NXD

2: 0 RESERVED

53.9.8 EDCL IP Register

*Table 804.*0x1C - EDCLIP - GRETH EDCL IP register

31 0
EDCL IP ADDRESS
*
rw

31: 0 EDCL IP address. Reset value is set with the ipaddrh and ipaddrl generics.

53.9.9 Hash Table Msb Register

*Table 805.*0x20 - HhSB - GRETH Hash table msb register

31 0
Hash table (64:32)
NR
rw

31: 0 Hash table msb. Bits 64 downto 32 of the hash table.

53.9.10 Hash Table Lsb Register

*Table 806.*0x24 - HCSB - GRETH Hash table lsb register

31	0
Hash table (64:32)	
NR	
rw	

31: 0 Hash table lsb. Bits 31downto 0 of the hash table.

53.9.11 EDCL MAC Address MSB

*Table 807.*0x28 - EMACMSB - GRETH EDCL MAC address MSB.

31 16	15 0
RESERVED	Bit 47 downto 32 of the EDCL MAC Address
	*
	rw

31: 16 RESERVED

15: 0 The two most significant bytes of the EDCL MAC Address. Hardcoded reset value set with the VHDL generic macaddrh.

53.9.12 EDCL MAC Address LSB

*Table 808.*0x2C - EMACLSB - GRETH EDCL MAC address LSB.

31	0
Bit 31 downto 0 of the EDCL MAC Address	
*	
rw	

31: 0 The 4 least significant bytes of the EDCL MAC Address. Hardcoded reset value set with the VHDL generics macaddrh and macaddrl. If the VHDL generic edcl is set to 2 bits 3 downto 0 are set with the edcladdr input signal.

53.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x1D. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

53.11 Implementation

53.11.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

53.11.2 Endianness

The core changes endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). Changing *grlib_little_endian* controls TBD

53.12 Configuration options

Table 809 shows the configuration options of the core (VHDL generics).

*Table 809.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRETH.	0 - NAHBIRQ-1	0
memtech	Memory technology used for the FIFOs.	0 - NTECH	inferred
ifg_gap	Number of ethernet clock cycles used for one interframe gap. Default value as required by the standard. Do not change unless you know what you are doing.	1 - 255	24
attempt_limit	Maximum number of transmission attempts for one packet. Default value as required by the standard.	1 - 255	16
backoff_limit	Limit on the backoff size of the backoff time. Default value as required by the standard. Sets the number of bits used for the random value. Do not change unless you know what your doing.	1 - 10	10
slot_time	Number of ethernet clock cycles used for one slot- time. Default value as required by the ethernet standard. Do not change unless you know what you are doing.	1 - 255	128
mdcscaler	Sets the divisor value use to generate the mdio clock (mdc). The mdc frequency will be clk/(2*(mdcs-caler+1)).	0 - 255	25
enable_mdio	Enable the Management interface,	0 - 1	0
fifosize	Sets the size in 32-bit words of the receiver and transmit-ter FIFOs.	4 - 32	8
nsync	Number of synchronization registers used.	1 - 2	2
edcl	Enable EDCL. 0 = disabled. 1 = enabled. 2 = enabled and 4-bit LSB of IP and ethernet MAC address pro-grammed by ethi.edcladdr, 3=in addition to features for value 2 the reset value for the EDCL disable bit is taken from the ethi.edcldisable signal instead of being hard-coded to 0. 4=in addition to features for value 2 and 3 the an option is given to disable the EDCL via external input signal.	0 - 4	0
edclbufsz	Select the size of the EDCL buffer in kB.	1 - 64	1
macaddrh	Sets the upper 24 bits of the EDCL MAC address. Not all addresses are allowed and most NICs and proto-col implementations will discard frames with illegal addresses silently. Consult network literature if unsure about the addresses.	0 - 16#FFFFFF#	16#00005E#
macaddrl	Sets the lower 24 bits of the EDCL MAC address. Not all addresses are allowed and most NICs and proto-col implementations will discard frames with illegal addresses silently. Consult network literature if unsure about the addresses.	0 - 16#FFFFFF#	16#000000#
ipaddrh	Sets the upper 16 bits of the EDCL IP address reset value.	0 - 16#FFFF#	16#C0A8#
ipaddrl	Sets the lower 16 bits of the EDCL IP address reset value.	0 - 16#FFFF#	16#0035#

*Table 809.*Configuration options

Generic	Function	Allowed range	Default
phyrstadr	Sets the reset value of the PHY address field in the MDIO register. When set to 32, the address is taken from the ethi.phyrstaddr signal.	0 - 32	0
rmii	Selects the desired PHY interface. 0 = MII, 1 = RMII.	0 - 1	0
oepol	Selects polarity on output enable (ETHO.MDIO_OE). 0 = active low, 1 = active high	0 - 1	0
mdint_pol	Selects polarity for level sensitive PHY interrupt line. 0 = active low, 1 = active high	0 - 1	0
enable_mdint	Enable mdio interrupts	0 - 1	0
multicast	Enable multicast support	0 - 1	0
ramdebug	Enables debug access to the core's RAM blocks through the APB interface. 1=enables access to the receiver and transmitter RAM buffers, 2=enables access to the EDCL buffers in addition to the functionality of value 1. Setting this generic to 2 will have no effect if the edcl generic is 0.	0 - 2	0
ehindex	AHB master index for the separate EDCL master interface. Only used if edclsepahb is 1.	0 - NAHBMST-1	0
edclsepahb	Enables separate EDCL AHB master interface. A signal determines if the separate interface or the common interface is used. Only available in the GRETH_GBIT_MB version of the core.	0 - 1	0
mdiohold	Set output hold time for MDIO in number of AHB cycles. Should be 10 ns or more.	1 - 30	1
maxsize	Set maximum length of the data field of Ethernet 802.3 frame. Values of 'maxsize' and below for this field indicate that the ethernet type field is used as the size of the payload of the Ethernet Frame while values of above 'maxsize' indicate that the field is used to represent EtherType. For 802.3q support set the length of the payload to 1504	64 - 2047	1500
gmiimode	Enable the use of receive and transmit valid signals to enter data to/from the PHY at the correct rate.	0-1	0
ft	Enable fault tolerance for receive and transmit buffers. 0 - no fault tolerance, 1 - Parity DMR, 2 - TMR.	0-2	0
edclft	Enable fault tolerance for EDCL buffers. 0 - no fault tolerance, 1 - Parity DMR, 2 - TMR. This generic only has effect if generic edcl is non-zero.	0-2	0
num_desc	Specifies the number of entries in the descriptor table	128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32728, 65536	128

53.13 Signal descriptions

Table 810 shows the interface signals of the core (VHDL ports).

*Table 810.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-

Table 810. Signal descriptions

Signal name	Field	Type	Function	Active
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
ETHI	gtx_clk	Input	Ethernet gigabit transmit clock.	-
	rmii_clk	Input	Ethernet RMII clock.	-
	tx_clk	Input	Ethernet transmit clock.	-
	tx_dv	Input	Ethernet transmitter enable	-
	rx_clk	Input	Ethernet receive clock.	-
	rxd	Input	Ethernet receive data.	-
	rx_dv	Input	Ethernet receive data valid.	High
	rx_er	Input	Ethernet receive error.	High
	rx_col	Input	Ethernet collision detected. (Asynchronous, sampled with tx_clk)	High
	rx_crs	Input	Ethernet carrier sense. (Asynchronous, sampled with tx_clk)	High
	rx_en	Input	Ethernet receiver enable.	-
	mdio_i	Input	Ethernet management data input	-
	mdint	Input	Ethernet management interrupt	-
	phyrstaddr	Input	Reset address for GRETH PHY address field.	-
	edcldis	Input	Disable EDCL functionality	-
	edcladdr	Input	Sets the four least significant bits of the EDCL MAC address and the EDCL IP address when the edcl generic is set to 2.	-
	edclsepahb	Input	Selects AHB master interface for the EDCL. ‘0’ selects the common interface and ‘1’ selects the separate interface. Only available in the GRETH_GBIT_MB version of the core when the VHDL generic edclsepahb is set to 1.	-
	edcldisable	Input	Reset value for edcl disable register bit. Setting the signal to 1 disables the EDCL at reset and 0 enables it.	-
ETHO	reset	Output	Ethernet reset (asserted when the MAC is reset).	Low
	txd	Output	Ethernet transmit data.	-
	tx_en	Output	Ethernet transmit enable.	High
	tx_er	Output	Ethernet transmit error.	High
	mdc	Output	Ethernet management data clock.	-
	mdio_o	Output	Ethernet management data output.	-
	mdio_oe	Output	Ethernet management data output enable.	Set by the oepol generic.
MTESTI**	BUF	Input	Memory BIST input signal to buffer RAM	-
	EDCL	Input	Memory BIST input signal to EDCL RAM	-
MTESTO**	BUF	Output	Memory BIST output signal from buffer RAM	-
	EDCL	Output	Memory BIST output signal from EDCL RAM	-
MTESTCLK**	N/A	Input	Memory BIST clock	-

* see GRLIB IP Library User’s Manual

** not available in FPGA releases

53.14 Library dependencies

Table 811 shows libraries used when instantiating the core (VHDL libraries).

*Table 811.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	NET	Signals, components	GRETH component declaration

53.15 Instantiation

The first example shows how the non-mb version of the core can be instantiated and the second one show the mb version.

53.15.1 Non-MB version

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.ethernet_mac.all;

entity greth_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- ethernet signals
    ethi :: in  eth_in_type;
    etho :  in  eth_out_type
    );
end;

architecture rtl of greth_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

begin

  -- AMBA Components are instantiated here
  ...

  -- GRETH
  e1 : greth
    generic map(
     hindex       => 0,
     pindex       => 12,
     paddr        => 12,
     pirq         => 12,
     memtech      => inferred,
     mdcscaler    => 50,
     enable_mdio  => 1,
     fifosize     => 32,
     nsync        => 1,
     edcl         => 1,
     edclbufsz    => 8,
     macaddrh     => 16#00005E#,
```

```
     macaddrl     => 16#00005D#,
     ipaddrh      => 16#c0a8#,
     ipaddrl      => 16#0035#)
 port map(
   rst          => rstn,
   clk          => clk,
   ahbmi        => ahbmi,
   ahbmo        => ahbmo(0),
   apbi         => apbi,
   apbo         => apbo(12),
   ethi         => ethi,
   etho         => etho
   );
end;
```

53.15.2 MB version

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.ethernet_mac.all;

entity greth_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- ethernet signals
    ethi :: in  eth_in_type;
    etho :  in  eth_out_type
    );
end;

architecture rtl of greth_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

begin

  -- AMBA Components are instantiated here
  ...

  -- GRETH
  e1 : greth_mb
    generic map(
     hindex       => 0,
     pindex       => 12,
     paddr        => 12,
     pirq         => 12,
     memtech      => inferred,
     mdcscaler    => 50,
     enable_mdio  => 1,
     fifosize     => 32,
     nsync        => 1,
     edcl         => 1,
     edclbufsz    => 8,
     macaddrh     => 16#00005E#,
     macaddrl     => 16#00005D#,
     ipaddrh      => 16#c0a8#,
     ipaddrl      => 16#0035#,
     ehindex      => 1,
     edclsepahb   => 1)
```

```
      port map(
        rst           => rstn,
        clk           => clk,
        ahbmi         => ahbmi,
        ahbmo         => ahbmo(0),
        ahbmi2        => ahbmi,
        ahbmo2        => ahbmo(1),
        apbi          => apbi,
        apbo          => apbo(12),
        ethi          => ethi,
        etho          => etho
        );
    end;
```

54 GRETH_GBIT - Gigabit Ethernet Media Access Controller (MAC) w. EDCL

54.1 Overview

Frontgrade Gaisler's Gigabit Ethernet Media Access Controller (GRETH_GBIT) provides an interface between an AMBA-AHB bus and an Ethernet network. It supports 10/100/1000 Mbit speed in both full- and half-duplex. The AMBA interface consists of an APB interface for configuration and control and an AHB master interface which handles the dataflow. The dataflow is handled through DMA channels. There is one DMA engine for the transmitter and one for the receiver. Both share the same AHB master interface.

The ethernet interface supports the MII and GMII interfaces which should be connected to an external PHY. The GRETH also provides access to the MII Management interface which is used to configure the PHY. Optional hardware support for the Ethernet Debug Communication Link (EDCL) protocol is also provided. This is an UDP/IP based protocol used for remote debugging.

Supported features for the DMA channels are Scatter Gather I/O and TCP/UDP over IPv4 checksum offloading for both receiver and transmitter.

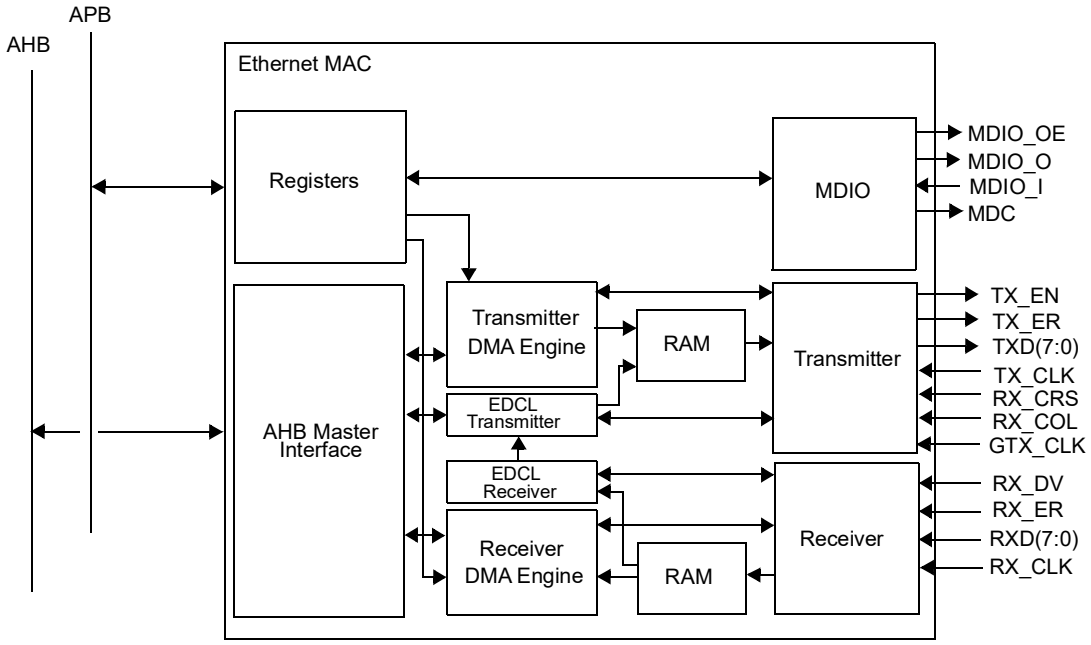


Figure 141. Block diagram of the internal structure of the GRETH_GBIT.

54.2 Operation

54.2.1 System overview

The GRETH_GBIT consists of 3 functional units: The DMA channels, MDIO interface and the optional Ethernet Debug Communication Link (EDCL).

The main functionality consists of the DMA channels which are used for transferring data between an AHB bus and an Ethernet network. There is one transmitter DMA channel and one Receiver DMA channel. The operation of the DMA channels is controlled through registers accessible through the APB interface.

The MDIO interface is used for accessing configuration and status registers in one or more PHYs connected to the MAC. The operation of this interface is also controlled through the APB interface.

The optional EDCL provides read and write access to an AHB bus through Ethernet. It uses the UDP, IP and ARP protocols together with a custom application layer protocol to accomplish this. The EDCL contains no user accessible registers and always runs in parallel with the DMA channels.

The Media Independent Interface (MII) and Gigabit Media Independent Interface (GMII) are used for communicating with the PHY. More information can be found in section 54.7.

The EDCL and the DMA channels share the Ethernet receiver and transmitter. More information on these functional units is provided in sections 54.3 - 54.6.

54.2.2 Protocol support

The GRETH_GBIT is implemented according to IEEE standard 802.3-2002. There is no support for the optional control sublayer. This means that packets with type 0x8808 (the only currently defined ctrl packets) are discarded.

54.2.3 Hardware requirements

The GRETH_GBIT is synthesisable with most Synthesis tools. There are three or four clock domains depending on if the gigabit mode is used. The three domains always present are the AHB clock, Ethernet Receiver clock (RX_CLK) and the 10/100 Ethernet transmitter clock (TX_CLK). If the gigabit mode is also used the fourth clock domain is the gigabit transmitter clock (GTX_CLK). Both full-duplex and half-duplex operating modes are supported and both can be run in either 10/100 or 1000 Mbit. The system frequency requirement (AHB clock) for 10 Mbit operation is 2.5 MHz, 18 MHz for 100 Mbit and 40 MHz for 1000 Mbit mode. The 18 MHz limit was tested on a Xilinx board with a DCM that did not support lower frequencies so it might be possible to run it on lower frequencies. It might also be possible to run the 10 Mbit mode on lower frequencies.

RX_CLK and TX_CLK are sourced by the PHY while GTX_CLK is sourced by the MAC according to the 802.3-2002 standard. The GRETH_GBIT does not contain an internal clock generator so GTX_CLK should either be generated in the FPGA (with a PLL/DLL) or with an external oscillator.

54.2.4 RAM debug support

Support for debug accesses the core's internal RAM blocks can be optionally enabled using the ramdebug VHDL generic. Setting it to 1 enables accesses to the transmitter and receiver RAM buffers and setting it to 2 enables accesses to the EDCL buffer in addition to the previous two buffers.

The transmitter RAM buffer is accessed starting from APB address offset 0x10000 which corresponds to location 0 in the RAM. There are 512 32-bit wide locations in the RAM which results in the last address being 0x107FC corresponding to RAM location 511 (byte addressing used on the APB bus).

Correspondingly the receiver RAM buffer is accessed starting from APB address offset 0x20000. The addresses, width and depth is the same.

The EDCL buffers are accessed starting from address 0x30000. The number of locations depend on the configuration and can be from 256 to 16384. Each location is 32-bits wide so the maximum address is 0x3FC and 0xFFFC correspondingly.

Before any debug accesses can be made the ramdebugen bit in the control register has to be set. During this time the debug interface controls the RAM blocks and normal operations is stopped. EDCL packets are not received. The MAC transmitter and receiver could still operate if enabled but the RAM buffers would be corrupt if debug accces are made simultaneously. Thus they MUST be disabled before the RAM debug mode is enabled.

54.2.5 Multibus version

There is a version of the core which has an additional master interface that can be used for the EDCL. Otherwise this version is identical to the basic version. The additional master interface is enabled with the edclsepahb VHDL generic. Then the ethi.edclsepahb signal control whether EDCL accesses are done on the standard master interface or the additional interface. Setting the signal to '0' makes the EDCL use the standard master interface while '1' selects the additional master. This signal is only sampled at reset and changes to this signal have no effect until the next reset.

54.2.6 Endianness

The core is designed for big-endian systems.

54.2.7 Timestamps

The GRETH_GBIT offers optional timestamping of packets. The timestamp of reception and transmission of packets are added to their respective descriptors. To enable the feature, the IP must be built with the timestamps configuration option set to 1 (section 54.12). To check during operation if the IP was built with timestamps enable, the corresponding capability bit can be read from the control register (section 54.9.1). If capable, the timestamping can be enabled by setting the timestamp enable bit in the control register. By setting the timestamp enable bit the IP will start writing timestamps to the descriptors. Note that number of descriptors supported are halved when timestamping is enabled, this is due to the increased size of the descriptor. Care must be taken by the user to handle the reduced number of descriptors. The user should therefore enable timestamping before setting up the descriptor buffers, to avoid any problems. For more information about the descriptors please refer to sections 54.3.2 and 54.4.2, for transmission and reception respectively.

54.3 Tx DMA interface

The transmitter DMA interface is used for transmitting data on an Ethernet network. The transmission is done using descriptors located in memory.

54.3.1 Setting up a descriptor.

A single descriptor is shown in table 812 and 813. The number of bytes to be sent should be set in the length field and the address field should point to the data. There are no alignment restrictions on the address field. If the interrupt enable (IE) bit is set, an interrupt will be generated when the packet has been sent (this requires that the transmitter interrupt bit in the control register is also set). The interrupt will be generated regardless of whether the packet was transmitted successfully or not.

*Table 812.*GRETH_GBIT transmit descriptor word 0 (address offset 0x0)

31 21	20	19	18	17	16	15	14	13	12	11	10 0
RESERVED	UC	TC	IC	MO	LC	AL	UE	IE	WR	EN	LENGTH

31: 21	RESERVED
20	UDP checksum (UC) - Calculate and insert the UDP checksum for this packet. The checksum is only inserted if an UDP packet is detected.
19	TCP checksum (TC) - Calculate and insert the TCP checksum for this packet. The checksum is only inserted if an TCP packet is detected.
18	IP checksum (IC) - Calculate and insert the IP header checksum for this packet. The checksum is only inserted if an IP packet is detected.
17	More (MO) - More descriptors should be fetched for this packet (Scatter Gather I/O).
16	Late collision (LC) - A late collision occurred during the transmission (1000 Mbit mode only).
15	Attempt limit error (AL) - The packet was not transmitted because the maximum number of attempts was reached.
14	Underrun error (UE) - The packet was incorrectly transmitted due to a FIFO underrun error.

*Table 812.*GRETH_GBIT transmit descriptor word 0 (address offset 0x0)

13	Interrupt enable (IE) - Enable Interrupts. An interrupt will be generated when the packet from this descriptor has been sent provided that the transmitter interrupt enable bit in the control register is set. The interrupt is generated regardless if the packet was transmitted successfully or if it terminated with an error.
12	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 8. The pointer automatically wraps to zero when the 1 kB boundary of the descriptor table is reached.
11	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
10: 0	LENGTH - The number of bytes to be transmitted.

*Table 813.*GRETH_GBIT transmit descriptor word 1 (address offset 0x4)

31	0
ADDRESS	

31: 0	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.

To enable a descriptor the enable (EN) bit should be set and after this is done, the descriptor should not be touched until the enable bit has been cleared by the GRETH_GBIT. The rest of the fields in the descriptor are explained later in this section.

54.3.2 Timestamping

If the core was built with timestamping support enabled and the feature is enabled in the control register, the descriptor is expanded by 2 words. The two new words contain the timestamps as described by table 814 and 815. The timestamps are latched by the GRETH_GBIT when the transmitter starts transmitting (at SFD), and written to the descriptor before the status is updated. In case of scatter gather operation, the timestamp is only written to the last descriptor.

*Table 814.*GRETH_GBIT transmit descriptor word 2 (address offset 0x8)

31	0
TIMESTAMP_MSB	

31: 0	Timestamp MSB(TIMESTAMP_MSB) - The most significant bits of the timestamp.

*Table 815.*GRETH_GBIT transmit descriptor word 3(address offset 0xC)

31	0
TIMESTAMP_LSB	

31: 0	Timestamp LSB(TIMESTAMP_LSB) - The least significant bits of the timestamp.

54.3.3 Starting transmissions

Enabling a descriptor is not enough to start a transmission. A pointer to the memory area holding the descriptors must first be set in the GRETH_GBIT. This is done in the transmitter descriptor pointer register. The address must be aligned to a 1 kB boundary. Bits 31 to 10 hold the base address of descriptor area while bits 9 to 3 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the GRETH_GBIT the pointer field is incremented by 8 to point at the next descriptor. The pointer will automatically wrap back to zero when the next 1 kB boundary has been reached (the descriptor at address offset 0x3F8 has been used). The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 1 kB boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when a transmission is active.

The final step to activate the transmission is to set the transmit enable bit in the control register. This tells the GRETH_GBIT that there are more active descriptors in the descriptor table. This bit should always be set when new descriptors are enabled, even if transmissions are already active. The descriptors must always be enabled before the transmit enable bit is set.

54.3.4 Descriptor handling after transmission

When a transmission of a packet has finished, status is written to the first word in the corresponding descriptor. The Underrun Error bit is set if the transmitter RAM was not able to provide data at a sufficient rate. This indicates a synchronization problem most probably caused by a low clock rate on the AHB clock. The whole packet is buffered in the transmitter RAM before transmission so underruns cannot be caused by bus congestion. The Attempt Limit Error bit is set if more collisions occurred than allowed. When running in 1000 Mbit mode the Late Collision bit indicates that a collision occurred after the slottime boundary was passed.

The packet was successfully transmitted only if these three bits are zero. The other bits in the first descriptor word are set to zero after transmission while the second word is left untouched.

The enable bit should be used as the indicator when a descriptor can be used again, which is when it has been cleared by the GRETH_GBIT. There are three bits in the GRETH_GBIT status register that hold transmission status. The Transmit Error (TE) bit is set each time an transmission ended with an error (when at least one of the three status bits in the transmit descriptor has been set). The Transmit Successful (TI) is set each time a transmission ended successfully.

The Transmit AHB Error (TA) bit is set when an AHB error was encountered either when reading a descriptor, reading packet data or writing status to the descriptor. Any active transmissions are aborted and the transmitter is disabled. The transmitter can be activated again by setting the transmit enable register.

54.3.5 Setting up the data for transmission

The data to be transmitted should be placed beginning at the address pointed by the descriptor address field. The GRETH_GBIT does not add the Ethernet address and type fields so they must also be stored in the data buffer. The 4 B Ethernet CRC is automatically appended at the end of each packet. Each descriptor will be sent as a single Ethernet packet. If the size field in a descriptor is greater than 1514 B, the packet will not be sent.

54.3.6 Scatter Gather I/O

A packet can be generated from data fetched from several descriptors. This is called Scatter Gather I/O. The More (MO) bit should be set to 1 to indicate that more descriptors should be used to generate the current packet. When data from the current descriptor has been read to the RAM the next descriptor is fetched and the new data is appended to the previous data. This continues until a descriptor with the MO bit set to 0 is encountered. The packet will then be transmitted.

Status is written immediately when data has been read to RAM for descriptors with MO set to 1. The status bits are always set to 0 since no transmission has occurred. The status bits will be written to the last descriptor for the packet (which had MO set to 0) when the transmission has finished.

No interrupts are generated for descriptors with MO set to 1 so the IE bit is don’t care in this case.

The checksum offload control bits (explained in section 54.3.7) must be set to the same values for all descriptors used for a single packet.

54.3.7 Checksum offloading

Support is provided for correct checksum calculations in hardware for TCP and UDP over IPv4 for non-fragmented packets with data length less or equal to maximum transmission unit. The checksum calculations are enabled in each descriptor and applies only to that packet (when the MO bit is set all descriptors used for a single packet must have the checksum control bits set in the same way).

The IP Checksum bit (IC) enables IP header checksum calculations. If an IPv4 packet is detected when transmitting the packet associated with the descriptor the header checksum is calculated and inserted. If TCP Checksum (TC) is set the TCP checksum is calculated and inserted if an TCP/IPv4 packet is detected. Finally, if the UDP Checksum bit is set the UDP checksum is calculated and inserted if a UDP/IPv4 packet is detected. In the case of fragmented IP packets, incorrect checksums for TCP and UDP are inserted for the first fragment (which contains the TCP or UDP header).

54.4 Rx DMA interface

The receiver DMA interface is used for receiving data from an Ethernet network. The reception is done using descriptors located in memory.

54.4.1 Setting up descriptors

A single descriptor is shown in table 816 and 817. The address field points at the location where the received data should be stored. There are no restrictions on alignment. The GRETH_GBIT will never store more than 1518 B to the buffer (the tagged maximum frame size excluding CRC). The CRC field (4 B) is never stored to memory so it is not included in this number. If the interrupt enable (IE) bit is set, an interrupt will be generated when a packet has been received to this buffer (this requires that the receiver interrupt bit in the control register is also set). The interrupt will be generated regardless of whether the packet was received successfully or not.

The enable bit is set to indicate that the descriptor is valid which means it can be used by the to store a packet. After it is set the descriptor should not be touched until the EN bit has been cleared by the GRETH_GBIT.

The rest of the fields in the descriptor are explained later in this section..

*Table 816.*GRETH_GBIT receive descriptor word 0 (address offset 0x0)

31 27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10 0
RESERVED	MC	IF	TR	TD	UR	UD	IR	ID	LE	OE	CE	FT	AE	IE	WR	EN	LENGTH

31: 27	RESERVED
26	Multicast address (MC) - The destination address of the packet was a multicast address (not broadcast).
25	IP fragment (IF) - Fragmented IP packet detected.
24	TCP error (TR) - TCP checksum error detected.
23	TCP detected (TD) - TCP packet detected.
22	UDP error (UR) - UDP checksum error detected.
21	UDP detected (UD) - UDP packet detected.
20	IP error (IR) - IP checksum error detected.
19	IP detected (ID) - IP packet detected.
18	Length error (LE) - The length/type field of the packet did not match the actual number of received bytes.
17	Overrun error (OE) - The frame was incorrectly received due to a FIFO overrun.

*Table 816.*GRETH_GBIT receive descriptor word 0 (address offset 0x0)

16	CRC error (CE) - A CRC error was detected in this frame.
15	Frame too long (FT) - A frame larger than the maximum size was received. The excessive part was truncated.
14	Alignment error (AE) - An odd number of nibbles were received.
13	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when a packet has been received to this descriptor provided that the receiver interrupt enable bit in the control register is set. The interrupt is generated regardless if the packet was received successfully or if it terminated with an error.
12	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 8. The pointer automatically wraps to zero when the 1 kB boundary of the descriptor table is reached.
11	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
10: 0	LENGTH - The number of bytes received to this descriptor.

*Table 817.*GRETH_GBIT receive descriptor word 1 (address offset 0x4)

31	0
ADDRESS	

31: 0	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.

54.4.2 Timestamping

If the core was built with timestamping support enabled and the feature is enabled in the control register, the descriptor is expanded by 2 words. The two new words contain the timestamps as described by table 818 and 819. The timestamps are latched by the GRETH_GBIT when the receiver starts receiving (at SFD), and written to the descriptor before the status is updated. In case of scatter gather operation, the timestamp is only written to the last descriptor.

*Table 818.*GRETH_GBIT receive descriptor word 2 (address offset 0x8)

31	0
TIMESTAMP_MSB	

31: 0	Timestamp MSB(TIMESTAMP_MSB) - The most significant bits of the timestamp.

*Table 819.*GRETH_GBIT receive descriptor word 3(address offset 0xC)

31	0
TIMESTAMP_LSB	

31: 0	Timestamp LSB(TIMESTAMP_LSB) - The least significant bits of the timestamp.

54.4.3 Starting reception

Enabling a descriptor is not enough to start reception. A pointer to the memory area holding the descriptors must first be set in the GRETH_GBIT. This is done in the receiver descriptor pointer register. The address must be aligned to a 1 kB boundary. Bits 31 to 10 hold the base address of descriptor area while bits 9 to 3 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the GRETH_GBIT the pointer field is incremented by 8 to point at the next descriptor. The pointer will automatically wrap back to zero when the next 1 kB boundary has been reached (the descriptor at address offset 0x3F8 has been used). The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 1 kB boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when reception is active.

The final step to activate reception is to set the receiver enable bit in the control register. This will make the GRETH_GBIT read the first descriptor and wait for an incoming packet.

54.4.4 Descriptor handling after reception

The GRETH indicates a completed reception by clearing the descriptor enable bit. The other control bits (WR, IE) are also cleared. The number of received bytes is shown in the length field. The parts of the Ethernet frame stored are the destination address, source address, type and data fields. Bits 24-14 in the first descriptor word are status bits indicating different receive errors. Bits 18 - 14 are zero after a reception without link layer errors. The status bits are described in table 816 (except the checksum offload bits which are also described in section 54.4.7).

Packets arriving that are smaller than the minimum Ethernet size of 64 B are not considered as a reception and are discarded. The current receive descriptor will be left untouched an used for the first packet arriving with an accepted size. The TS bit in the status register is set each time this event occurs.

If a packet is received with an address not accepted by the MAC, the IA status register bit will be set.

Packets larger than maximum size cause the FT bit in the receive descriptor to be set. The length field is not guaranteed to hold the correct value of received bytes. The counting stops after the word containing the last byte up to the maximum size limit has been written to memory.

The address word of the descriptor is never touched by the GRETH.

54.4.5 Reception with AHB errors

If an AHB error occurs during a descriptor read or data store, the Receiver AHB Error (RA) bit in the status register will be set and the receiver is disabled. The current reception is aborted. The receiver can be enabled again by setting the Receive Enable bit in the control register.

54.4.6 Accepted MAC addresses

In the default configuration the core receives packets with either the unicast address set in the MAC address register or the broadcast address. Multicast support can also be enabled and in that case a hash function is used to filter received multicast packets. A 64-bit register, which is accessible through the APB interface, determines which addresses should be received. Each address is mapped to one of the 64 bits using the hash function and if the bit is set to one the packet will be received. The address is mapped to the table by taking the 6 least significant bits of the 32-bit Ethernet crc calculated over the destination address of the MAC frame. A bit in the receive descriptor is set if a packet with a multicast address has been received to it.

54.4.7 Checksum offload

Support is provided for checksum calculations in hardware for TCP/UDP over IPv4. The checksum logic is always active and detects IPv4 packets with TCP or UDP payloads. If IPv4 is detected the ID bit is set, UD is set if an UDP payload is detected in the IP packet and TD is set if a TCP payload is detected in the IP packet (TD and UD are never set if an IPv4 packet is not detected). When one or more of these packet types is detected its corresponding checksum is calculated and if an error is detected the checksum error bit for that packet type is set. The error bits are never set if the corresponding packet type is not detected. The core does not support checksum calculations for TCP and UDP when the IP packet has been fragmented. This condition is indicated by the IF bit in the receiver descriptor and when set neither the TCP nor the UDP checksum error indications are valid.

54.5 MDIO Interface

The MDIO interface provides access to PHY configuration and status registers through a two-wire interface which is included in the MII interface. The GRETH_GBIT provides full support for the MDIO interface.

The MDIO interface can be used to access from 1 to 32 PHY containing 1 to 32 16-bit registers. A read transfer i set up by writing the PHY and register addresses to the MDIO Control register and setting the read bit. This caused the Busy bit to be set and the operation is finished when the Busy bit is cleared. If the operation was successful the Linkfail bit is zero and the data field contains the read data. An unsuccessful operation is indicated by the Linkfail bit being set. The data field is undefined in this case.

A write operation is started by writing the 16-bit data, PHY address and register address to the MDIO Control register and setting the write bit. The operation is finished when the busy bit is cleared and it was successful if the Linkfail bit is zero.

54.5.1 PHY interrupts

The core also supports status change interrupts from the PHY. A level sensitive interrupt signal can be connected on the mdint input. The mdint_pol vhdl generic can be used to select the polarity. The PHY status change bit in the status register is set each time an event is detected in this signal. If the PHY status interrupt enable bit is set at the time of the event the core will also generate an interrupt on the AHB bus.

54.6 Ethernet Debug Communication Link (EDCL)

The EDCL provides access to an on-chip AHB bus through Ethernet. It uses the UDP, IP and ARP protocols together with a custom application layer protocol. The application layer protocol uses an ARQ algorithm to provide reliable AHB instruction transfers. Through this link, a read or write transfer can be generated to any address on the AHB bus. The EDCL is optional and must be enabled with a generic.

54.6.1 Operation

The EDCL receives packets in parallel with the MAC receive DMA channel. It uses a separate MAC address which is used for distinguishing EDCL packets from packets destined to the MAC DMA channel. The EDCL also has an IP address which is set through generics. Since ARP packets use the Ethernet broadcast address, the IP-address must be used in this case to distinguish between EDCL ARP packets and those that should go to the DMA-channel. Packets that are determined to be EDCL packets are not processed by the receive DMA channel.

When the packets are checked to be correct, the AHB operation is performed. The operation is performed with the same AHB master interface that the DMA-engines use. The replies are automatically sent by the EDCL transmitter when the operation is finished. It shares the Ethernet transmitter with the transmitter DMA-engine but has higher priority.

54.6.2 EDCL protocols

The EDCL accepts Ethernet frames containing IP or ARP data. ARP is handled according to the protocol specification with no exceptions.

IP packets carry the actual AHB commands. The EDCL expects an Ethernet frame containing IP, UDP and the EDCL specific application layer parts. Table 820 shows the IP packet required by the EDCL. The contents of the different protocol headers can be found in TCP/IP literature.

Table 820. The IP packet expected by the EDCL.

Ethernet Header	IP Header	UDP Header	2 B Offset	4 B Control word	4 B Address	Data 0 - 242 4B Words	Ethernet CRC

The following is required for successful communication with the EDCL: A correct destination MAC address as set by the generics, an Ethernet type field containing 0x0806 (ARP) or 0x0800 (IP). The IP-address is then compared with the value determined by the generics for a match. The IP-header checksum and identification fields are not checked. There are a few restrictions on the IP-header fields. The version must be four and the header size must be 5 B (no options). The protocol field must always be 0x11 indicating a UDP packet. The length and checksum are the only IP fields changed for the reply.

The EDCL only provides one service at the moment and it is therefore not required to check the UDP port number. The reply will have the original source port number in both the source and destination fields. UDP checksum are not used and the checksum field is set to zero in the replies.

The UDP data field contains the EDCL application protocol fields. Table 821 shows the application protocol fields (data field excluded) in packets received by the EDCL. The 16-bit offset is used to align the rest of the application layer data to word boundaries in memory and can thus be set to any value. The R/W field determines whether a read (0) or a write(1) should be performed. The length

Table 821. The EDCL application layer fields in received frames.

16-bit Offset	14-bit Sequence number	1-bit R/W	10-bit Length	7-bit Unused

field contains the number of bytes to be read or written. If R/W is one the data field shown in Table 820 contains the data to be written. If R/W is zero the data field is empty in the received packets. Table 822 shows the application layer fields of the replies from the EDCL. The length field is always zero for replies to write requests. For read requests it contains the number of bytes of data contained in the data field.

Table 822. The EDCL application layer fields in transmitted frames.

16-bit Offset	14-bit sequence number	1-bit ACK/NAK	10-bit Length	7-bit Unused

The EDCL implements a Go-Back-N algorithm providing reliable transfers. The 14-bit sequence number in received packets are checked against an internal counter for a match. If they do not match, no operation is performed and the ACK/NAK field is set to 1 in the reply frame. The reply frame contains the internal counter value in the sequence number field. If the sequence number matches, the operation is performed, the internal counter is incremented, the internal counter value is stored in the sequence number field and the ACK/NAK field is set to 0 in the reply. The length field is always set to 0 for ACK/NAK=1 frames. The unused field is not checked and is copied to the reply. It can thus be set to hold for example some extra id bits if needed.

54.6.3 EDCL IP and Ethernet address settings

The default value of the EDCL IP and MAC addresses are set by `ipaddrh, ipaddrl, macaddrh and macaddrl` generics. The IP address and MAC address can later be changed by software. To allow several EDCL enabled GRETH controllers on the same sub-net, the 4 LSB bits of the IP and MAC address can optionally be set by an input signal. This is enabled by setting the `edcl` generic = 2, and driving the 4-bit LSB value on ethi.edcladdr.

54.6.4 EDCL buffer size

The EDCL has a dedicated internal buffer memory which stores the received packets during processing. The size of this buffer is configurable with a VHDL generic to be able to obtain a suitable compromise between throughput and resource utilization in the hardware. Table 823 lists the different buffer configurations. For each size the table shows how many concurrent packets the EDCL can handle, the maximum size of each packet including headers and the maximum size of the data payload. Sending more packets before receiving a reply than specified for the selected buffer size will lead to dropped packets. The behavior is unspecified if sending larger packets than the maximum allowed.

*Table 823.*EDCL buffer sizes

Total buffer size (kB)	Number of packet buffers	Packet buffer size (B)	Maximum data payload (B)
1	4	256	200
2	4	512	456
4	8	512	456
8	8	1024	968
16	16	1024	968
32	32	1024	968
64	64	1024	968

54.7 Media Independent Interfaces

There are several interfaces defined between the MAC sublayer and the Physical layer. The GRETH_GBIT supports the Media Independent Interface (MII) and the Gigabit Media Independent Interface (GMII).

The GMII is used in 1000 Mbit mode and the MII in 10 and 100 Mbit. These interfaces are defined separately in the 802.3-2002 standard but in practice they share most of the signals. The GMII has 9 additional signals compared to the MII. Four data signals are added to the receiver and transmitter data interfaces respectively and a new transmit clock for the gigabit mode is also introduced.

To support lower speed where the operation and clock frequency of the core and phy remains unchanged i.e. running at 100Mb/s when the IP is configured for 1000Mb/s speed enable signals should be created to mimic the desired bit rate.

When operating at 100Mb/s, every byte of the MAC frame is repeated 10 clock periods to achieve the correct bit rate. The GRETH_GBIT core does not take care of this operation and enable signals with toggling frequency of the correct bit rate needs to be created.

When operating at 10Mb/s, every byte of the MAC frame is repeated 100 clock periods to achieve the correct bit rate. The GRETH_GBIT core does not take care of this operation and enable signals with toggling frequency of the correct bit rate needs to be created.

*Table 824.*Signals in GMII and MII.

MII and GMII	GMII Only
txd[3:0]	txd[7:4]
tx_en	rxd[7:4]
tx_er	gtx_clk
rx_col	
rx_crs	
rxd[3:0]	
rx_clk	
rx_er	
rx_dv	
rx_en	
tx_dv	

54.8 AXI support

The core is designed for an AMBA system but can be adapted for AXI using the AHBM2AXI adapter.

54.9 Registers

The core is programmed through registers mapped into APB address space.

*Table 825.*GRETH_GBIT registers

APB address offset	Register
0x00	Control register
0x04	Status/Interrupt-source register
0x08	MAC Address MSB
0x0C	MAC Address LSB
0x10	MDIO Control/Status
0x14	Transmit descriptor pointer
0x18	Receiver descriptor pointer
0x1C	EDCL IP
0x20	Hash table msb
0x24	Hash table lsb
0x28	EDCL MAC address MSB
0x2C	EDCL MAC address LSB
0x10000 - 0x107FC	Transmit RAM buffer debug access
0x20000 - 0x207FC	Receiver RAM buffer debug access
0x30000 - 0x3FFFC	EDCL buffer debug access

54.9.1 Control register

*Table 826.*0x00 - CTRL - GRETH control register

31	30 28	27	26	25	24	23	22 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EA	BS	GA	MA	MC	R	TC	RESERVED	TS	ED	RD	DD	ME	PI	BM	GB	SP	RS	PM	FD	RI	TI	RE	TE
*	*	*	*	*	0	*	0	0	*	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	r	r	r	r	r	r		rw	rw	rw	rw	rw	rw	rw	rw	rw	wc	rw	rw	rw	rw	rw	rw

31 EDCL available (EA) - Set to one if the EDCL is available.

30: 28 EDCL buffer size (BS) - Shows the amount of memory used for EDCL buffers. 0 = 1 kB, 1 = 2 kB, ..., 6 = 64 kB.

27 Gigabit MAC available (GA) - This bit always reads as a 1 and indicates that the MAC has 1000 Mbit capability.

26 Mdio interrupts enabled (ME) - Set to one when the core supports mdio interrupts. Read only.

25 Multicast available (MC) - Set to one when the core supports multicast address reception. Read only.

24 RESERVED (R)

23 Timestamp capable (TC) - Set to one when the core supports timestamps. Read only.

22: 16 RESERVED

15 Timestamps enable (TS) - Set to one to enable timestamping. Is only effective if the timestamps configuration option was set to 1.

14 EDCL Disable(ED) - Set to one to disablethe EDCL and zero to enable it. Reset value taken from the ethi.edcldisable signal. Only available if the EDCL hardware is present in the core.

13 RAM debug enable (RD) - Set to one to enable the RAM debug mode. Reset value: ‘0’. Only available if the VHDL generic ramdebug is nonzero.

12 Disable duplex detection (DD) - Disable the EDCL speed/duplex detection FSM. If the FSM cannot complete the detection the MDIO interface will be locked in busy mode. If software needs to access the MDIO the FSM can be disabled here and as soon as the MDIO busy bit is 0 the interface is available. Note that the FSM cannot be reenabled again.

11 Multicast enable (ME) - Enable reception of multicast addresses. Reset value: ‘0’.

10 PHY status change interrupt enable (PI) - Enables interrupts for detected PHY status changes.

9 Burstmode (BM) - When set to 1, transmissions use burstmode in 1000 Mbit Half-duplex mode (GB=1, FD = 0). When 0 in this speed mode normal transmissions are always used with extension inserted. Operation is undefined when set to 1 in other speed modes. Reset value: ‘0’.

8 Gigabit (GB) - 1 sets the current speed mode to 1000 Mbit and when set to 0, the speed mode is selected with bit 7 (SP). Reset value: ‘0’.

7 Speed (SP) - Sets the current speed mode. 0 = 10 Mbit, 1 = 100 Mbit. Must not be set to 1 at the same time as bit 8 (GB). Reset valuie: ‘0’.

6 Reset (RS) - A one written to this bit resets the GRETH_GBIT core. Self clearing. No other accesses should be done .to the slave interface other than polling this bit until it is cleared.

5 Promiscuous mode (PM) - If set, the GRETH_GBIT operates in promiscuous mode which means it will receive all packets regardless of the destination address. Reset value: ‘0’.

Table 826. 0x00 - CTRL - GRETH control register

4	Full duplex (FD) - If set, the GRETH_GBIT operates in full-duplex mode otherwise it operates in half-duplex. Reset value: ‘0’.
3	Receiver interrupt (RI) - Enable Receiver Interrupts. An interrupt will be generated each time a packet is received when this bit is set. The interrupt is generated regardless if the packet was received successfully or if it terminated with an error. Reset value: ‘0’.
2	Transmitter interrupt (TI) - Enable Transmitter Interrupts. An interrupt will be generated each time a packet is transmitted when this bit is set. The interrupt is generated regardless if the packet was transmitted successfully or if it terminated with an error. Reset value: ‘0’.
1	Receive enable (RE) - Should be written with a one each time new descriptors are enabled. As long as this bit is one the GRETH_GBIT will read new descriptors and as soon as it encounters a disabled descriptor it will stop until RE is set again. This bit should be written with a one after the new descriptors have been enabled. Reset value: ‘0’.
0	Transmit enable (TE) - Should be written with a one each time new descriptors are enabled. As long as this bit is one the GRETH_GBIT will read new descriptors and as soon as it encounters a disabled descriptor it will stop until TE is set again. This bit should be written with a one after the new descriptors have been enabled. Reset value: ‘0’.

54.9.2 Status Register

*Table 827.*0x04 - STAT - GRETH_GBIT status register.

31 28	27 24	23 9	8	7	6	5	4	3	2	1	0
RESERVED	RESERVED	RESERVED	PS	IA	TS	TA	RA	TI	RI	TE	RE
0	0	0	0	0	0	NR	NR	NR	NR	NR	NR
r	r	r	wc	wc	wc	wc	wc	wc	wc	wc	wc

31: 9 RESERVED

8 PHY status changes (PS) - Set each time a PHY status change is detected.

7 Invalid address (IA) - A packet with an address not accepted by the MAC was received. Cleared when written with a one. Reset value: ‘0’.

6 Too small (TS) - A packet smaller than the minimum size was received. Cleared when written with a one. Reset value: ‘0’.

5 Transmitter AHB error (TA) - An AHB error was encountered in transmitter DMA engine. Cleared when written with a one. Not Reset.

4 Receiver AHB error (RA) - An AHB error was encountered in receiver DMA engine. Cleared when written with a one. Not Reset.

3 Transmit successful (TI) - A packet was transmitted without errors. Cleared when written with a one. Not Reset.

2 Receive successful (RI) - A packet was received without errors. Cleared when written with a one. Not Reset.

1 Transmitter error (TE) - A packet was transmitted which terminated with an error. Cleared when written with a one. Not Reset.

0 Receiver error (RE) - A packet has been received which terminated with an error. Cleared when written with a one. Not Reset.

54.9.3 Mac Address MSB

*Table 828.*0x08 - MACMSB - GRETH_GBIT MAC address MSB.

31 16	15 0
RESERVED	Bit 47 downto 32 of the MAC Address
0	DR
r	rw

31: 16 RESERVED

15: 0 The two most significant bytes of the MAC Address. Not Reset.

54.9.4 Mac Address LSB

*Table 829.*0x0C - MACLSB - GRETH_GBIT MAC address LSB.

31 0
Bit 31 downto 0 of the MAC Address
NR
rw

31: 0 The 4 least significant bytes of the MAC Address. Not Reset.

54.9.5 MDIO control/status Register

*Table 830.*0x10 - MDIO - GRETH_GBIT MDIO control/status register.

31 16	15 11	10 5	4	3	2	1	0
DATA	PHYADDR	REGADDR	RES	BU	LF	RD	WR
0	*	0		0	1	0	0
rw	rw	rw		r	r	rw	rw

31: 16 Data (DATA) - Contains data read during a read operation and data that is transmitted is taken from this field. Reset value: 0x0000.

15: 11 PHY address (PHYADDR) - This field contains the address of the PHY that should be accessed during a write or read operation. Reset value: “00000”.

10: 6 Register address (REGADDR) - This field contains the address of the register that should be accessed during a write or read operation. Reset value: ‘”00000”.

5: 4 RESERVED

3 Busy (BU) - When an operation is performed this bit is set to one. As soon as the operation is finished and the management link is idle this bit is cleared. Reset value: ‘0’.

2 Linkfail (LF) - When an operation completes (BUSY = 0) this bit is set if a functional management link was not detected. Reset value: ‘1’.

1 Read (RD) - Start a read operation on the management interface. Data is stored in the data field. Reset value: ‘0’.

0 Write (WR) - Start a write operation on the management interface. Data is taken from the Data field. Reset value: ‘0’.

54.9.6 Transmitter Descriptor Table Base Address Register

*Table 831.*0x14 - TXBASE - GRETH_GBIT transmitter descriptor table base address register.

31 10	9 3	2 0
BASEADDR	DESCPNT	RES
NR	0	0
rw	rw	r

31: 10 Transmitter descriptor table base address (BASEADDR) - Base address to the transmitter descriptor table.Not Reset.

9: 3 Descriptor pointer (DESCPNT) - Pointer to individual descriptors. Automatically incremented by the Ethernet MAC.

2: 0 RESERVED

54.9.7 Receiver Descriptor Table Base Address Register

*Table 832.*0x18 - RXBASE - GRETH_GBIT receiver descriptor table base address register.

31 10	9 3	2 0
BASEADDR	DESCPNT	RES
NR	0	0
rw	rw	r

31: 10 Receiver descriptor table base address (BASEADDR) - Base address to the receiver descriptor table.Not Reset.

9: 3 Descriptor pointer (DESCPNT) - Pointer to individual descriptors. Automatically incremented by the Ethernet MAC.

2: 0 RESERVED

54.9.8 IP Register

*Table 833.*0x1C - EDCLIP - GRETH_GBIT EDCL IP register

31	0
EDCL IP ADDRESS	
0	
rw	

31: 0 EDCL IP address. Reset value is set with the ipaddrh and ipaddrl generics.

54.9.9 Hash Table MSB Register

*Table 834.*0x20 - HMSB - GRETH Hash table MSB register

31	0
Hash table (64:32)	
NR	
rw	

31: 0 Hash table msb. Bits 64 downto 32 of the hash table.

54.9.10 Hash Table LSB Register

*Table 835.*0x24 - HLSB - GRETH Hash table LSB register

31	0
Hash table (64:32)	
NR	
rw	

31: 0 Hash table lsb. Bits 31downto 0 of the hash table.

54.9.11 MAC Address MSB

*Table 836.*0x28 - EMACMSB - GRETH_GBIT EDCL MAC address MSB.

31 16	15 0
RESERVED	Bit 47 downto 32 of the EDCL MAC Address
0	*
r	rw

31: 16 RESERVED

15: 0 The two most significant bytes of the EDCL MAC Address. Hardcoded reset value set with the VHDL generic macaddrh.

54.9.12 Mac Address LSB

*Table 837.*0x2C - EMACLSB - GRETH_GBIT EDCL MAC address LSB.

31	0
Bit 31 downto 0 of the EDCL MAC Address	
*	
rw	

31: 0 The 4 least significant bytes of the EDCL MAC Address. Hardcoded reset value set with the VHDL generics macaddrh and macaddrl. If the VHDL generic edcl is set to 2 bits 3 downto 0 are set with the edcladdr input signal.

54.10 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x01D. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

54.11 Implementation

54.11.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers if the GRLIB config package setting *grlib_async_reset_enable* is set.

54.11.2 Internal reset and reset via EDCL link

It is recommended to only use the internal reset bit in the control register for GRLIB releases 2018.2 and later. For earlier releases of GRLIB it is recommend to always use the input signal RST.

54.11.3 Endianness

The core changes endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). Changing *grlib_little_endian* controls TBD

54.12 Configuration options

Table 838 shows the configuration options of the core (VHDL generics).*) Not all addresses are allowed and most NICs and protocol implementations will discard frames with illegal addresses silently. Consult network literature if unsure about the addresses.

*Table 838.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRETH.	0 - NAHBIRQ-1	0
memtech	Memory technology used for the FIFOs.	0 - NTECH	inferred
ifg_gap	Number of ethernet clock cycles used for one interframe gap. Default value as required by the standard. Do not change unless you know what your doing.	1 - 255	24
attempt_limit	Maximum number of transmission attempts for one packet. Default value as required by the standard.	1 - 255	16
backoff_limit	Limit on the backoff size of the backoff time. Default value as required by the standard. Sets the number of bits used for the random value. Do not change unless you know what your doing.	1 - 10	10
slot_time	Number of ethernet clock cycles used for one slot- time. Default value as required by the ethernet standard. Do not change unless you know what you are doing.	1 - 255	128
mdcscaler	Sets the divisor value use to generate the mdio clock (mdc). The mdc frequency will be clk/(2*(mdcscaler+1)).	0 - 255	25

*Table 838.*Configuration options

Generic	Function	Allowed range	Default
nsync	Number of synchronization registers used.	1 - 2	2
edcl	Enable EDCL. 0 = disabled. 1 = enabled. 2 = enabled and 4-bit LSB of IP and ethernet MAC address programmed by ethi.edcladdr, 3=in addition to features for value 2 the reset value for the EDCL disable bit is taken from the ethi.edcldisable signal instead of being hardcoded to 0. 4=in addition to features for value 2 and 3 the an option is given to disable the EDCL via external input signal.	0 - 4	0
edclbufsz	Select the size of the EDCL buffer in kB.	1 - 64	1
burstlength	Sets the maximum burstlength used during DMA	4 - 128	32
macaddrh	Sets the upper 24 bits of the EDCL MAC address. Not all addresses are allowed and most NICs and protocol implementations will discard frames with illegal addresses silently. Consult network literature if unsure about the addresses.	0 - 16#FFFFFF#	16#00005E#
macaddrl	Sets the lower 24 bits of the EDCL MAC address. Not all addresses are allowed and most NICs and protocol implementations will discard frames with illegal addresses silently. Consult network literature if unsure about the addresses.	0 - 16#FFFFFF#	16#000000#
ipaddrh	Sets the upper 16 bits of the EDCL IP address reset value.	0 - 16#FFFF#	16#C0A8#
ipaddrl	Sets the lower 16 bits of the EDCL IP address reset value.	0 - 16#FFFF#	16#0035#
phyrstadr	Sets the reset value of the PHY address field in the MDIO register. When set to 32, the address is taken from the ethi.phyrstaddr signal.	0 - 32	0
sim	Set to 1 for simulations and 0 for synthesis. 1 selects a faster mdc clock to speed up simulations.	0 - 1	0
mdint_pol	Selects polarity for level sensitive PHY interrupt line. 0 = active low, 1 = active high	0 - 1	0
enable_mdint	Enables mdio interrupts.	0 - 1	0
multicast	Enables multicast support.	0 - 1	0
ramdebug	Enables debug access to the core's RAM blocks through the APB interface. 1=enables access to the receiver and transmitter RAM buffers, 2=enables access to the EDCL buffers in addition to the functionality of value 1. Setting this generic to 2 will have no effect if the edcl generic is 0.	0 - 2	0
ehindex	AHB master index for the separate EDCL master interface. Only used if edclsepahb is 1.	0 - NAHBMST-1	0
edclsepahb	Enables separate EDCL AHB master interface. A signal determines if the separate interface or the common interface is used. Only available in the GRETH_GBIT_MB version of the core.	0 - 1	0
mdiohold	Set output hold time for MDIO in number of AHB cycles. Should be 10 ns or more.	1 - 30	1
gmiimode	Enable the use of receive and transmit valid signals to enter data to/from the PHY at the correct rate.	0-1	0
ft	Enable fault tolerance for receive and transmit buffers. 0 - no fault tolerance, 1 - Parity DMR, 2 - TMR.	0-2	0
edclft	Enable fault tolerance for EDCL buffers. 0 - no fault tolerance, 1 - Parity DMR, 2 - TMR. This generic only has effect if generic edcl is non-zero.	0-2	0
timestamps	Enables timestamp support.	0-1	0

*Table 838.*Configuration options

Generic	Function	Allowed range	Default
external_mdio_ctrl	Locks the internal duplex detection mechanism to permanently disabled. ctrl.disable_duplex_detection will be stuck at 1. Also enables the phy_aneg_* interface for sourcing of speed and duplex configuration. The MDIO IOs are expected to be tied off/ left unconnected when this generic is set to 1.	0-1	0

54.13 Signal descriptions

Table 839 shows the interface signals of the core (VHDL ports).

*Table 839.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-

Table 839. Signal descriptions

Signal name	Field	Type	Function	Active
ETHI	gtx_clk	Input	Ethernet gigabit transmit clock.	-
	rmii_clk	Input	Ethernet RMII clock.	-
	tx_clk	Input	Ethernet transmit clock.	-
	tx_dv	Input	Ethernet transmitter enable	-
	rx_clk	Input	Ethernet receive clock.	-
	rxd	Input	Ethernet receive data.	-
	rx_dv	Input	Ethernet receive data valid.	High
	rx_er	Input	Ethernet receive error.	High
	rx_col	Input	Ethernet collision detected. (Asynchronous, sampled with tx_clk)	High
	rx_crs	Input	Ethernet carrier sense. (Asynchronous, sampled with tx_clk)	High
	rx_en	Input	Ethernet receiver enable.	-
	mdio_i	Input	Ethernet management data input	-
	mdint	Input	Ethernet management interrupt	-
	phyrstaddr	Input	Reset address for GRETH PHY address field.	-
	edcldis	Input	Disable EDCL functionality	-
	edcladdr	Input	Sets the four least significant bits of the EDCL MAC address and the EDCL IP address when the edcl generic is set to 2.	-
	edclsepahb	Input	Selects AHB master interface for the EDCL. '0' selects the common interface and '1' selects the separate interface. Only available in the GRETH_GBIT_MB version of the core when the VHDL generic edclsepahb is set to 1.	-
	edcldisable	Input	Reset value for edcl disable register bit. Setting the signal to 1 disables the EDCL at reset and 0 enables it.	-
ETHO	reset	Output	Ethernet reset (asserted when the MAC is reset).	Low
	txd	Output	Ethernet transmit data.	-
	tx_en	Output	Ethernet transmit enable.	High
	tx_er	Output	Ethernet transmit error.	High
	mdc	Output	Ethernet management data clock.	-
	mdio_o	Output	Ethernet management data output.	-
	mdio_oe	Output	Ethernet management data output enable.	Set by the oepol generic.
MTESTI**	BUF	Input	Memory BIST input signal to buffer RAM	-
	EDCL	Input	Memory BIST input signal to EDCL RAM	-
MTESTO**	BUF	Output	Memory BIST output signal from buffer RAM	-
	EDCL	Output	Memory BIST output signal from EDCL RAM	-
MTESTCLK**	N/A	Input	Memory BIST clock	-
TIMESTAMP	N/A	Input	64-bit timestamp input.	-
PHY_ANEG_VALID	N/A	Input	Data qualifier for the phy_aneg_result signal.	High

*Table 839.*Signal descriptions

Signal name	Field	Type	Function	Active
PHY_ANEG_RESULT	N/A	Input	Speed and duplex information from an external source. The generic external_mdio_ctrl must be set to 1 for this interface to be active. Bit 0: 0 - 10 Mbit/s, 1 - 100 Mbit/s Bit 1: 0 - Refer to bit 0, 1 - 1Gbit/s Bit 2: 0 - Half duplex, 1 - Full duplex	-

* see GRLIB IP Library User's Manual

** not available in FPGA releases

54.14 Library dependencies

Table 840 shows libraries used when instantiating the core (VHDL libraries).

*Table 840.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	ETHERNET_MAC	Signals, component	GRETH_GBIT component declarations, GRETH_GBIT signals.
GAISLER	NET	Signals	Ethernet signals

54.15 Instantiation

The first example shows how the non-mb version of the core can be instantiated and the second one show the mb version.

54.15.1 Non-MB version

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.ethernet_mac.all;

entity greth_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- ethernet signals
    ethi :  in  eth_in_type;
    etho :  in  eth_out_type
    );
end;

architecture rtl of greth_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

begin
```

```
  -- AMBA Components are instantiated here
  ...

  -- GRETH
  e1 : greth_gbit
    generic map(
     hindex       => 0,
     pindex       => 12,
     paddr        => 12,
     pirq         => 12,
     memtech      => inferred,
     mdcscaler    => 50,
     burstlength  => 32,
     nsync        => 1,
     edcl         => 1,
     edclbufsz    => 8,
     macaddrh     => 16#00005E#,
     macaddrl     => 16#00005D#,
     ipaddrh      => 16#c0a8#,
     ipaddrl      => 16#0035#)
 port map(
   rst          => rstn,
   clk          => clk,
   ahbmi        => ahbmi,
   ahbmo        => ahbmo(0),
   apbi         => apbi,
   apbo         => apbo(12),
   ethi         => ethi,
   etho         => etho
   );
end;
```

54.15.2 MB version

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.ethernet_mac.all;

entity greth_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- ethernet signals
    ethi :  in  eth_in_type;
    etho :  in  eth_out_type
    );
end;

architecture rtl of greth_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
begin

  -- AMBA Components are instantiated here
  ...

  -- GRETH
  e1 : greth_gbit_mb
```

```
      generic map(
       hindex       => 0,
       pindex       => 12,
       paddr        => 12,
       pirq         => 12,
       memtech      => inferred,
       mdcscaler    => 50,
       burstlength  => 32,
       nsync        => 1,
       edcl         => 1,
       edclbufsz    => 8,
       macaddrh     => 16#00005E#,
       macaddrl     => 16#00005D#,
       ipaddrh      => 16#c0a8#,
       ipaddrl      => 16#0035#,
       ehindex      => 1
       edclsepahb   => 1)
  port map(
    rst           => rstn,
    clk           => clk,
    ahbmi         => ahbmi,
    ahbmo         => ahbmo(0),
    ahbmi2        => ahbmi,
    ahbmo2        => ahbmo(1),
    apbi          => apbi,
    apbo          => apbo(12),
    ethi          => ethi,
    etho          => etho
    );
end;
```

55 GRFIFO - FIFO Interface

55.1 Overview

The FIFO interface is assumed to operate in an AMBA bus system where both the AMBA AHB bus and the APB bus are present. The AMBA APB bus is used for configuration, control and status handling. The AMBA AHB bus is used for retrieving and storing FIFO data in memory external to the FIFO interface. This memory can be located on-chip or external to the chip.

The FIFO interface supports transmission and reception of blocks of data by use of circular buffers located in memory external to the core. Separate transmit and receive buffers are assumed. Reception and transmission of data can be ongoing simultaneously.

After a data transfer has been set up via the AMBA APB interface, the DMA controller initiates a burst of read accesses on the AMBA AHB bus to fetch data from memory that are performed by the AHB master. The data are then written to the external FIFO. When a programmable amount of data has been transmitted, the DMA controller issues an interrupt.

After reception has been set up via the AMBA APB interface, data are read from the external FIFO. To store data to memory, the DMA controller initiates a burst of write accesses on the AMBA AHB bus that are performed by the AHB master. When a programmable amount of data has been received, the DMA controller issues an interrupt.

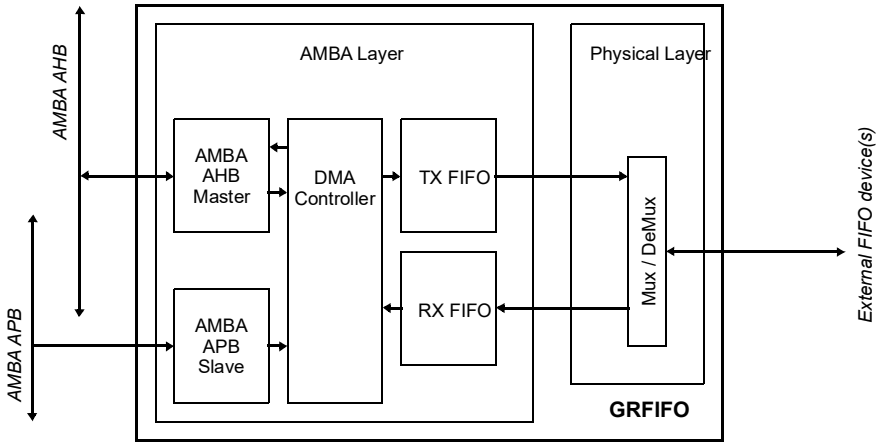


Figure 142. Block diagram

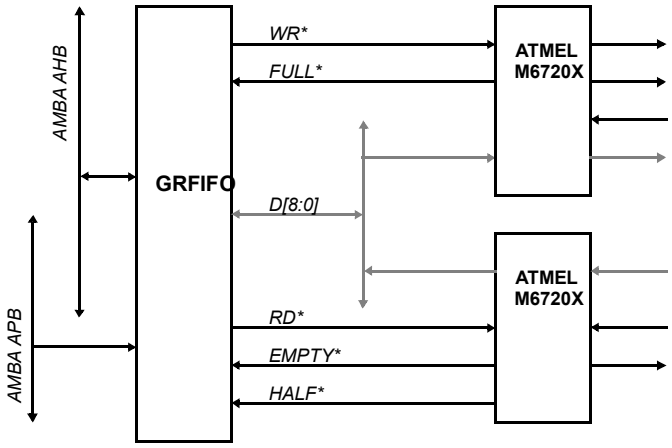


Figure 143. Example of usage, with shared external data bus

55.1.1 Function

The core implements the following functions:

- data transmission to external FIFO
- circular transmit buffer
- direct memory access for transmitter
- data reception from external FIFO
- circular receive buffer for receiver
- direct memory access
- automatic 8- and 16-bit data width conversion
- general purpose input output

55.1.2 Transmission

Data to be transferred via the FIFO interface are fetched via the AMBA AHB master interface from on-chip or off-chip memory. This is performed by means of direct memory access (DMA), implementing a circular transmit buffer in the memory. The transmit channel is programmable via the AMBA APB slave interface, which is also used for the monitoring of the FIFO and DMA status.

The transmit channel is programmed in terms of a base address and size of the circular transmit buffer. The outgoing data are stored in the circular transmit buffer by the system. A write address pointer register is then set by the system to indicate the last byte written to the circular transmit buffer. An interrupt address pointer register is used by the system to specify a location in the circular transmit buffer from which a data read should cause an interrupt to be generated.

The FIFO interface automatically indicates with a read address pointer register the location of the last fetched byte from the circular transmit buffer. Read accesses are performed as incremental bursts, except when close to the location specified by the interrupt pointer register at which point the last bytes might be fetched by means of single accesses.

Data transferred via the FIFO interface can be either 8- or 16-bit wide. The handling of the transmit channel is however the same. All transfers performed by the AMBA AHB master are 32-bit word based. No byte or half-word transfers are performed.

To handle the 8- and 16-bit FIFO data width, a 32-bit read access might carry less than four valid bytes. In such a case, the remaining bytes are ignored. When additional data are available in the circular transmit buffer, the previously fetched bytes will be re-read together with the newly written bytes to form the 32-bit data. Only the new bytes will be transmitted to the FIFO, not to transmit the same byte more than once. The aforementioned write address pointer indicates what bytes are valid.

An interrupt is generated when the circular transmit buffer is empty. The status of the external FIFO is observed via the AMBA APB slave interface, indicating Full Flag and Half-Full Flag.

55.1.3 Reception

Data received via the FIFO interface are stored via the AMBA AHB master interface to on-chip or off-chip memory. This is performed by means of direct memory access (DMA), implementing a circular receive buffer in the memory. The receive channel is programmable via the AMBA APB slave interface, which is also used for the monitoring of the FIFO and DMA status.

The receive channel is programmed in terms of a base address and size of the circular receive buffer. The incoming data are stored in the circular receive buffer. The interface automatically indicates with a write address pointer register the location of the last stored byte. A read address pointer register is used by the system to indicate the last byte read from the circular receive buffer. An interrupt address pointer register is used by the system to specify a location in the circular receive buffer to which a data write should cause an interrupt to be generated.

Write accesses are performed as incremental bursts, except when close to the location specified by the interrupt pointer register at which point the last bytes might be stored by means of single accesses.

Data transferred via the FIFO interface can be either 8- or 16-bit wide. The handling of the receive channel is however the same. All transfers performed by the AMBA AHB master are 32-bit word based. No byte or half-word transfers are performed.

To handle the 8- and 16-bit FIFO data width, a 32-bit write access might carry less than four valid bytes. In such a case, the remaining bytes will all be zero. When additional data are received from the FIFO interface, the previously stored bytes will be re-written together with the newly received bytes to form the 32-bit data. In this way, the previously written bytes are never overwritten. The aforementioned write address pointer indicates what bytes are valid.

An interrupt is generated when the circular receive buffer is full. If more FIFO data are available, they will not be moved to the circular receive buffer. The status of the external FIFO is observed via the AMBA APB slave interface, indicating Empty Flag and Half-Full Flag.

55.1.4 General purpose input output

Data input and output signals unused by the FIFO interface can be used as general purpose input output, providing 0, 8 or 16 individually programmable channels.

55.1.5 Interfaces

The core provides the following external and internal interfaces:

- FIFO interface
- AMBA AHB master interface, with sideband signals as per [GLRIB] including:
- cachability information
- interrupt bus
- configuration information
- diagnostic information
- AMBA APB slave interface, with sideband signals as per [GLRIB] including:
- interrupt bus
- configuration information
- diagnostic information

The interface is intended to be used with the following FIFO devices from ATMEL:

Name:Type:

M67204H4K x 9 FIFOESA/SCC 9301/049, SMD/5962-89568

M67206H16K x 9 FIFOESA/SCC 9301/048, SMD/5962-93177

M672061H16K x 9 FIFO ESA/SCC 9301/048, SMD/5962-93177

55.2 Interface

The external interface supports one or more FIFO devices for data output (transmission) and/or one or more FIFO devices for data input (reception). The external interface supports FIFO devices with 8- and 16-bit data width. Note that one device is used when 8-bit and two devices are used when 16-bit data width is needed. The data width is programmable. Note that this is performed commonly for both directions.

The external interface supports one parity bit over every 8 data bits. Note that there can be up to two parity bits in either direction. The parity is programmable in terms of odd or even parity. Note that odd parity is defined as an odd number of logical ones in the data bits and parity bit. Note that even parity is defined as an even number of logical ones in the data bits and parity bit. Parity is generated for write accesses to the external FIFO devices. Parity is checked for read accesses from the external FIFO devices and a parity failure results in an internal interrupt.

The external interface provides a Write Enable output signal. The external interface provides a Read Enable output signal. The timing of the access towards the FIFO devices is programmable in terms of wait states based on system clock periods.

The external interface provides an Empty Flag input signal, which is used for flow-control during the reading of data from the external FIFO, not reading any data while the external FIFO is empty. Note that the Empty Flag is sampled at the end of the read access to determine if the FIFO is empty. To determine when the FIFO is not empty, the Empty Flag is re-synchronized with Clk.

The external interface provides a Full Flag input signal, which is used for flow-control during the writing of data to the external FIFO, not writing any data while the external FIFO is full. Note that the Full Flag is sampled at the end of the write access to determine if the FIFO is full. To determine when the FIFO is not full, the Full Flag is re-synchronized with Clk.

The external interface provides a Half-Full Flag input signal, which is used as status information only.

The data input and output signals are possible to use as general purpose input output channels. This need is only realized when the data signals are not used by the FIFO interface. Each general purpose input output channel is individually programmed as input or output. The default reset configuration for each general purpose input output channel is as input. The default reset value each general purpose input output channel is logical zero. Note that protection toward spurious pulse commands during power up shall be mitigated as far as possible by means of I/O cell selection from the target technology.

55.3 Waveforms

The following figures show read and write accesses to the FIFO with 0 and 4 wait states, respectively.

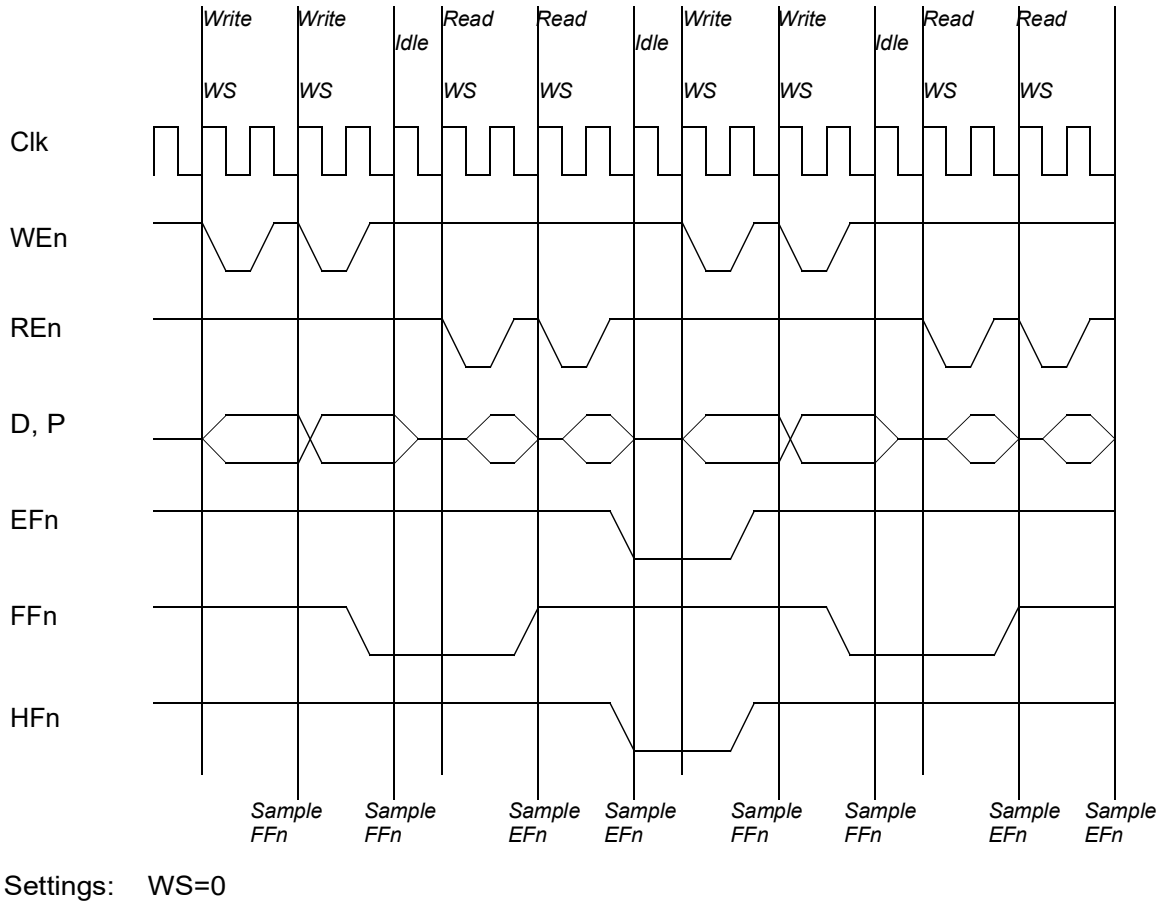


Figure 144. FIFO read and write access waveform, 0 wait states (WS)

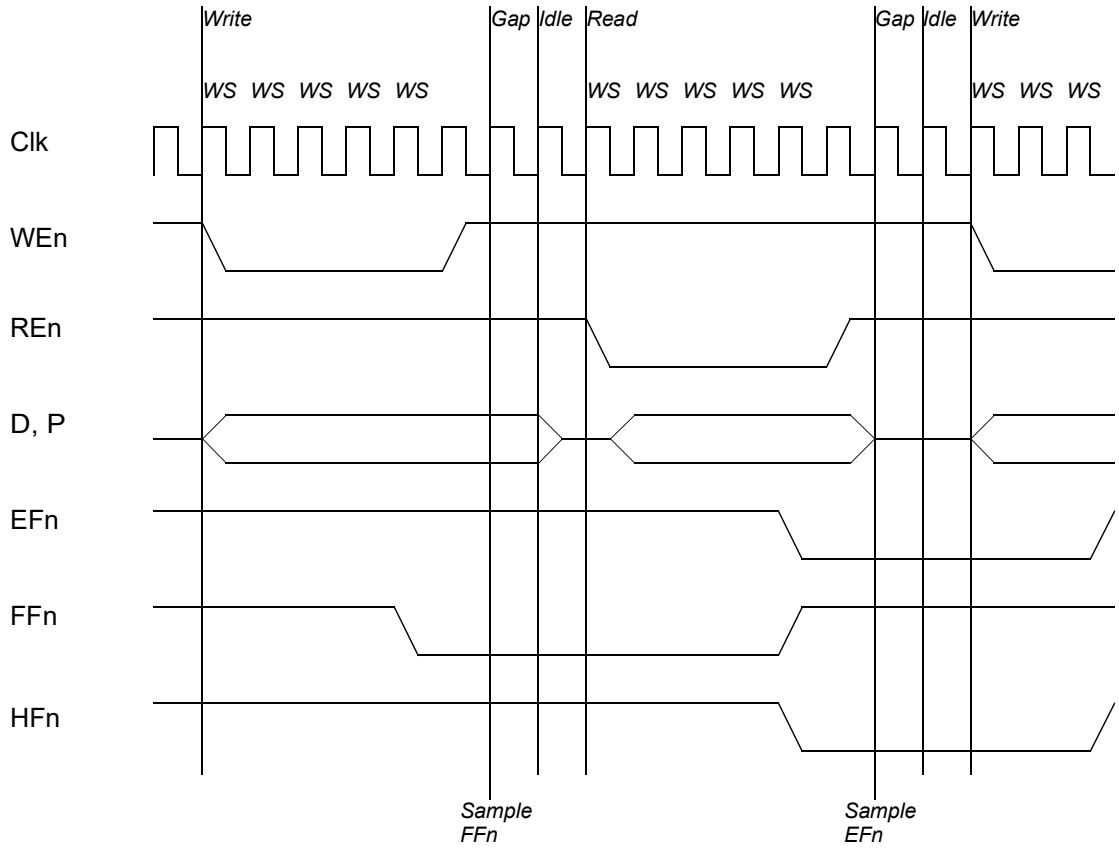


Figure 145. FIFO read and write access waveform, 4 wait states (WS)

55.4 Transmission

The transmit channel is defined by the following parameters:

- base address
- buffer size
- write pointer
- read pointer

The transmit channel can be enabled or disabled.

55.4.1 Circular buffer

The transmit channel operates on a circular buffer located in memory external to the FIFO controller. The circular buffer can also be used as a straight buffer. The buffer memory is accessed via the AMBA AHB master interface.

The size of the buffer is defined by the FifoTxSIZE.SIZE field, specifying the number of 64 byte blocks that fit in the buffer.

E.g. FifoTxSIZE.SIZE = 1 means 64 bytes fit in the buffer.

Note however that it is not possible to fill the buffer completely, leaving at least one word in the buffer empty. This is to simplify wrap-around condition checking.

E.g. FifoTxSIZE.SIZE = 1 means that 60 bytes fit in the buffer at any given time.

55.4.2 Write and read pointers

The write pointer (FifoTxWR.WRITE) indicates the position+1 of the last byte written to the buffer. The write pointer operates on number of bytes, not on absolute or relative addresses.

The read pointer (FifoTxRD.READ) indicates the position+1 of the last byte read from the buffer. The read pointer operates on number of bytes, not on absolute or relative addresses.

The difference between the write and the read pointers is the number of bytes available in the buffer for transmission. The difference is calculated using the buffer size, specified by the FifoTxSIZE.SIZE field, taking wrap around effects of the circular buffer into account.

Examples:

- There are 2 bytes available for transmit when FifoTxSIZE.SIZE=1, FifoTxWR.WRITE=2 and FifoTxRD.READ=0.
- There are 2 bytes available for transmit when FifoTxSIZE.SIZE=1, FifoTxWR.WRITE =0 and FifoTxRD.READ =62.
- There are 2 bytes available for transmit when FifoTxSIZE.SIZE=1, FifoTxWR.WRITE =1 and FifoTxRD.READ =63.
- There are 2 bytes available for transmit when FifoTxSIZE.SIZE=1, FifoTxWR.WRITE =5 and FifoTxRD.READ =3.

When a byte has been successfully written to the FIFO, the read pointer (FifoTxRD.READ) is automatically incremented, taking wrap around effects of the circular buffer into account. Whenever the write pointer FifoTxWR.WRITE and read pointer FifoTxRD.READ are equal, there are no bytes available for transmission.

55.4.3 Location

The location of the circular buffer is defined by a base address (FifoTxADDR.ADDR), which is an absolute address. The location of a circular buffer is aligned on a 1kbyte address boundary.

55.4.4 Transmission procedure

When the channel is enabled (FifoTxCTRL.ENABLE=1), as soon as there is a difference between the write and read pointer, a transmission will be started. Note that the channel should not be enabled if a potential difference between the write and read pointers could be created, to avoid the data transmission to start prematurely.

A data transmission will begin with a fetch of the data from the circular buffer to a local buffer in the FIFO controller. After a successful fetch, a write access will be performed to the FIFO.

The read pointer (FifoTxRD.READ) is automatically incremented after a successful transmission, taking wrap around effects of the circular buffer into account. If there is at least one byte available in the circular buffer, a new fetch will be performed.

If the write and read pointers are equal, no more prefetches and fetches will be performed, and transmission will stop.

Interrupts are provided to aid the user during transmission, as described in detail later in this section. The main interrupts are the TxError, TxEmpty and TxIrq which are issued on the unsuccessful transmission of a byte due to an error condition on the AMBA bus, when all bytes have been transmitted successfully and when a predefined number of bytes have been transmitted successfully.

Note that 32-bit wide read accesses past the address of the last byte or halfword available for transmission can be performed as part of a burst operation, although no read accesses are made beyond the circular buffer size.

All accesses to the AMBA AHB bus are performed as two consecutive 32-bit accesses in a burst, or as a single 32-bit access in case of an AMBA AHB bus error.

55.4.5 Straight buffer

It is possible to use the circular buffer as a straight buffer, with a higher granularity than the 1kbyte address boundary limited by the base address (FifoTxADDR.ADDR) field.

While the channel is disabled, the read pointer (FifoTxRD.READ) can be changed to an arbitrary value pointing to the first byte to be transmitted, and the write pointer (FifoTxWR.WRITE) can be changed to an arbitrary value.

When the channel is enabled, the transmission will start from the read pointer and continue to the write pointer.

55.4.6 AMBA AHB error

An AHB error response occurring on the AMBA AHB bus while data is being fetched will result in a TxError interrupt.

If the FifoCONF.ABORT bit is set to 0b, the channel causing the AHB error will re-try to read the data being fetched from memory till successful.

If the FifoCONF.ABORT bit is set to 1b, the channel causing the AHB error will be disabled (FifoTxCTRL.ENABLE is cleared automatically to 0 b). The read pointer can be used to determine which data caused the AHB error. The interface will not start any new write accesses to the FIFO. Any ongoing FIFO access will be completed and the FifoTxSTAT.TxOnGoing bit will be cleared. When the channel is re-enabled, the fetch and transmission of data will resume at the position where it was disabled, without losing any data.

55.4.7 Enable and disable

When an enabled transmit channel is disabled (FifoTxCTRL.ENABLE=0b), the interface will not start any new read accesses to the circular buffer by means of DMA over the AMBA AHB bus. No new write accesses to the FIFO will be started. Any ongoing FIFO access will be completed. If the

data is written successfully, the read pointer (FifoTxRD.READ) is automatically incremented and the FifoTxSTAT.TxOnGoing bit will be cleared. Any associated interrupts will be generated.

Any other fetched or pre-fetched data from the circular buffer which is temporarily stored in the local buffer will be discarded, and will be fetched again when the transmit channel is re-enabled.

The progress of the any ongoing access can be observed via the FifoTxSTAT.TxOnGoing bit. The FifoTxSTAT.TxOnGoing must be 0b before the channel can be re-configured safely (i.e. changing address, size or read/write pointers). It is also possible to wait for the TxEmpty interrupt described hereafter.

The channel can be re-enabled again without the need to re-configure the address, size and pointers. No data transmission is started while the channel is not enabled.

55.4.8 Interrupts

During transmission several interrupts can be generated:

- TxEmpty: Successful transmission of all data in buffer
- TxIrq: Successful transmission of a predefined number of data
- TxError: AHB access error during transmission

The TxEmpty and TxIrq interrupts are only generated as the result of a successful data transmission, after the FifoTxRD.READ pointer has been incremented.

55.5 Reception

The receive channel is defined by the following parameters:

- base address
- buffer size
- write pointer
- read pointer

The receive channel can be enabled or disabled.

55.5.1 Circular buffer

The receive channel operates on a circular buffer located in memory external to the FIFO controller. The circular buffer can also be used as a straight buffer. The buffer memory is accessed via the AMBA AHB master interface.

The size of the buffer is defined by the FifoRxSIZE.SIZE field, specifying the number 64 byte blocks that fit in the buffer.

E.g. FifoRxSIZE.SIZE=1 means 64 bytes fit in the buffer.

Note however that it is not possible for the hardware to fill the buffer completely, leaving at least two words in the buffer empty. This is to simplify wrap-around condition checking.

E.g. FifoRxSIZE.SIZE=1 means that 56 bytes fit in the buffer at any given time.

55.5.2 Write and read pointers

The write pointer (FifoRxWR.WRITE) indicates the position+1 of the last byte written to the buffer. The write pointer operates on number of bytes, not on absolute or relative addresses.

The read pointer (FifoRxRD.READ) indicates the position+1 of the last byte read from the buffer. The read pointer operates on number of bytes, not on absolute or relative addresses.

The difference between the write and the read pointers is the number of bytes available in the buffer for reception. The difference is calculated using the buffer size, specified by the FifoRxSIZE.SIZE field, taking wrap around effects of the circular buffer into account.

Examples:

- There are 2 bytes available for read-out when FifoRxSIZE.SIZE=1, FifoRxWR.WRITE =2 and FifoRxRD.READ=0.
- There are 2 bytes available for read-out when FifoRxSIZE.SIZE=1, FifoRxWR.WRITE =0 and FifoRxRD.READ=62.
- There are 2 bytes available for read-out when FifoRxSIZE.SIZE=1, FifoRxWR.WRITE =1 and FifoRxRD.READ=63.
- There are 2 bytes available for read-out when FifoRxSIZE.SIZE=1, FifoRxWR.WRITE =5 and FifoRxRD.READ=3.

When a byte has been successfully received and stored, the write pointer (FifoRxWR.WRITE) is automatically incremented, taking wrap around effects of the circular buffer into account.

55.5.3 Location

The location of the circular buffer is defined by a base address (FifoRxADDR.ADDR), which is an absolute address. The location of a circular buffer is aligned on a 1kbyte address boundary.

55.5.4 Reception procedure

When the channel is enabled (FifoRxCTRL.ENABLE=1), and there is space available for data in the circular buffer (as defined by the write and read pointer), a read access will be started towards the FIFO, and then an AMBA AHB store access will be started. The received data will be temporarily stored in a local store-buffer in the FIFO controller. Note that the channel should not be enabled until the write and read pointers are configured, to avoid the data reception to start prematurely

After a datum has been successfully stored the FIFO controller is ready to receive new data. The write pointer (FifoRxWR.WRITE) is automatically incremented, taking wrap around effects of the circular buffer into account.

Interrupts are provided to aid the user during reception, as described in detail later in this section. The main interrupts are the RxError, RxParity, RxFull and RxIrq which are issued on the unsuccessful reception of data due to an AMBA AHB error or parity error, when the buffer has been successfully filled and when a predefined number of data have been received successfully.

All accesses to the AMBA AHB bus are performed as two consecutive 32-bit accesses in a burst, or as a single 32-bit access in case of an AMBA AHB bus error.

55.5.5 Straight buffer

It is possible to use the circular buffer as a straight buffer, with a higher granularity than the 1kbyte address boundary limited by the base address (FifoRxADDR.ADDR) field.

While the channel is disabled, the write pointer (FifoRxWR.WRITE) can be changed to an arbitrary value pointing to the first data to be received, and the read pointer (FifoRxRD.READ) can be changed to an arbitrary value.

When the channel is enabled, the reception will start from the write pointer and continue to the read pointer.

55.5.6 AMBA AHB error

An AHB error response occurring on the AMBA AHB bus while data is being stored will result in an RxError interrupt.

If the FifoCONF.ABORT bit is set to 0b, the channel causing the AHB error will retry to store the received data till successful

If the FifoCONF.ABORT bit is set to 1b, the channel causing the AHB error will be disabled (FifoRxCTRL.ENABLE is cleared automatically to 0b). The write pointer can be used to determine which address caused the AHB error. The interface will not start any new read accesses to the FIFO. Any ongoing FIFO access will be completed and the data will be stored in the local receive buffer. The FifoRxSTAT.ONGOING bit will be cleared. When the receive channel is re-enabled, the reception and storage of data will resume at the position where it was disabled, without losing any data.

55.5.7 Enable and disable

When an enabled receive channel is disabled (FifoRxCTRL.ENABLE=0b), any ongoing data storage on the AHB bus will not be aborted, and no new storage will be started. If the data is stored successfully, the write pointer (FifoRxWR.WRITE) is automatically incremented. Any associated interrupts will be generated. The interface will not start any new read accesses to the FIFO. Any ongoing FIFO access will be completed.

The channel can be re-enabled again without the need to re-configure the address, size and pointers. No data reception is performed while the channel is not enabled.

The progress of the any ongoing access can be observed via the FifoRxSTAT.ONGOING bit. Note that the there might be data left in the local store-buffer in the FIFO controller. This can be observed via the FifoRxSTAT.RxByteCntr field. The data will not be lost if the channel is not reconfigured before re-enabled.

To empty this data from the local store-buffer to the external memory, the channel needs to be renabled. By setting the FifoRxIRQ.IRQ field to match the value of the FifoRxWR.WRITE field plus the value of the FifoRxSTAT.RxByteCntr field, an emptying to the external memory is forced of any data temporarily stored in the local store-buffer. Note however that additional data could be received in the local store-buffer when the channel is re-enabled.

The FifoRxSTAT.ONGOING must be 0b before the channel can be re-configured safely (i.e. changing address, size or read/write pointers).

55.5.8 Interrupts

During reception several interrupts can be generated:

- RxFull: Successful reception of all data possible to store in buffer
- RxIrq: Successful reception of a predefined number of data
- RxError: AHB access error during reception
- RxParity: Parity error during reception

The RxFull and RxIrq interrupts are only generated as the result of a successful data reception, after the FifoRxWR.WRITE pointer has been incremented.

55.6 Operation

55.6.1 Global reset and enable

When the FifoCTRL.RESET bit is set to 1b, a reset of the core is performed. The reset clears all the register fields to their default values. Any ongoing data transfers will be aborted.

55.6.2 Interrupt

Seven interrupts are implemented by the FIFO interface:

Index: Name:Description:

0 TxIrq Successful transmission of block of data

1 TxEmptyCircular transmission buffer empty

2 TxErrorAMBA AHB access error during transmission

3 RxIrq Successful reception of block of data

4 RxFullCircular reception buffer full

5 RxErrorAMBA AHB access error during reception

6 RxParityParity error during reception

The interrupts are configured by means of the *pirq* VHDL generic. The setting of the *singleirq* VHDL generic results in a single interrupt output, instead of multiple, configured by the means of the *pirq* VHDL generic, and enables the read and write of the interrupt registers. When multiple interrupts are implemented, each interrupt is generated as a one system clock period long active high output pulse. When a single interrupt is implemented, it is generated as an active high level output.

55.6.3 Reset

After a reset the values of the output signals are as follows:

Signal: Value after reset:

FIFOO.WEnde-asserted

FIFOO.REnde-asserted

55.6.4 Asynchronous interfaces

The following input signals are synchronized to Clk:

- FIFOI.EFn
- FIFOI.FFn
- FIFOI.HFn

55.7 Registers

The core is programmed through registers mapped into APB address space.

*Table 841.*GRFIFO registers

APB address offset	Register
0x000	Configuration Register
0x004	Status Register
0x008	Control Register
0x020	Transmit Channel Control Register
0x024	Transmit Channel Status Register
0x028	Transmit Channel Address Register
0x02C	Transmit Channel Size Register
0x030	Transmit Channel Write Register
0x034	Transmit Channel Read Register
0x038	Transmit Channel Interrupt Register
0x040	Receive Channel Control Register
0x044	Receive Channel Status Register
0x048	Receive Channel Address Register
0x04C	Receive Channel Size Register
0x050	Receive Channel Write Register
0x054	Receive Channel Read Register
0x058	Receive Channel Interrupt Register
0x060	Data Input Register
0x064	Data Output Register
0x068	Data Direction Register
0x100	Pending Interrupt Masked Status Register
0x104	Pending Interrupt Masked Register
0x108	Pending Interrupt Status Register
0x10C	Pending Interrupt Register
0x110	Interrupt Mask Register
0x114	Pending Interrupt Clear Register

55.7.1 Configuration Register [FifoCONF]

*Table 842.*Configuration Register

31 30 29 28 27 26 25 24 7	6	5 4	3	2 0
RESERVED	Abort	DW	Parity	WS
	0	0	0	0
	rw	rw	rw	rw

Field: Description:
6: ABORT Abort transfer on AHB ERROR
5-4: DW Data width:
00b = none
01b = 8 bitFIFOO.Dout[7:0],
FIFOI.Din[7:0]
10b = 16 bitFIFOO.Dout[15:0]
FIFOI.Din[15:0]

		11b = spare/none
3:	PARITY	Parity type: 0b = even 1b = odd
2-0:	WS	Number of wait states, 0 to 7

Note that the transmit or receive channel active during the AMBA AHB error is disabled if the ABORT bit is set to 1b. Note that all accesses on the affected channel will be disabled after an AMBA AHB error occurs while the ABORT bit is set to 1b. The accesses will be disabled until the affected channel is re-enabled setting the FifoTxCTRL.ENABLE or FifoRxCTRL.ENABLE bit, respectively.

Note that a wait states corresponds to an additional clock cycle added to the period when the read or write strobe is asserted. The default asserted width is one clock period for the read or write strobe when WS=0. Note that an idle gap of one clock cycle is always inserted between read and write accesses, with neither the read nor the write strobe being asserted.

Note that an additional gap of one clock cycle with the read or write strobe de-asserted is inserted between two accesses when WS is equal to or larger than 100b.

55.7.2 Status Register [FifoSTAT]

*Table 843.*Status register

31 … 6	5	4 … 0
-	SingleIrq	-
	*	
	r	

5: SingleIrq — Single interrupt output and interrupt registers when set to 1
*Reflect, VHDL generic /singleirq/

55.7.3 Control Register [FifoCTRL]

*Table 844.*Control Register

31 … 2	1	0
	Reset	
	0	
	w	

1: RESET — Reset complete FIFO interface, all registers

55.7.4 Transmit Channel Control Register [FifoTxCTRL]

*Table 845.*Transmit Channel Control Register

31 … 1	0
	Enable
	0
	rw

0: ENABLE — Enable channel

All bits are cleared to 0 at reset.

Note that in the case of an AHB bus error during an access while fetching transmit data, and the FifoConf.ABORT bit is 1b, then the ENABLE bit will be reset automatically.

At the time the ENABLE is cleared to 0b, any ongoing data writes to the FIFO are not aborted.

55.7.5 Transmit Channel Status Register [FifoTxSTAT]

Table 846. Transmit Channel Status Register

32 ... 7	6	5	4	3	2	1	0
	TxOnGoing	R	TxIrq	TxEmpty	TxError	FF	HF

6: TxOnGoing Access ongoing
4: TxIrq Successful transmission of block of data. cleared on read
3: TxEmpty Transmission buffer has been emptied. cleared on read
2: TxError AMB AHB access error during transmission. cleared on read
1: FF FIFO Full Flag
0: HF FIFO Half-Full Flag

55.7.6 Transmit Channel Address Register [FifoTxADDR]

Table 847. Transmit Channel Address Register

31 ... 10	9 ... 0
ADDR	
0	
rw	

31-10: ADDR Base address for circular buffer

55.7.7 Transmit Channel Size Register [FifoTxSIZE]

Table 848. Transmit Channel Size Register

31 ... 17	16 ... 6	5 ... 0
	SIZE	
	0	
	rw	

16-6: SIZE Size of circular buffer, in number of 64 bytes block

Valid SIZE values are 0, and between 1 and 1024. Note that the resulting behavior of invalid SIZE values is undefined.

Note that only SIZE*64-4 bytes can be stored simultaneously in the buffer. This is to simplify wrap-around condition checking.

The width of the SIZE field is configurable indirectly by means of the VHDL generic (ptrwidth) which sets the width of the read and write data pointers. In the above example VHDL generic ptrwidth=16, making the SIZE field 11 bits wide.

55.7.8 Transmit Channel Write Register [FifoTxWR]

Table 849. Transmit Channel Write Register

31 16	15 0
	WRITE
	0
	rw

15-0: WRITE Pointer to last written byte + 1

The WRITE field is written to in order to initiate a transfer, indicating the position +1 of the last byte to transmit.

Note that it is not possible to fill the buffer. There is always one word position in buffer unused. Software is responsible for not over-writing the buffer on wrap around (i.e. setting WRITE=READ).

Note that the LSB may be ignored for 16-bit wide FIFO devices.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

55.7.9 Transmit Channel Read Register [FifoTxRD]

Table 850. Transmit Channel Read Register

31 16	15 0
	READ
	0
	rw

15-0: READ Pointer to last read byte + 1

The READ field is written to automatically when a transfer has been completed successfully, indicating the position +1 of the last byte transmitted.

Note that the READ field can be used to read out the progress of a transfer.

Note that the READ field can be written to in order to set up the starting point of a transfer. This should only be done while the transmit channel is not enabled.

Note that the READ field can be automatically incremented even if the transmit channel has been disabled, since the last requested transfer is not aborted until completed.

Note that the LSB may be ignored for 16-bit wide FIFO devices.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

55.7.10 Transmit Channel Interrupt Register [FifoTxIRQ]

Table 851. Transmit Channel Interrupt Register

31 16	15 0
	IRQ
	0
	rw

15-0: IRQ Pointer+1 to a byte address from which the read of transmitted data shall result in an interrupt

Note that this indicates that a programmed amount of data has been sent. Note that the LSB may be ignored for 16-bit wide FIFO devices.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

55.7.11 Receive Channel Control Register [FifoRxCTRL]

*Table 852.*Receive Channel Control Register

31	2	1	0
			Enable
			0
			rw

0: ENABLE Enable channel

Note that in the case of an AHB bus error during an access while storing receive data, and the FifoConf.ABORT bit is 1b, then the ENABLE bit will be reset automatically.

At the time the ENABLE is cleared to 0b, any ongoing data reads from the FIFO are not aborted.

55.7.12 Receive Channel Status Register [FifoRxSTAT]

*Table 853.*Receive Channel Status Register

31 11	10 8	7	6	5	4	3	2	1	0
RESERVED	RxByteCntr	R	RxOnGoing	RxParity	RxIrq	RxFull	RxError	EF	HF
	0		0	0	0	0	0	1	1
	r		r	r*	r*	r*	r*	r	r

10-8: RxByteCntrNumber of bytes in local buffer
6: RxOnGoingAccess ongoing
5: RxParity Parity error during reception. Cleared on error.
4: RxIrq Successful reception of block of data. Cleared on error.
3: RxFull Reception buffer has been filled. Cleared on error.
2: RxError AMB AHB access error during reception. Cleared on error.
1: EF FIFO Empty Flag
0: HF FIFO Half-Full Flag

The circular buffer is considered as full when there are two words or less left in the buffer.

55.7.13 Receive Channel Address Register [FifoRxADDR]

*Table 854.*Receive Channel Address Register

31 10	9 0
ADDR	
0	
rw	

31-10: ADDR Base address for circular buffer

55.7.14 Receive Channel Size Register [FifoRxSIZE]

*Table 855.*Receive Channel Size Register

31 17	16 6	5 0
	SIZE	
	0	
	rw	

16-6: SIZE Size of circular buffer, in number of 64 byte blocks

Valid SIZE values are 0, and between 1 and 1024. Note that the resulting behavior of invalid SIZE values is undefined.

Note that only SIZE*64-8 bytes can be stored simultaneously in the buffer. This is to simplify wrap-around condition checking.

The width of the SIZE field is configurable indirectly by means of the VHDL generic (ptrwidth) which sets the width of the read and write data pointers. In the above example VHDL generic ptr-width=16, making the SIZE field 11 bits wide.

55.7.15 Receive Channel Write Register [FifoRxWR]

*Table 856.*Receive Channel Write Register

31 16	15 0
	WRITE
	0
	rw

15-0: WRITE Pointer to last written byte +1

The field is implemented as relative to the buffer base address (scaled with SIZE field).

The WRITE field is written to automatically when a transfer has been completed successfully, indicating the position +1 of the last byte received.

Note that the WRITE field can be used to read out the progress of a transfer.

Note that the WRITE field can be written to in order to set up the starting point of a transfer. This should only be done while the transmit channel is not enabled.

Note that the LSB may be ignored for 16-bit wide FIFO devices.

55.7.16 Receive Channel Read Register [FifoRxRD]

*Table 857.*Receive Channel Read Register

31 16	15 0
	READ
	0
	rw

15-0: READ Pointer to last read byte +1

The field is implemented as relative to the buffer base address (scaled with SIZE field).

The READ field is written to in order to release the receive buffer, indicating the position +1 of the last byte that has been read out.

Note that it is not possible to fill the buffer. There is always one word position unused in the buffer. Software is responsible for not over-reading the buffer on wrap around (i.e. setting WRITE=READ).

Note that the LSB may be ignored for 16-bit wide FIFO devices

55.7.17 Receive Channel Interrupt Register [FifoRxIRQ]

*Table 858.*Receive Channel Interrupt Register

31 16	15 0
	IRQ
	0
	rw

15-0: IRQ Pointer+1 to a byte address to which the write of received data shall result in an interrupt

Note that this indicates that a programmed amount of data has been received.

The field is implemented as relative to the buffer base address (scaled with SIZE field).

Note that the LSB may be ignored for 16-bit wide FIFO devices.

Note that by setting the IRQ field to match the value of the Receive Channel Write Register.WRITE field plus the value of the Receive Channel Status Register.RxByteCntr field, an emptying to the external memory is forced of any data temporarily stored in the local buffer.

55.7.18 Data Input Register [FifoDIN]

*Table 859.*Data Input Register

31 16	15 0
	DIN
	0
	r

15-0: DIN Input data *FIFOI.Din[15:0]*

Note that only the part of FIFOI.Din[15:0] not used by the FIFO can be used as general purpose input output, see FifoCONF.DW.

Note that only bits dwidth-1 to 0 are implemented.

55.7.19 Data Output Register [FifoDOUT]

*Table 860.*Data Output Register

31 16	15 0
	DOUT
	0
	rw

15-0: DOUT Output data *FIFOO.Dout[15:0]*

Note that only the part of FIFOO.Dout[15:0] not used by the FIFO can be used as general purpose input output, see FifoCONF.DW.

Note that only bits dwidth-1 to 0 are implemented.

55.7.20 Data Register [FifoDDIR]

*Table 861.*Data Direction Register

31 16	15 0
	DDIR
	0
	rw

15-0: DDIR Direction: *FIFOO.Dout[15:0]*
0b = input = high impedance,
1b = output = driven

Note that only the part of FIFOO.Dout[15:0] not used by the FIFO can be used as general purpose input output, see FifoCONF.DW.

Note that only bits dwidth-1 to 0 are implemented.

55.7.21 Interrupt registers

The interrupt registers give complete freedom to the software, by providing means to mask interrupts, clear interrupts, force interrupts and read interrupt status.

When an interrupt occurs the corresponding bit in the Pending Interrupt Register is set. The normal sequence to initialize and handle a module interrupt is:

- Set up the software interrupt-handler to accept an interrupt from the module.
- Read the Pending Interrupt Register to clear any spurious interrupts.
- Initialize the Interrupt Mask Register, unmasking each bit that should generate the module interrupt.
- When an interrupt occurs, read the Pending Interrupt Status Register in the software interrupt-handler to determine the causes of the interrupt.
- Handle the interrupt, taking into account all causes of the interrupt.
- Clear the handled interrupt using Pending Interrupt Clear Register.

Masking interrupts: After reset, all interrupt bits are masked, since the Interrupt Mask Register is zero. To enable generation of a module interrupt for an interrupt bit, set the corresponding bit in the Interrupt Mask Register.

Clearing interrupts: All bits of the Pending Interrupt Register are cleared when it is read or when the Pending Interrupt Masked Register is read. Reading the Pending Interrupt Masked Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register. Selected bits can be cleared by writing ones to the bits that shall be cleared to the Pending Interrupt Clear Register.

Forcing interrupts: When the Pending Interrupt Register is written, the resulting value is the original contents of the register logically OR-ed with the write data. This means that writing the register can force (set) an interrupt bit, but never clear it.

Reading interrupt status: Reading the Pending Interrupt Status Register yields the same data as a read of the Pending Interrupt Register, but without clearing the contents.

Reading interrupt status of unmasked bits: Reading the Pending Interrupt Masked Status Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register, but without clearing the contents.

The interrupt registers comprise the following:

*Table 862.*Interrupt registers

Description	Name	Mode
Pending Interrupt Masked Status Register	FifoPIMSR	r
Pending Interrupt Masked Register	FifoPIMR	r
Pending Interrupt Status Register	FifoPISR	r
Pending Interrupt Register	FifoPIR	rw
Interrupt Mask Register	FifoIMR	rw
Pending Interrupt Clear Register	FifoPICR	w

*Table 863.*Interrupt registers template

31 7	6	5	4	3	2	1	0
R	RxParity	RxError	RxFull	RxIrq	TxError	TxEmpty	TxIrq
	0	0	0	0	0	0	0
	*	*	*	*	*	*	*

6: RxParity Parity error during reception*
5: RxError AMBA AHB access error during reception*
4: RxFull Circular reception buffer full*
3: RxIrq Successful reception of block of data*
2: TxError AMBA AHB access error during transmission*
1: TxEmpty Circular transmission buffer empty*
0: TxIrq Successful transmission of block of data*

*See table 862.

55.8 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x035. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

55.9 Implementation

55.9.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

55.9.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core only performs 32-bit accesses and supports both big-endian and little-endian systems as long as the system can handle the 32-bit accesses.

55.10 Configuration options

Table 864 shows the configuration options of the core (VHDL generics).

*Table 864.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRFIFO.	0 - NAHBIRQ-1	0
dwidth	Data width	16	16
ptrwidth	Width of data pointers	16 - 16	16
singleirq	Single interrupt output. A single interrupt is assigned to the AMBA APB interrupt bus instead of multiple separate ones. The single interrupt output is controlled by the interrupt registers which are also enabled with this VHDL generic.	0, 1	0
oepol	Output enable polarity	0, 1	1

55.11 Signal descriptions

Table 865 shows the interface signals of the core (VHDL ports).

*Table 865.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-
FIFOI	DIN[31:0]	Input	Data input	-
	PIN[3:0]		Parity input	-
	EFN		Empty flag	Low
	FFN		Full flag	Low
	HFN		Half flag	Low
FIFOO	DOUT[31:0]	Output	Data output	-
	DEN[31:0]		Data output enable	-
	POUT[3:0]		Parity output	-
	PEN[3:0]		Parity output enable	-
	WEN		Write enable	Low
	REN		Read enable	Low

* see GRLIB IP Library User's Manual

55.12 Signal definitions and reset values

The signals and their reset values are described in table 866.

*Table 866.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
d[]	Input/Output	Data	High	Tri-state
p[]	Input/Output	Parity	High	Tri-state
wen	Output	Write Enable	-	Logical 1
ren	Output	Read Enable	-	Logical 1
efn	Input	Empty Flag	-	-
ffn	Input	Full Flag	-	-
hfn	Input	Half Flag	-	-

55.13 Timing

The timing waveforms and timing parameters are shown in figure 146 and are defined in table 867.

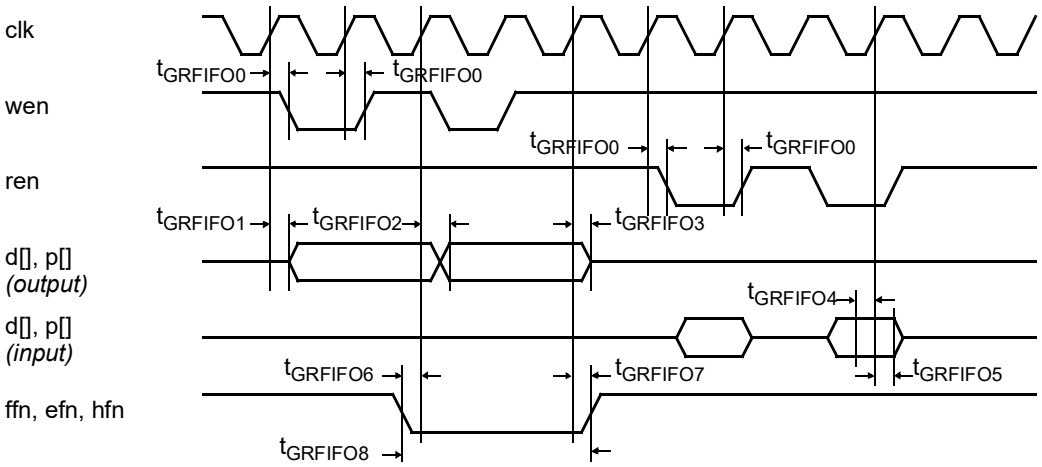


Figure 146. Timing waveforms

*Table 867.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{GRFIFO0}$	clock to output delay	rising *clk* edge	-	TBD	ns
$t_{GRFIFO1}$	clock to non-tri-state delay	rising *clk* edge	TBD	-	ns
$t_{GRFIFO2}$	clock to output delay	rising *clk* edge	-	TBD	ns
$t_{GRFIFO3}$	clock to tri-state delay	rising *clk* edge	-	TBD	ns
$t_{GRFIFO4}$	input to clock setup	rising *clk* edge	TBD	-	ns
t_{GRIFO5}	input from clock hold	rising *clk* edge	TBD	-	ns
t_{GRIFO6}	input to clock setup	rising *clk* edge	-	TBD	ns
t_{GRIFO7}	input from clock hold	rising *clk* edge	TBD	-	ns
t_{GRIFO8}	input assertion duration	-	TBD	-	*clk* periods

55.14 Library dependencies

Table 868 shows the libraries used when instantiating the core (VHDL libraries).

Table 868. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GRLIB	AMBA	Signals, component	DMA2AHB definitions
GAISLER	MISC	Signals, component	Component declarations, signals.

55.15 Instantiation

This example shows how the core can be instantiated.

```
grfifo0: grfifo
  generic map (
    hindex    => GRFIFO_HINDEX,
    pindex    => GRFIFO_PINDEX,
    paddr     => GRFIFO_PADDR,
    pmask     => 16#fff#,
    pirq      => CFG_GRFIFO_IRQ,
    dwidth    => CFG_GRFIFO_WIDTH,
    ptrwidth  => CFG_GRFIFO_ABITS,
    singleirq => CFG_GRFIFO_SINGLE,
    oepol     => 1)
  port map (
    rstn  => rstn,
    clk   => clk,
    apbi  => apbi,
    apbo  => apbo(GRFIFO_PINDEX),
    ahbi  => ahbi,
    ahbo  => ahbo(GRFIFO_HINDEX),
    fifoi => fifoi,
    fifoo => fifoo);
```

56 GRFPU - High-performance IEEE-754 Floating-point unit

56.1 Overview

GRFPU is a high-performance FPU implementing floating-point operations as defined in the IEEE Standard for Binary Floating-Point Arithmetic (IEEE-754) and the SPARC V8 standard (IEEE-1754). Supported formats are single and double precision floating-point numbers. The advanced design combines two execution units, a fully pipelined unit for execution of the most common FP operations and a non-blocking unit for execution of divide and square-root operations.

The logical view of the GRFPU is shown in figure 147.

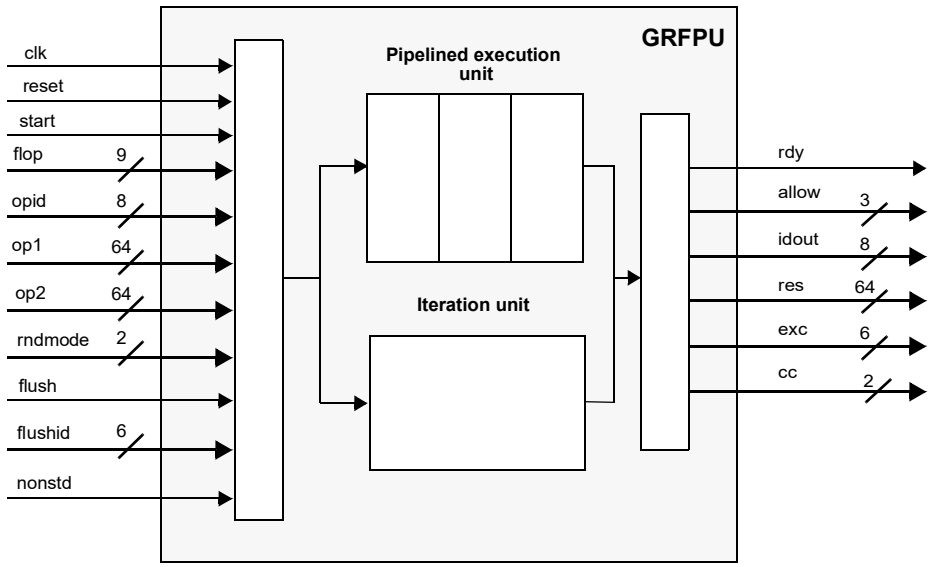


Figure 147. GRFPU Logical View

This document describes GRFPU from functional point of view. Chapter “Functional description” gives details about GRFPU’s implementation of the IEEE-754 standard including FP formats, operations, opcodes, operation timing, rounding and exceptions. “Signals and timing” describes the GRFPU interface and its signals. “GRFPU Control Unit” describes the software aspects of the GRFPU integration into a LEON processor through the GRFPU Control Unit - GRFPC. For implementation details refer to the white paper, “GRFPU - High Performance IEEE-754 Floating-Point Unit” (available at *www.gaisler.com*).

56.2 Functional description

56.2.1 Floating-point number formats

GRFPU handles floating-point numbers in single or double precision format as defined in the IEEE-754 standard with exception for denormalized numbers. See section 56.2.5 for more information on denormalized numbers.

56.2.2 FP operations

GRFPU supports four types of floating-point operations: arithmetic, compare, convert and move. The operations implement all FP instructions specified by SPARC V8 instruction set, and most of the operations defined in IEEE-754. All operations are summarized in table 869, with their opcodes, operands, results and exception codes. Throughputs and latencies and are shown in table 869.

Table 869.: GRFPU operations

Operation	OpCode[8:0]	Op1	Op2	Result	Exceptions	Description
Arithmetic operations						
FADDS FADDD	001000001 001000010	SP DP	SP DP	SP DP	UNF, NV, OF, UF, NX	Addition
FSUBS FSUBD	001000101 001000110	SP DP	SP DP	SP DP	UNF, NV, OF, UF, NX	Subtraction
FMULS FMULD FSMULD	001001001 001001010 001101001	SP DP SP	SP DP SP	SP DP DP	UNF, NV, OF, UF, NX UNF, NV, OF, UF, NX UNF, NV, OF, UF	Multiplication, FSMULD gives exact double-precision product of two single-precision operands.
FDIVS FDIVD	001001101 001001110	SP DP	SP DP	SP DP	UNF, NV, OF, UF, NX, DZ	Division
FSQRTS FSQRTD	000101001 000101010	- -	SP DP	SP DP	UNF, NV, NX	Square-root
Conversion operations						
FITOS FITOD	011000100 011001000	-	INT	SP DP	NX -	Integer to floating-point conversion
FSTOI FDTOI	011010001 011010010	-	SP DP	INT	UNF, NV, NX	Floating-point to integer conversion. The result is rounded in round-to-zero mode.
FSTOI_RND FDTOI_RND	111010001 111010010	-	SP DP	INT	UNF, NV, NX	Floating-point to integer conversion. Rounding according to RND input.
FSTOD FDTOS	011001001 011000110	-	SP DP	DP SP	UNF, NV UNF, NV, OF, UF, NX	Conversion between floating-point formats
Comparison operations						
FCMPS FCMPD	001010001 001010010	SP DP	SP DP	CC	NV	Floating-point compare. Invalid exception is generated if either operand is a signaling NaN.
FCMPES FCMPED	001010101 001010110	SP DP	SP DP	CC	NV	Floating point compare. Invalid exception is generated if either operand is a NaN (quiet or signaling).
Negate, Absolute value and Move						
FABSS	000001001	-	SP	SP	-	Absolute value.
FNEGS	000000101	-	SP	SP	-	Negate.
FMOVS	000000001		SP	SP	-	Move. Copies operand to result output.

SP - single precision floating-point number
DP - double precision floating-point number
INT - 32 bit integer
CC - condition codes, see table 872
UNF, NV, OF, UF, NX - floating-point exceptions, see section 56.2.3

Arithmetic operations include addition, subtraction, multiplication, division and square-root. Each arithmetic operation can be performed in single or double precision formats. Arithmetic operations have one clock cycle throughput and a latency of four clock cycles, except for divide and square-root operations, which have a throughput of 16 - 25 clock cycles and latency of 16 - 25 clock cycles (see

table 870). Add, sub and multiply can be started on every clock cycle, providing high throughput for these common operations. Divide and square-root operations have lower throughput and higher latency due to complexity of the algorithms, but are executed in parallel with all other FP operations in a non-blocking iteration unit. Out-of-order execution of operations with different latencies is easily handled through the GRFPU interface by assigning an id to every operation which appears with the result on the output once the operation is completed (see section 56.4).

Table 870.: Throughput and latency

Operation	Throughput	Latency
FADDS, FADDD, FSUBS, FSUBD, FMULS, FMULD, FSMULD	1	4
FITOS, FITOD, FSTOI, FSTOI_RND, FDTOI, FDTOI_RND, FSTOD, FDTOS	1	4
FCMPS, FCMPD, FCMPES, FCMPED	1	4
FDIVS	16	16
FDIVD	16.5 (15/18)*	16.5 (15/18)*
FSQRTS	24	24
FSQRTD	24.5 (23/26)*	24.5 (23/26)*

* Throughput and latency are data dependent with two possible cases with equal statistical possibility.

Conversion operations execute in a pipelined execution unit and have throughput of one clock cycle and latency of four clock cycles. Conversion operations provide conversion between different floating-point numbers and between floating-point numbers and integers.

Comparison functions offering two different types of quiet Not-a-Numbers (QNaNs) handling are provided. Move, negate and absolute value are also provided. These operations do not ever generate unfinished exception (unfinished exception is never signaled since compare, negate, absolute value and move handle denormalized numbers).

56.2.3 Exceptions

GRFPU detects all exceptions defined by the IEEE-754 standard. This includes detection of Invalid Operation (NV), Overflow (OF), Underflow (UF), Division-by-Zero (DZ) and Inexact (NX) exception conditions. Generation of special results such as NaNs and infinity is also implemented. Overflow (OF) and underflow (UF) are detected before rounding. If an operation underflows the result is flushed to zero (GRFPU does not support denormalized numbers or gradual underflow). A special Unfinished exception (UNF) is signaled when one of the operands is a denormalized number which is not handled by the arithmetic and conversion operations.

56.2.4 Rounding

All four rounding modes defined in the IEEE-754 standard are supported: round-to-nearest, round-to-+inf, round-to--inf and round-to-zero.

56.2.5 Denormalized numbers

Denormalized numbers are not handled by the GRFPU arithmetic and conversion operations. A system (microprocessor) with the GRFPU could emulate rare cases of operations on denormals in software using non-FPU operations. A special Unfinished exception (UNF) is used to signal an arithmetic or conversion operation on the denormalized numbers. Compare, move, negate and absolute value operations can handle denormalized numbers and do not raise the unfinished exception. GRFPU does not generate any denormalized numbers during arithmetic and conversion operations on normalized numbers. If the infinitely precise result of an operation is a tiny number (smaller than minimum value representable in normal format) the result is flushed to zero (with underflow and inexact flags set).

56.2.6 Non-standard Mode

GRFPU can operate in a non-standard mode where all denormalized operands to arithmetic and conversion operations are treated as (correctly signed) zeroes. Calculations are performed on zero operands instead of the denormalized numbers obeying all rules of the floating-point arithmetics including rounding of the results and detecting exceptions.

56.2.7 NaNs

GRFPU supports handling of Not-a-Numbers (NaNs) as defined in the IEEE-754 standard. Operations on signaling NaNs (SNaNs) and invalid operations (e.g. inf/inf) generate the Invalid exception and deliver QNaN_GEN as result. Operations on Quiet NaNs (QNaNs), except for FCMPES and FCMPED, do not raise any exceptions and propagate QNaNs through the FP operations by delivering NaN-results according to table 871. QNaN_GEN is 0x7fffe00000000000 for double precision results and 0x7fff0000 for single precision results.

Table 871.: Operations on NaNs

	Operand 2			
Operand 1		FP	QNaN2	SNaN2
	none	FP	QNaN2	QNaN_GEN
	FP	FP	QNaN2	QNaN_GEN
	QNaN1	QNaN1	QNaN2	QNaN_GEN
	SNaN1	QNaN_GEN	QNaN_GEN	QNaN_GEN

56.3 Signal descriptions

Table 872 shows the interface signals of the core (VHDL ports). All signals are active high except for RST which is active low.

Table 872.: Signal descriptions

Signal	I/O	Description
CLK	I	Clock
RESET	I	Reset
START	I	Start an FP operation on the next rising clock edge
NONSTD	I	Nonstandard mode. Denormalized operands are converted to zero.
FLOP[8:0]	I	FP operation. For codes see table 869.
OP1[63:0] OP2[63:0]	I	FP operation operands are provided on these one or both of these inputs. All 64 bits are used for IEEE-754 double precision floating-point numbers, bits [63:32] are used for IEEE-754 single precision floating-point numbers and 32-bit integers.
OPID[7:0]	I	FP operation id. Every operation is associated with an id which will appear on the RESID output when the FP operation is completed. This value shall be incremented by 1 (with wrap-around) for every started FP operation. If flushing is used, FP operation id is 6 -bits wide (OPID[5:0] are used for id, OPID[7:6] are tied to “00”). If flushing is not used (input signal FLUSH is tied to ‘0’), all 8-bits (OPID[7:0]) are used.
FLUSH	I	Flush FP operation with FLUSHID.
FLUSHID[5:0]	I	Id of the FP operation to be flushed.
RNDMODE[1:0]	I	Rounding mode. 00 - rounding-to-nearest, 01 - round-to-zero, 10 - round-to-+inf, 11 - round-to--inf.
RES[63:0]	O	Result of an FP operation. If the result is double precision floating-point number all 64 bits are used, otherwise single precision or integer result appears on RESULT[63:32].
EXC[5:0]	O	Floating-point exceptions generated by an FP operation. EXC[5] - Unfinished FP operation. Generated by an arithmetic or conversion operation with denormalized input(s). EXC[4] - Invalid exception. EXC[3] - Overflow. EXC[2] - Underflow. EXC[1] - Division by zero. EXC[0] - Inexact.
ALLOW[2:0]	O	Indicates allowed FP operations during the next clock cycle. ALLOW[0] - FDIVS, FDIVD, FSQRTS and FSQRTD allowed ALLOW[1] - FMULS, FMULD, FSMULD allowed ALLOW[2] - all other FP operations allowed
RDY	O	The result of a FP operation will be available at the end of the next clock cycle.
CC[1:0]	O	Result (condition code) of an FP compare operation. 00 - equal 01 - operand1 < operand2 10 - operand1 > operand2 11 - unordered
IDOUT[7:0]	O	Id of the FP operation whose result appears at the end of the next clock cycle.
TESTEN	I	Test enable. When GRLIB has been configured to use asynchronous reset then TESTEN selects if the internal reset signal should come from RESET or TESTRST.
TESTRST	I	Reset signal in test mode (TESTEN = High). Only used when GRLIB has been configured to use asynchronous reset,

56.4 Timing

An FP operation is started by providing the operands, opcode, rounding mode and id before rising edge. The operands need to be provided a small set-up time before a rising edge while all other signals are latched on rising edge.

The FPU is fully pipelined and a new operation can be started every clock cycle. The only exceptions are divide and square-root operations which require 16 to 26 clock cycles to complete, and which are not pipelined. Division and square-root are implemented through iterative series expansion algorithm. Since the algorithms basic step is multiplication the floating-point multiplier is shared between multiplication, division and square-root. Division and square-root do not occupy the multiplier during the whole operation and allow multiplication to be interleaved and executed parallelly with division or square-root.

One clock cycle before an operation is completed, the output signal RDY is asserted to indicate that the result of an FPU operation will appear on the output signals at the end of the next cycle. The id of the operation to be completed and allowed operations are reported on signals RESID and ALLOW. During the next clock cycle the result appears on RES, EXCEPT and CC outputs.

Figure 148 shows signal timing during four arithmetic operations on GRFPU.

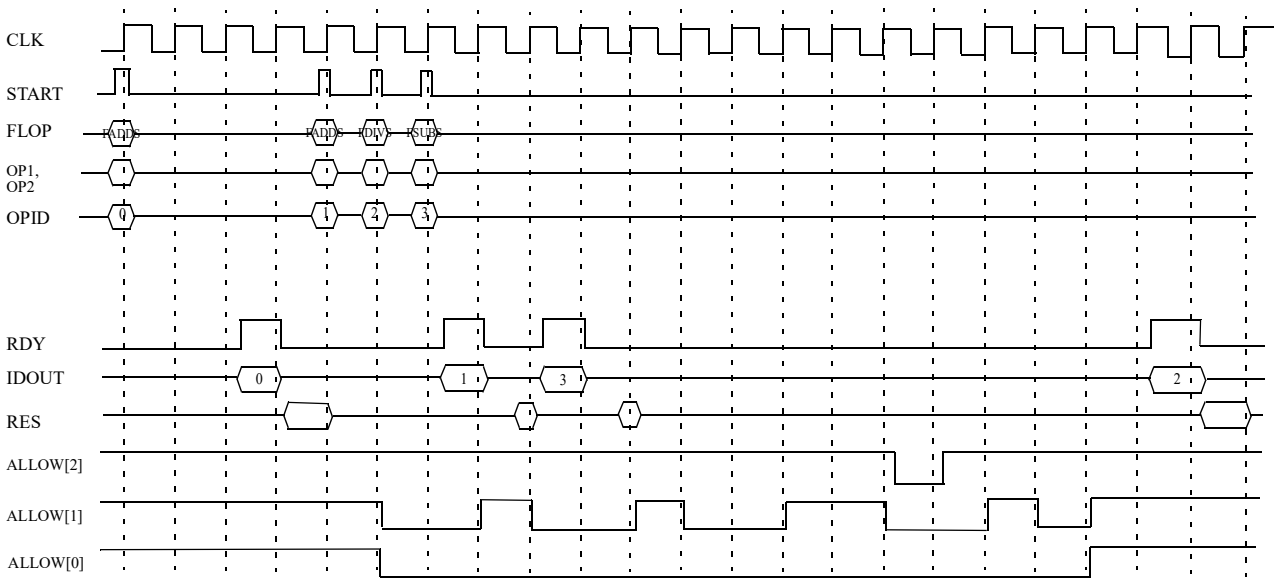


Figure 148. Signal timing

56.5 Shared FPU

56.5.1 Overview

In multi-processor systems, a single GRFPU can be shared between multiple CPU cores providing an area efficient solution. In this configuration, the GRFPU is extended with a wrapper. Each CPU core issues a request to execute a specific FP operation to the wrapper, which performs fair arbitration using the round-robin algorithm. When a CPU core has started a divide or square-root operation, the FPU is not able to accept a new division or square-root until the current operation has finished. Also, during the execution of a division or square-root, other operations cannot be accepted during certain cycles. This can lead to the, currently, highest prioritized CPU core being prevented from issuing an operation to the FPU. If this happens, the next CPU core that has a operation that can be started will be allowed to access the FPU and the current arbitration order will be saved. The arbitration order will be restored when the operation type that was prevented can be started. This allows the FPU resource

the be fairly shared between several CPU cores while at the same time allowing maximum utilization of the FPU.

In shared FPU configuration, GRFPU uses an 8 bit wide id for each operation. The three high-order bits are used to identify the CPU core which issued the FP operation, while the five low-order bits are used to enumerate FP operations issued by one core. FP operation flushing is not possible in shared FPU configuration.

56.5.2 Shared FPU and clock gating

Clock gating of LEON processors is typically implemented so that the clock for a processor core is gated off when the processor is idle. The clock for a shared FPU is typically gated off when the connected processors are all idle or have floating-point disabled.

This means that, in a shared FPU configuration, a processor may be clock gated off while the connected FPU continues to be clocked. The power-down instruction may overtake a previously issued floating-point instruction and cause the processor to be gated off before the floating-point operation has completed. This can in turn lead to the processor not reacting to the completion of the floating-point operation and to a subsequent processor freeze after the processor wakes up and continues to wait for the completion of the floating-point operation.

In order to avoid this, software must make sure that all floating-point operations have completed before the processor enters power-down. This is generally not a problem in real-world applications as the power-down instruction is typically used in a idle loop and floating-point results have been stored to memory before entering the idle loop. To make sure that there are no floating-point operations pending, software should perform a store of the %fsr register before the power-down instruction.

56.6 Implementation

56.6.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_async_reset_enable* is set.

57 GRFPC - GRFPU Control Unit

The GRFPU Control Unit (GRFPC) is used to attach the GRFPU to the LEON integer unit (IU). GRFPC performs scheduling, decoding and dispatching of the FP operations to the GRFPU as well as managing the floating-point register file, the floating-point state register (FSR) and the floating-point deferred-trap queue (FQ). Floating-point operations are executed in parallel with other integer instructions, the LEON integer pipeline is only stalled in case of operand or resource conflicts.

In the FT-version, all registers are protected with TMR and the floating-point register file is protected using parity coding.

57.1 Floating-Point register file

The GRFPU floating-point register file contains 32 32-bit floating-point registers (%f0-%f31). The register file is accessed by floating-point load and store instructions (LDF, LDDF, STD, STDF) and floating-point operate instructions (FPop).

57.2 Floating-Point State Register (FSR)

The GRFPC manages the floating-point state register (FSR) containing FPU mode and status information. All fields of the FSR register as defined in SPARC V8 specification are implemented and managed by the GRFPU conforming to the SPARC V8 specification and the IEEE-754 standard. Implementation-specific parts of the FSR managing are the NS (non-standard) bit and *ftt* field.

If the NS (non-standard) bit of the FSR register is set, all floating-point operations will be performed in non-standard mode as described in section 56.2.6. When the NS bit is cleared all operations are performed in standard IEEE-compliant mode.

Following floating-point trap types never occur and are therefore never set in the ftt field:
- unimplemented_FPop: all FPop operations are implemented
- hardware_error: non-resumable hardware error
- invalid_fp_register: no check that double-precision register is 0 mod 2 is performed

GRFPU implements the *qne* bit of the FSR register which reads 0 if the floating-point deferred-queue (FQ) is empty and 1 otherwise.

The FSR is accessed using LDFSR and STFSR instructions.

57.3 Floating-Point Exceptions and Floating-Point Deferred-Queue

GRFPU implements the SPARC deferred trap model for floating-point exceptions (fp_exception). A floating-point exception is caused by a floating-point instruction performing an operation resulting in one of following conditions:

- an operation raises IEEE floating-point exception (ftt = IEEE_754_exception) e.g. executing invalid operation such as 0/0 while the NVM bit of the TEM field id set (invalid exception enabled).
- an operation on denormalized floating-point numbers (in standard IEEE-mode) raises unfinished_FPop floating-point exception
- sequence error: abnormal error condition in the FPU due to the erroneous use of the floating-point instructions in the supervisor software.

The trap is deferred to one of the floating-point instructions (FPop, FP load/store, FP branch) following the trap-inducing instruction (note that this may not be next floating-point instruction in the program order due to exception-detecting mechanism and out-of-order instruction execution in the GRFPC). When the trap is taken the floating-point deferred-queue (FQ) contains the trap-inducing instruction and up to seven FPop instructions that were dispatched in the GRFPC but did not complete.

After the trap is taken the *qne* bit of the FSR is set and remains set until the FQ is emptied. The STDFQ instruction reads a double-word from the floating-point deferred queue, the first word is the address of the instruction and the second word is the instruction code. All instructions in the FQ are FPop type instructions. The first access to the FQ gives a double-word with the trap-inducing instruction, following double-words contain pending floating-point instructions. Supervisor software should emulate FPops from the FQ in the same order as they were read from the FQ.

Note that instructions in the FQ may not appear in the same order as the program order since GRFPU executes floating-point instructions out-of-order. A floating-point trap is never deferred past an instruction specifying source registers, destination registers or condition codes that could be modified by the trap-inducing instruction. Execution or emulation of instructions in the FQ by the supervisor software gives therefore the same FPU state as if the instructions were executed in the program order.

57.4 Implementation

57.4.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

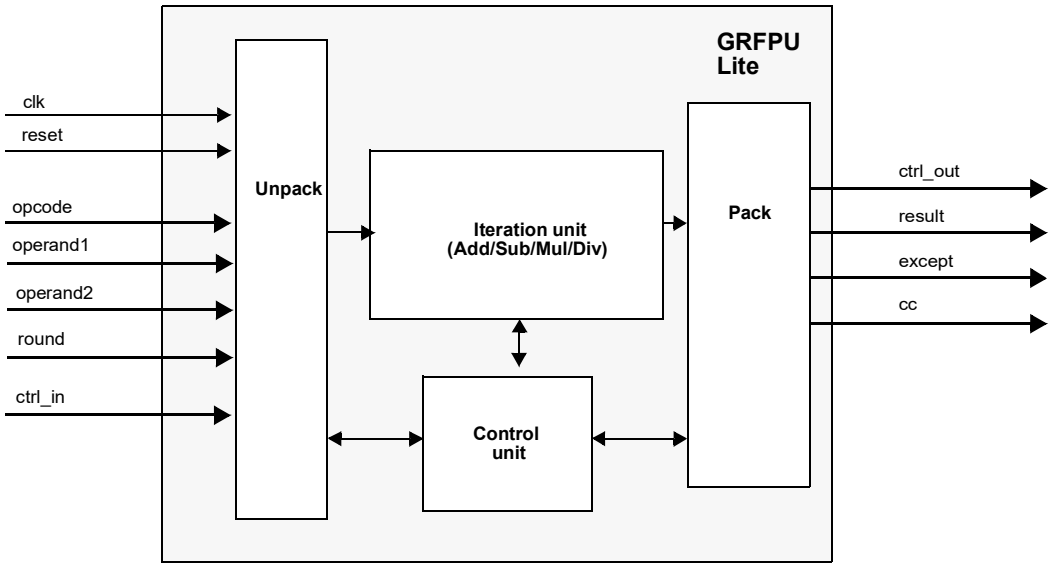
The GRFPC4 core will use asynchronous reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_async_reset_enable* is set. The GRFPC core does not support *grlib_async_reset_enable.*

58 GRFPU Lite - IEEE-754 Floating-Point Unit

58.1 Overview

The GRFPU Lite floating-point unit implements floating-point operations as defined in IEEE Standard for Binary Floating-Point Arithmetic (IEEE-754) and SPARC V8 standard (IEEE-1754).

Supported formats are single and double precision floating-point numbers. The floating-point unit is not pipelined and executes one floating-point operation at a time.



58.2 Functional Description

58.2.1 Floating-point number formats

The floating-point unit handles floating-point numbers in single or double precision format as defined in IEEE-754 standard.

58.2.2 FP operations

The floating-point unit supports four types of floating-point operations: arithmetic, compare, convert and move. The operations, summarised in the table below, implement all FP instructions specified by the SPARC V8 instruction set except FSMULD and instructions with quadruple precision.

Table 873.:Floating-point operations

Operation	Op1	Op2	Result	Exceptions	Description
Arithmetic operations					
FADDS FADDD	SP DP	SP DP	SP DP	NV, OF, UF, NX	Addition
FSUBS FSUBD	SP DP	SP DP	SP DP	NV, OF, UF, NX	Subtraction
FMULS FMULD	SP DP	SP DP	SP DP	NV, OF, UF, NX NV, OF, UF, NX	Multiplication
FDIVS FDIVD	SP DP	SP DP	SP DP	NV, OF, UF, NX, DZ	Division
FSQRTS FSQRTD	- -	SP DP	SP DP	NV, NX	Square-root
Conversion operations					
FITOS FITOD	-	INT	SP DP	NX -	Integer to floating-point conversion
FSTOI FDTOI	-	SP DP	INT	NV, NX	Floating-point to integer conversion. The result is rounded in round-to-zero mode.
FSTOD FDTOS	-	SP DP	DP SP	NV NV, OF, UF, NX	Conversion between floating-point formats
Comparison operations					
FCMPS FCMPD	SP DP	SP DP	CC	NV	Floating-point compare. Invalid exception is generated if either operand is a signaling NaN.
FCMPES FCMPED	SP DP	SP DP	CC	NV	Floating point compare. Invalid exception is generated if either operand is a NaN (quiet or signaling).
Negate, Absolute value and Move					
FABSS	-	SP	SP	-	Absolute value.
FNEGS	-	SP	SP	-	Negate.
FMOVS		SP	SP	-	Move. Copies operand to result output.

SP - single precision floating-point number

DP - double precision floating-point number

INT - 32 bit integer

CC - condition codes

NV, OF, UF, NX - floating-point exceptions, see section 58.2.3

Below is a table of worst-case throughput of the floating point unit.

*Table 874.*Worst-case instruction timing

Instruction	Throughput	Latency
FADDS, FADDD, FSUBS, FSUBD,FMULS, FMULD, FITOS, FITOD, FSTOI, FDTOI, FSTOD, FDTOS, FCMPS, FCMPD, FCMPES. FCMPED	8	8
FDIVS	31	31
FDIVD	57	57
FSQRTS	46	46
FSQRTD	65	65

58.2.3 Exceptions

The floating-point unit detects all exceptions defined by the IEEE-754 standard. This includes detection of Invalid Operation (NV), Overflow (OF), Underflow (UF), Division-by-Zero (DZ) and Inexact (NX) exception conditions. Generation of special results such as NaNs and infinity is also implemented.

58.2.4 Rounding

All four rounding modes defined in the IEEE-754 standard are supported: round-to-nearest, round-to-+inf, round-to--inf and round-to-zero.

58.3 Implementation

58.3.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

59 GRLFPC - GRFPU Lite Floating-point unit Controller

59.1 Overview

The GRFPU Lite Floating-Point Unit Controller (GRLFPC) is used to attach the GRFPU Lite floating-point unit (FPU) to the LEON integer unit (IU). It performs decoding and dispatching of the floating-point (FP) operations to the floating-point unit as well as managing the floating-point register file, the floating-point state register (FSR) and the floating-point deferred-trap queue (FQ).

The GRFPU Lite floating-point unit is not pipelined and executes only one instruction at a time. To improve performance, the controller (GRLFPC) allows the GRFPU Lite floating-point unit to execute in parallel with the processor pipeline as long as no new floating-point instructions are pending.

59.2 Floating-Point register file

The floating-point register file contains 32 32-bit floating-point registers (%f0-%f31). The register file is accessed by floating-point load and store instructions (LDF, LDDF, STD, STDF) and floating-point operate instructions (FPop).

In the FT-version, the floating-point register file is protected using 4-bit parity per 32-bit word. The controller is capable of detecting and correcting one bit error per byte. Errors are corrected using the instruction restart function in the IU.

59.3 Floating-Point State Register (FSR)

The controller manages the floating-point state register (FSR) containing FPU mode and status information. All fields of the FSR register as defined in SPARC V8 specification are implemented and managed by the controller conform to the SPARC V8 specification and IEEE-754 standard.

The non-standard bit of the FSR register is not used, all floating-point operations are performed in standard IEEE-compliant mode.

Following floating-point trap types never occur and are therefore never set in the ftt field:
- unimplemented_FPop: all FPop operations are implemented
- unfinished_FPop: all FPop operation complete with valid result
- invalid_fp_register: no check that double-precision register is 0 mod 2 is performed

The controller implements the *qne* bit of the FSR register which reads 0 if the floating-point deferred-queue (FQ) is empty and 1 otherwise. The FSR is accessed using LDFSR and STFSR instructions.

59.4 Floating-Point Exceptions and Floating-Point Deferred-Queue

The floating-point unit implements the SPARC deferred trap model for floating-point exceptions (fp_exception). A floating-point exception is caused by a floating-point instruction performing an operation resulting in one of following conditions:

- an operation raises IEEE floating-point exception (ftt = IEEE_754_exception) e.g. executing invalid operation such as 0/0 while the NVM bit of the TEM field id set (invalid exception enabled).
- sequence error: abnormal error condition in the FPU due to the erroneous use of the floating-point instructions in the supervisor software.
- hardware_error: uncorrectable parity error is detected in the FP register file

The trap is deferred to the next floating-point instruction (FPop, FP load/store, FP branch) following the trap-inducing instruction. When the trap is taken the floating-point deferred-queue (FQ) contains the trap-inducing instruction.

After the trap is taken the *qne* bit of the FSR is set and remains set until the FQ is emptied. STDFQ instruction reads a double-word from the floating-point deferred queue, the first word is the address of the instruction and the second word is the instruction code.

59.5 Implementation

59.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

60 GRGPRBANK - General Purpose Register Bank

60.1 Overview

The core provides an array of programmable registers. The registers drive an output vector that can be used to control other parts of the design.

60.2 Operation

The core contains registers of configurable number and length that is mapped into APB address space. The concatenated values of these registers are propagated to an output vector. All registers are reset to the same value, which is specified via a VHDL generic.

The core can optionally support a reset vector that allows setting the reset value on each bit. The core can also support a read vector that supply a different read-out value than the value of the registers. This can be used to implement read-only/write-only registers.

60.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 875.*General purpose register registers

APB address offset	Register
0x00	General purpose register 0
0x04	General purpose register 1
...	...
4*(N-1)	General purpose register (N-1)

*Table 876.*General purpose register N

31	regbits	regbits-1	0
RESERVED		REGISTER BITS	
0		*	
r		rw	

31:regbits RESERVED (not present if regbits = 32)

regbits-1:0 Register bits. Position i corresponds to bit (N*regbits+i) in the core's output vector

60.4 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x08F. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

60.5 Implementation

60.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset for its internal registers.

60.6 Configuration options

Table 877 shows the configuration options of the core (VHDL generics).

*Table 877.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
nregs	Number of registers	1 - 32	1
regbits	Number of register bits	1 - 32	32
rstval	Reset value for all registers	0 - 16#FFFFFFFF#	0
extrst	Support external reset vector	0 -1	0
rdataen	Support external read data vector	0 - 1	0

60.7 Signal descriptions

Table 878 shows the interface signals of the core (VHDL ports).

*Table 878.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
REGO	N/A	Output	Value of all registers concatenated	-
RESVAL	N/A	Input	Reset values for all registers concatenated (only used if extrst generic is set)	-
RDATA	N/A	Input	Read data for all registers concatenated (only used if rdata generic is set)	-

* see GRLIB IP Library User's Manual

60.8 Library dependencies

Table 879 shows the libraries used when instantiating the core (VHDL libraries).

*Table 879.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component, signals	Component declaration

60.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;

library gaisler;
use gaisler.misc.all;
```

```
entity grgprbank_ex is
  port (
    clkm  : in std_ulogic;
    rstn : in std_ulogic;
);
end;

architecture rtl of grgprbank_ex is
  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- Width of general purpose register
  constant GRGPRBANK_NBITS : integer := 10;
  constant GRGPRBANK_NREGS : integer := 5;

  signal rego : std_logic_vector(GRGPRBANK_NBITS*GRGPRBANK_NREGS-1 downto 0);

begin

  -- AMBA Components are instantiated here
  ...

  -- General purpose register bank
  grgprbank0 : grgprbank
    generic map (
      pindex  => 10,
      paddr   => 16#0a0#,
      pmask   => 16#fff#,
      regbits => GRGPRBANK_NBITS,
      nregs   => GRGPRBANK_NREGS,
      rstval  => 0)
    port map (
      rst     => rstn,
      clk     => clkm,
      apbi    => apbi,
      apbo    => apbo(10),
      rego    => rego);
end;
```

61 GRGPIO - General Purpose I/O Port

61.1 Overview

The general purpose input output port core is a scalable and provides optional interrupt support. The port width can be set to 2 - 32 bits through the *nbits* VHDL generic. Interrupt generation and shaping is only available for those I/O lines where the corresponding bit in the *imask* VHDL generic has been set to 1.

Each bit in the general purpose input output port can be individually set to input or output, and can optionally generate an interrupt. For interrupt generation, the input can be filtered for polarity and level/edge detection.

It is possible to share GPIO pins with other signals. The output register can then be bypassed through the bypass register.

The figure 149 shows a diagram for one I/O line.

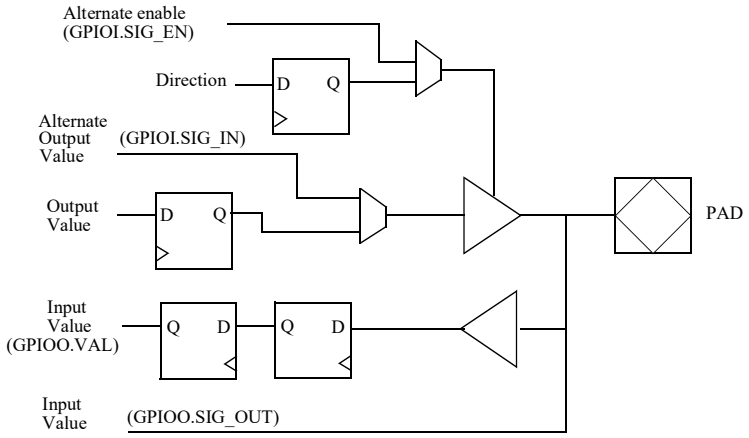


Figure 149. General Purpose I/O Port diagram

61.2 Operation

The I/O ports are implemented as bi-directional buffers with programmable output enable. The input from each buffer is synchronized by two flip-flops in series to remove potential meta-stability. The synchronized values can be read-out from the I/O port data register. They are also available on the GPIOO.VAL signals. The output enable is controlled by the I/O port direction register. A '1' in a bit position will enable the output buffer for the corresponding I/O line. The output value driven is taken from the I/O port output register.

The core can be implemented with one of three different alternatives for interrupt generation. Either each I/O line can drive a separate interrupt line on the APB interrupt bus, the interrupt line to use can be assigned dynamically for each I/O line, or one interrupt line can be shared for all I/O lines. In the fixed mapping with a separate interrupt line for each I/O line, the interrupt number is equal to the I/O line index plus an offset given by the first interrupt line assigned to the core, *pirq*, (PIO[1] = interrupt *pirq*+1, etc.). If the core has been implemented to support dynamic mapping of interrupts, each I/O line can be mapped using the Interrupt map register(s) to an interrupt line starting at interrupt *pirq*. When the core is implemented to drive one, fixed, shared interrupt line for all I/O lines, the core will drive interrupt line *pirq* only. The value of *pirq* can be read out from the core's AMBA plug'n'play information.

Interrupt generation is controlled by three registers: interrupt mask, polarity and edge registers. To enable an interrupt, the corresponding bit in the interrupt mask register must be set. If the edge register is '0', the interrupt is treated as level sensitive. If the polarity register is '0', the interrupt is active

low. If the polarity register is ‘1’, the interrupt is active high. If the edge register is ‘1’, the interrupt is edge-triggered. The polarity register then selects between rising edge (‘1’) or falling edge (‘0’).

The core can be implemented with a Interrupt flag register that can be used to determine if, and which, GPIO pin that caused an interrupt to be asserted. The core implements the Interrupt flag register, and the Interrupt available register, if the IFL field in the core’s capability register is non-zero.

A GPIO pin can be shared with other signals. The ports that should have the capability to be shared are specified with the *bypass* generic (the corresponding bit in the generic must be 1). The unfiltered inputs are available through GPIOO.SIG_OUT and the alternate output value must be provided in GPIOI.SIG_IN. The bypass register then controls whether the alternate output is chosen. The direction of the GPIO pin can also be shared, if the corresponding bit is set in the *bpdir* generic. In such case, the output buffer is enabled when GPIOI.SIG_EN is active. The direction of the pin can also be made to depend on the bypass register. See the documentation of the *bpmode* VHDL generic for details.

A GPIO pin can also be toggled when a pulse is detected on an internal signal. If the pulse VHDL generic is nonzero, then the Pulse register is available in the core.

61.3 Registers

The core is programmed through registers mapped into APB address space.

Table 880. General Purpose I/O Port registers

APB address offset	Register
0x00	I/O port data register
0x04	I/O port output register
0x08	I/O port direction register
0x0C	Interrupt mask register
0x10	Interrupt polarity register
0x14	Interrupt edge register
0x18	Bypass register
0x1C	Capability register
0x20 - 0x3C	Interrupt map register(s). Address 0x20 + 4*n contains interrupt map registers for IO[4*n : 3+4+n], if implemented.
0x40	Interrupt available register, if implemented
0x44	Interrupt flag register, if implemented
0x48	Input enable register, if implemented
0x4C	Pulse register, if implemented
0x50	Input enable register, if implemented, logical-OR
0x54	I/O port output register, logical-OR
0x58	I/O port direction register, logical-OR
0x5C	Interrupt mask register, logical-OR
0x60	Input enable register, if implemented, logical-AND
0x64	I/O port output register, logical-AND
0x68	I/O port direction register, logical-AND
0x6C	Interrupt mask register, logical-AND
0x70	Input enable register, if implemented, logical-XOR
0x74	I/O port output register, logical-XOR
0x78	I/O port direction register, logical-XOR
0x7C	Interrupt mask register, logical-XOR

61.3.1 I/O Port Data Register

Table 881. 0x00 - DATA - I/O port data register

31 nbits	nbits-1 0
RESERVED	DATA
0	*
r	r

nbits-1: 0 I/O port input value (DATA) - Data value read from GPIO lines

61.3.2 I/O Port Output Register

Table 882. 0x04 - OUTPUT - I/O port output register

31 nbits	nbits-1 0
RESERVED	DATA
0	0
r	rw

nbits-1: 0 I/O port output value (DATA) - Output value for GPIO lines

61.3.3 I/O Port Direction Register

Table 883. 0x08 - DIRECTION - I/O port direction register

31 nbits	nbits-1 0
RESERVED	DIR
0	0
r	rw

nbits-1: 0 I/O port direction value (DIR) - 0=output disabled, 1=output enabled

61.3.4 Interrupt Mask Register

Table 884. 0x0C - IMASK - Interrupt mask register

31 nbits	nbits-1 0
RESERVED	MASK
0	0
r	rw

nbits-1: 0 Interrupt mask (MASK) - 0=interrupt masked, 1=intrrupt enabled

61.3.5 Interrupt Polarity Register

Table 885. 0x10 - IPOL - Interrupt polarity register

31 nbits	nbits-1 0
RESERVED	POL
0	NR
r	rw

nbits-1: 0 Interrupt polarity (POL) - 0=low/falling, 1=high/rising

61.3.6 Interrupt Edge Register

*Table 886.*0x14 - IEDGE - Interrupt edge register

31 nbits	nbits-1 0
RESERVED	EDGE
0	NR
r	rw

nbits-1: 0 Interrupt edge (EDGE) - 0=level, 1=edge

61.3.7 Bypass Register

*Table 887.*0x18 - BYPASS - Bypass register

31 nbits	nbits-1 0
RESERVED	BYPASS
0	0
r	rw

nbits-1: 0 Bypass.(BYPASS) - 0=normal output, 1=alternate output

61.3.8 Capability Register

*Table 888.*0x1C - CAP - Capability register

31 18	17	16	15 13	12 8	7 5	4 0
RESERVED	PU	IER	IFL	r	IRQGEN	r	NLINES
0	*	*	*	0	*	0	*
r	r	r	r	r	r	r	r

18 PU: Pulse register implemented: If this field is '1' then the core implements the Pulse register.

This field is available in revision 2 and above of the GPIO port. This field is read-only.

17 IER: Input Enable register implemented. If this field is '1' then the core implements the Input enable register.

This field is available in revision 2 and above of the GPIO port. This field is read-only.

16 IFL: Interrupt flag register implemented. If this field is '1' then the core implements the Interrrupt available and Interrupt flag registers (registers at offsets 0x40 and 0x44).

This field is available in revision 2 and above of the GPIO port. This field is read-only.

12 8 IRQGEN: Interrupt generation setting: If irqgen = 0, I/O line n will drive interrupt line pirq + n, up to NAHBIRQ-1. No Interrupt map registers will be implemented. This is the default, and traditional, implementation of the core.

If irqgen = 1, all I/O lines capable of generating interrupts will use interrupt pirq and no Interrupt map registers are implemented.

If irqgen > 1, the core will include Interrupt map registers allowing software to dynamically map which lines that should drive interrupt lines [pirq : pirq+irqgen-1].

The value of pirq can be read out from the core's plug&play information.

This field is available in revision 2 and above of the GPIO port. This field is read-only.

4: 0 NLINES. Number of pins in GPIO port - 1. Compatibility note: This field is available in revision 2 and above of the GPIO port. This field is read-only.

61.3.9 Interrupt Map Register n

*Table 889.*0x20+un - IRQMAPRn - Interrupt map register n

31 29	28 24	23 21	20 16	15 13	12 8	7 6	4 0
RESERVED	IRQMAP[4*n]	RESERVED	IRQMAP[4*n+1]	RESERVED	IRQMAP[4*n+2]	RESERVED	IRQMAP[4*n+3]
0	0	0	0	0	0	0	0
r	rw	r	rw	r	rw	r	rw

31: 0 IRQMAP[i] : The field IRQMAP[i] determines to which interrupt I/O line i is connected. If IRQMAP[i] is set to *x*, IO[i] will drive interrupt *pirq+x*. Where *pirq* is the first interrupt assigned to the core. Several I/O can be mapped to the same interrupt.

The core has one IRQMAP field per I/O line. The Interrupt map register at offset 0x20+*4*n* contains the IRQMAP fields for IO[*4*n* : *4*n+3*]. This means that the fields for IO[0:3] are located on offset 0x20, IO[4:7] on offset 0x24, IO[8:11] on offset 0x28, and so on. An I/O line's interrupt generation must be enabled in the Interrupt mask register in order for the I/O line to drive the interrupt specified by the IRQMAP field. The Interrupt map register(s) can only be written if the core was implemented with support for interrupt mapping.

61.3.10 Interrupt Available Register

*Table 890.*0x40 - IAVAIL - Interrupt available register

31 0
IMASK
*
r

31: 0 IMASK: Interrupt mask bit field. If IMASK[n] is 1 then GPIO line n can generate interrupts.

This register is not available in all implementations. See capability register.

61.3.11 Interrupt Flag Register

*Table 891.*0x44 - IFLAG - Interrupt flag register

31 0
IFLAG
0
wc

31: 0 IFLAG : If IFLAG[n] is set to '1' then GPIO line n has generated an interrupt. Write '1' to the orresponding bit to clear. Writes of '0' have no effect.

This register is not available in all implementations, see capability register.

61.3.12 Input Enable Register

*Table 892.*0x48 - IPEN - Input enable register

31 0
IPEN
0
rw

31: 0 IPEN : If IPEN[n] is set to '1' then values from GPIO line n will be visible in the data register. Otherwise the GPIO line input is gated-off to disable input signal propagation.

This register is not available in all implementations, see capability register.

61.3.13 Pulse Register

*Table 893.*0x4C - PULSE - Pulse register

31	0
PULSE	
0	
rw	

31: 0 PULSE : If PULSE[n] is set to '1' then I/O port output register bit n will be inverted whenever GPIOI.SIG_IN[n] is high.

This register is not available in all implementations, see capability register.

61.3.14 Logical-OR/AND/XOR Register

*Table 894.*0x54-0x7C - LOR,LAND,LXOR - Logical-OR/AND/XOR registers

31	0
VALUE	
-	
w*	

31: 0 The logical-OR/AND/XOR registers will update the corresponding register according to:

New value = <Old value> logical-op <Write data>

There exists logical-OR, AND and XOR registers for the Input enable, I/O port output, I/O port direction and Interrupt mask registers.

61.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x01A. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

61.5 Implementation

61.5.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. See also the description for the *syncrst* VHDL generic for how the core implements reset.

61.6 Configuration options

Table 895 shows the configuration options of the core (VHDL generics).

*Table 895.*Configuration options

Generic	Function	Allowed range	Default
pindex	Selects which APB select signal (PSEL) will be used to access the GPIO unit	0 to NAHBIRQ-1	0
paddr	The 12-bit MSB APB address	0 to 16#FFF#	0
pmask	The APB address mask	0 to 16#FFF#	16#FFF#
nbits	Defines the number of bits in the I/O port	1 to 32	8

Table 895. Configuration options

Generic	Function	Allowed range	Default
imask	Defines which I/O lines are provided with interrupt generation and shaping. Note that line 31 (out of lines 0 to 31 in a 32-bit GPIO port) cannot be configured to assert interrupts.	0 - 16#7FFFFFF#	0
oepol	Select polarity of output enable signals. 0 = active low, 1 = active high.	0 - 1	0
syncrst	Selects between synchronous (1) or asynchronous (0) reset during power-up. 0: Logic is placed on the output enable signals to keep them as inputs while reset is enabled. 1: The register controlling the output enable signals are reset with an asynchronous reset.	0 - 1	0
bypass	Defines which I/O lines are provided bypass capabilities. Note that line 31 (out of lines 0 to 31 in a 32-bit GPIO port) cannot be configured for bypass.	0 - 16#7FFFFFF#	0
scantest	Enable scan support for asynchronous-reset flip-flops	0 - 1	0
bpdir	Defines which I/O lines are provided output enable bypass capabilities. Note that line 31 (out of lines 0 to 31 in a 32-bit GPIO port) cannot be configured for bypass.	0 - 16#7FFFFFF#	0
pirq	First interrupt line that the core will drive. The core will only drive interrupt lines up to line NAHBIRQ-1. If NAHBIRQ is set to 32 and *pirq* is set to 16, the core will only be able to generate interrupts for I/O lines 0 - 15.	0 - NAHBIRQ-1	0
irqgen	This generic configures interrupt generation. If *irqgen* = 0, I/O line *n* will drive interrupt line *pirq* + *n*, up to NAHBIRQ-1. No Interrupt map registers will be implemented. This is the default, and traditional, implementation of the core. If *irqgen* = 1, all I/O lines capable of generating interrupts will use interrupt *pirq* and no Interrupt map registers will be implemented. If *irqgen* > 1, the core will include Interrupt map registers allowing software to dynamically map which lines that should drive interrupt lines [*pirq* : *pirq*+*irqgen*-1]	0 - NAHBIRQ-1	0
iflagreg	If this generic is set to 1 then the core will be implemented with the Interrupt available and Interrupt flag registers. If this generic is set to 1 then the IFL field in the core's capability register is also set.	0 - 1	0
bpmode	Controls if output enable bypass depends on the bypass register and the behaviour of the gpioi.sig_en inputs. If bpmode = 0 then gpioi.sig_en(i) enables GPIO line *i* and connects gpioi.sig_in(i) to the output, regardless of the value in the bypass register. If bpmode = 1 then the corresponding bit in the bypass register must be set to '1' for the output enable bypass to be active. When the bypass register is active then the gpioi.sig_en(i) controls the output enable of the corresponding GPIO line and gpioi.sig_in(i) is connected to the corresponding output. In both cases, bit *i* in the bpdir VHDL generic must be set for the gpioi.sig_en(i) signal to have any effect.	0 - 1	0
inpen	If this generic is non-zero, then the core will be implemented with the Input enable register that can be used to prevent input signal propagation.	0 - 1	0

*Table 895.*Configuration options

Generic	Function	Allowed range	Default
doutresv	Reset value for output register		0
dirresv	Reset value for direction register		0
bpresv	Reset value for bypass register		0
inpresv	Reset value for input enable register (if implemented)		0
pulse	If this generic is non-zero, then the core will be implemented with the Pulse register and corresponding functionality.	0 - 1	0

61.7 Signal descriptions

Table 896 shows the interface signals of the core (VHDL ports).

*Table 896.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
GPIOO	OEN[31:0]	Output	I/O port output enable	see oepol
	DOUT[31:0]	Output	I/O port outputs	-
	VAL[31:0]	Output	The current (synchronized) value of the GPIO signals	-
	SIG_OUT[31:0]	Output	The current (unsynchronized) value of the GPIO signals. Note that this value is affected by the Input enable register, if implemented.	
GPIOI	DIN[31:0]	Input	I/O port inputs	-
	SIG_IN[31:0]	Input	Alternative output	-
	SIG_EN[31:0]	Input	Alternative output enable	High

* see GRLIB IP Library User's Manual

61.8 Signal definitions and reset values

The signals and their reset values are described in table 897.

*Table 897.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
gpio[]	Input/Output	General purpose input output	-	Tri-state

61.9 Timing

The timing waveforms and timing parameters are shown in figure 150 and are defined in table 898.

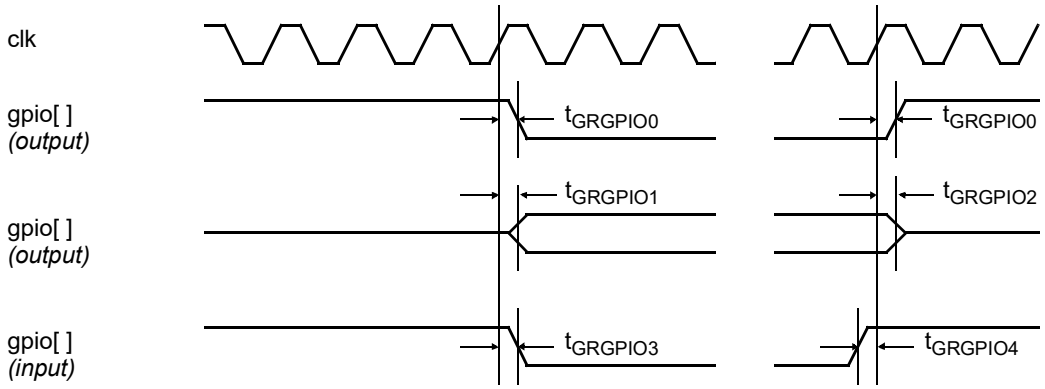


Figure 150. Timing waveforms

Table 898. Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{GRGPIO0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns
$t_{GRGPIO1}$	clock to non-tri-state delay	rising *clk* edge	TBD	TBD	ns
$t_{GRGPIO2}$	clock to tri-state delay	rising *clk* edge	TBD	TBD	ns
$t_{GRGPIO3}$	input to clock hold	rising *clk* edge	-	-	ns
$t_{GRGPIO4}$	input to clock setup	rising *clk* edge	-	-	ns

Note: The *gpio* inputs are re-synchronized internally. The signals do not have to meet any setup or hold requirements.

61.10 Library dependencies

Table 899 shows libraries used when instantiating the core (VHDL libraries).

Table 899. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Signals, component	Component declaration

61.11 Component declaration

The core has the following component declaration.

```
ibrary gaisler;
use gaisler.misc.all;

entity grgpio is
  generic (
    pindex   : integer := 0;
    paddr    : integer := 0;
    pmask    : integer := 16#fff#;
    imask    : integer := 16#0000#;
    nbits    : integer := 16-- GPIO bits

  );
  port (
```

```
    rst    : in  std_ulogic;
    clk    : in  std_ulogic;
    apbi   : in  apb_slv_in_type;
    apbo   : out apb_slv_out_type;
    gpioi  : in  gpio_in_type;
    gpioo  : out gpio_out_type
  );
end;
```

61.12 Instantiation

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

signal gpti : gptimer_in_type;

begin

gpio0 : if CFG_GRGPIO_EN /= 0 generate     -- GR GPIO unit
    grgpio0: grgpio
      generic map( pindex => 11, paddr => 11, imask => CFG_GRGPIO_IMASK, nbits => 8)
      port map( rstn, clkm, apbi, apbo(11), gpioi, gpioo);

      pio_pads : for i in 0 to 7 generate
        pio_pad : iopad generic map (tech => padtech)
            port map (gpio(i), gpioo.dout(i), gpioo.oen(i), gpioi.din(i));
      end generate;
end generate;
```

62 GRGPREG - General Purpose Register

62.1 Overview

The core provides a programmable register that drives an output vector that can be used to control miscellaneous options in a design.

62.2 Operation

The core contains one register of configurable length that is mapped into APB address space. The value of this register is propagated to an output vector. The reset value of the register can be specified via VHDL generics, or via an input vector.

62.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 900.*General purpose register registers

APB address offset	Register
0x00	General purpose register bits 31:0
0x04	General purpose register bits 63:0

62.3.1 General Purpose Register

*Table 901.*0x00 - GPREG0 - General purpose register

31 nbits	nbits-1 0
RESERVED	REGISTER BITS
0	*
r	rw

31:nbits RESERVED (not present if nbits >= 32)

nbits-1:0 Register bits. Position i corresponds to bit i in the core's output vector

62.3.2 General Purpose Register (extended)

*Table 902.*0x04 - GRPEG1 - General purpose register (extended)

31 nbits-32	nbits-33 0
RESERVED	REGISTER BITS
0	*
r	rw

31:nbits-32 RESERVED (not present if nbits = 64 or nbits <= 32)

nbits-33:0 Register bits. Position i corresponds to bit 31+i in the core's output vector (not present if nbits < 33)

62.4 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x087. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

62.5 Implementation

62.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset for its internal registers.

62.6 Configuration options

Table 903 shows the configuration options of the core (VHDL generics).

*Table 903.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
nbits	Number of register bits	1 - 64	16
rstval	Reset value for bits 31:0	0 - 16#FFFFFFFF#	0
rstval2	Reset value for bits 63:32	0 - 16#FFFFFFFF#	0
extrst	Use input vector *resval* to specify reset value. If this generic is 0 the register reset value is determined by VHDL generics *rstval* and *rstval2*. If this generic is 1, the reset value is specified by the input vector *resval*.	0 - 1	0

62.7 Signal descriptions

Table 904 shows the interface signals of the core (VHDL ports).

*Table 904.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
GRGPREGO	N/A	Output	Value of register mapped into APB address space	-
RESVAL	N/A	Input	(Optionally) Specifies register reset value	-

* see GRLIB IP Library User's Manual

62.8 Library dependencies

Table 905 shows the libraries used when instantiating the core (VHDL libraries).

*Table 905.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component, signals	Component declaration, I2C signal definitions

62.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
```

```
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;

library gaisler;
use gaisler.misc.all;

entity grgpreg_ex is
  port (
    clkm  : in std_ulogic;
    rstn : in std_ulogic;

    -- I2C signals
    iic_scl : inout std_ulogic;
    iic_sda : inout std_ulogic
    );
end;

architecture rtl of i2c_ex is
  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- Width of general purpose register
  constant GRGPREG_NBITS : integer := 9;

  signal gprego           : std_logic_vector(GRGPREG_NBITS-1 downto 0);
  signal gpregresval      : std_logic_vector(GRGPREG_NBITS-1 downto 0);
begin

  -- AMBA Components are instantiated here
  ...

  -- General purpose register
  grgpreg0 : grgpreg                  -- General purpose register
    generic map (
      pindex  => 10,
      paddr   => 16#0a0#,
      pmask   => 16#fff#,
      nbits   => GRGPREG_NBITS,
      rstval  => 0,                   -- Not used
      rstval2 => 0,                   -- Not used
      extrst  => 1)                   -- Use input vector for reset value
    port map (
      rst     => rstn,
      clk     => clkm,
      apbi    => apbi,
      apbo    => apbo(10),
      gprego  => gprego,
      resval  => gpregresval);
end;
```

63 GRHSSL - SpaceFibre and WizardLink controller with DMA engine

63.1 Overview

GRHSSL provides an interface between the system bus (such as AHB or AXI) and a high-speed serial link. It can implement either SpaceFibre or WizardLink controllers, or both. The SpaceFibre controller complies with the SpaceFibre specification ECSS-E-ST-50-11C, whereas the WizardLink codec has been designed to inter-operate with the TLK2711 SerDes transceiver from Texas Instrument, although the IP can interface other SerDes devices.

The core features a flexible DMA layer with a configurable number of DMA channels, each one with its own bus master interface to fetch and store descriptors and data from/to external memory. There are wrappers available adapting the generic bus master interfaces to either AMBA AHB 2.0 or AXI4. The packets are autonomously handled by the DMA engine, both in transmission and reception. The user controls the process via hardware descriptors. The optional Remote Memory Access Protocol (RMAP) target, adhering to the ECSS-E-ST-50-52C standard, can be implemented in the SpaceFibre layer.

The IP is configured through registers accessed through an AHB slave interface. SpaceFibre and WizardLink have separate sets of AHB registers, meaning that the IP will include two AHB I/O banks if both controllers are generated.

If both controllers are present, only one can be active at a time, which is selectable at run time via the AHB registers. The active controller will drive the bus master and SerDes interfaces.

The block of GRHSSL can be seen in the picture below.

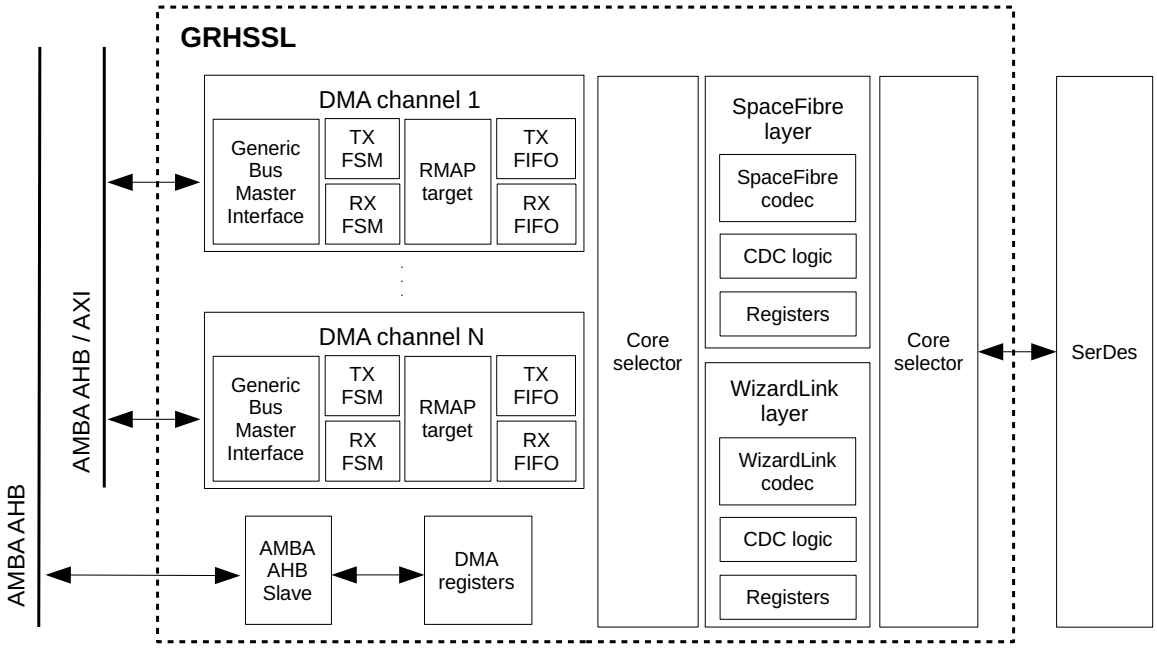


Figure 151. GRHSSL block diagram

63.2 Operation

63.2.1 Functional description

The IP can be split into the following parts from a functional point of view:

- SpaceFibre controller.
- WizardLink controller.

- DMA engine with optional RMAP target for SpaceFibre.
- AHB slave interface.
- Clock-domain-crossing synchronization mechanism.
- DMA FIFOs.

The SpaceFibre controller in GRHSSL is the GRSPFI IP, which is described in a separate entry of this document. Please refer to that chapter for a detailed description of the IP functionality, configuration and register interface. GRSPFI is generated if the VHDL generic *spfi* is set to 1. If so, GRHSSL will include a dedicated AHB I/O bank for the SpaceFibre registers, regardless of whether WizardLink is also present or not.

The WizardLink controller in GRHSSL is the GRWIZL IP, which is described in a separate entry of this document. Please refer to that chapter for a detailed description of the IP functionality, configuration and register interface. GRWIZL is generated if the VHDL generic *wizl* is set to 1. If so, GRHSSL will include a dedicated AHB I/O bank for the WizardLink registers, regardless of whether SpaceFibre is also present or not.

The DMA engine is shared by both SpaceFibre and WizardLink controller. It contains four finite-state machines:

- SpaceFibre TX FSM
- SpaceFibre RX FSM
- WizardLink TX FSM
- WizardLink RX FSM

The active protocol determines which pair of FSMs can be in a state other than idle. When selected, the corresponding DMA channels will internally drive the bus master interfaces and will connect the inputs and outputs of the selected codec to the SerDes interface.

Note that GRWIZL does not support more than one bus master interface. If the IP is instantiated with multiple DMA channels, the rest is only used for SpaceFibre packets, and remain unused when operating in WizardLink mode.

The DMA engine also performs the clock-domain crossing synchronization between the AMBA and codec clock domains by means of handshake mechanisms and dual port FIFOs. It also contains the AHB registers used to configure and control the IP.

63.2.2 Protocol support

The SpaceFibre codec has been designed in accordance with the SpaceFibre standard ECSS-E-ST-50-11C. It is a Single-Lane implementation of the protocol; therefore the optional Multi-Lane layer is not part of the IP. The optional RMAP target has been implemented as per the ECSS-E-ST-50-52C specification.

The WizardLink codec has been designed to interface the TLK2711 transceiver with minimal protocol overhead. Other SerDes models are supported by the IP.

63.2.3 Endianness

The core is designed for big-endian and little-endian systems.

Please refer to the GRSPFI and GRWIZL sections of this manual for further detail regarding how the endianness support is implemented in each core.

63.3 Protocol selection

When both controllers are instantiated, the user shall select which protocol is active. This can be done in runtime by means of AHB registers.

Both GRSPFI and GRWIZL contains an Enable register that allows to set the respective codec as the active controller. These registers are called SpaceFibre Enable Register and WizardLink Enable Register. Note that these registers become read-only if GRHSSL only implements a single protocol.

The user must ensure that the active controller is completely idle before switching modes. This means that the DMA engine must be off, both transmitter and receiver for all DMA channels. Switching modes while the IP is still processing a descriptor or a packet will most likely corrupt the normal operation and/or provoke link errors.

The general scenario is to select the active protocol as part of the setup of the IP, without needing to switch to the alternative mode at any later time. Only in special cases (e.g. two GRHSSL IPs interfacing one another, or if GRHSSL interfaces a remote node that also supports dual protocol) it is expected to switch the functional mode in the middle of the operation.

63.4 Registers

The core is programmed through registers mapped into AHB address space. Only 32-bit single accesses to the registers are supported.

SpaceFibre and WizardLink registers are implemented in separate AHB I/O areas. Please refer to the respective chapters in this document (GRSPFI and GRWIZL) for a complete description of their register interfaces.

63.5 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler). The device identifier changes depending on the configuration of the IP:

- 0x0BC (GRSPFI) when GRHSSL only implements SpaceFibre (generics *spfi* = 1 and *wizl* = 0)
- 0x0C7 (GRWIZL) when GRHSSL only implements WizardLink (generics *spfi* = 0 and *wizl* = 1)
- 0x0C8 (GRHSSL) when both controllers are implemented (generics *spfi* = 1 and *wizl* = 1)

For description of vendor and device identifiers see GRLIB IP Library User's Manual.

63.6 Configuration options

Table 906 shows the configuration options of the core (VHDL generics).

Table 906. Configuration options

Generic name	Function	Allowed range	Default
tech	Technology for the internal memories	0 - NTECH	inferred (0)
hmindex	AHB master index of the first DMA channel. The rest of the DMA channels are assigned an index depending on the settings of the generic *incr_hmindex*. Only present in the AHB wrapper.	0 - NAHBMST-1	0
hsindex	AHB slave index	0 - NAHBSLV-1	0
haddr_spfi	Addr field of the SpaceFibre AHB bar	0 - 16#FFF#	0
hmask_spfi	Mask field of the SpaceFibre AHB bar	0 - 16#FFF#	16#FF0#
haddr_wizl	Addr field of the WizardLink AHB bar	0 - 16#FFF#	0
hmask_wizl	Mask field of the WizardLink AHB bar	0 - 16#FFF#	16#FF0#
hirq	Interrupt line used by GRHSSL	0 - NAHBIRQ-1	0
use_8b10b	If set, internal 8B10B encoding and decoding is activated in both SpaceFibre and WizardLink codecs.	0 - 1	1
use_sep_txclk	If set, the SerDes transmission clock is decoupled from the SpaceFibre/WizardLink core clock. An additional transmit buffer is instantiated for this reason.	0 - 1	0

*Table 906.*Configuration options

Generic name	Function	Allowed range	Default
sel_16_20_bit_-mode	If set, the SerDes interface is 16+2 bit (without 8B10B) or 20 bit (with 8B10B) wide instead of 36/40 bit. If set, *use_sep_txclk* must also be set. Applies to both codecs.	0 - 1	0
ticks_2us	Clock ticks corresponding to 2 us. Only used by the SpaceFibre codec.	1 - 8192	125
tx_skip_freq	Frequency of SpaceFibre SKIP word transmission in clock cycles.	1 - 8192	5000
prbs_init1	If set, the INIT1 sequence during SpaceFibre lane initial-ization is embedded into a stream of pseudo-random numbers.	0 - 1	1
depth_rbuf_data	Log(Depth) of the SpaceFibre data retry buffer.	1 - 32	8
depth_rbuf_fct	Log(Depth) of the SpaceFibre FCT retry buffer.	1 - 32	4
depth_rbuf_bc	Log(Depth) of the SpaceFibre broadcast retry buffer.	1 - 32	8
num_vc	Number of SpaceFibre virtual channels.	1 - 32	4
fct_multiplier	Multiplier used in the SpaceFibre internal FCT logic. The value of the FCT counter is compared with 64**fct_-multiplier* to decide if the far end can receive new words.	1 - 8	1
depth_vc_rx_buf	Log(Depth) of SpaceFibre virtual channel input buffer.	1 - 32	10
depth_vc_tx_buf	Log(Depth) of SpaceFibre virtual channel output buffer.	1 - 32	10
remote_fct_cnt_-max	Width of the SpaceFibre remote FCT counter.	1 - 32	9
width_bw_credit	Width of the SpaceFibre bandwidth credit counter.	0 - 16#FFFFFFFF#	20
min_bw_credit	SpaceFibre minimum bandwidth credit threshold limit.	0 - 16#FFFFFFFF#	52428
idle_time_limit	SpaceFibre bandwidth idle time limit in clock cycles.	0 - 65535	62500
num_dmach	Number of DMA channels. Note that only the first DMA channel is used when WizardLink is the active protocol.	1 - 8	1
num_txdesc	Size of the TX descriptor table in number of descriptors.	64, 128, 256, 512	64
num_rxdesc	Size of the RX descriptor table in number of descriptors.	128, 256, 512, 1024	128
depth_dma_fifo	Depth of the DMA FIFOs. Sets the number of entries in the transmitter and receiver FIFOs (2 FIFOs per DMA channel). The FIFOs are shared by both codecs.	16 - 256	32
depth_bc_fifo	Depth of the BC FIFO. Sets the number of entries in the receiver FIFO. Only used by the SpaceFibre layer.	1 - 32	4
depth_wizl_tx_buf	Log(Depth) of the WizardLink codec internal TX buffer, expressed as number of address bits.	1 - 32	10
depth_wizl_rx_buf	Log(Depth) of the WizardLink codec internal RX buffer, expressed as number of address bits.	1 - 32	10
incr_hmindex	If set, the AHB master indices of the DMA channels are assigned incrementally, starting with hmindex. Other-wise, all AHB master indices are set to hmindex. Only present in the AHB wrapper.	0 - 1	1
use_async_rxrst	Force asynchronous reset in the recovered clock domain, regardless of the general GRLIB settings.	0 - 1	0
ft_core_vc	Enable fault-tolerance against SEU errors for the FIFOs in the SpaceFibre Virtual Channel Layer and in the WizardLink codec. Possible options are byte parity protection (ft = 1), TMR registers (ft = 2), SECDED BCH (ft = 4) or technology specific protection (ft = 5). If set to 0, no protection is implemented.	0 - 2, 4 - 5	0

Table 906. Configuration options

Generic name	Function	Allowed range	Default
ft_core_rt1	Enable fault-tolerance against SEU errors for the large FIFOs in the SpaceFibre Retry Layer (Data and Broadcast). The core supports the same configurations as above.	0 - 2, 4 - 5	0
ft_core_rt2	Enable fault-tolerance against SEU errors for the small FIFOs in the SpaceFibre Retry Layer (FCT and pointers). The core supports the same configurations as above.	0 - 2, 4 - 5	0
ft_core_if	Enable fault-tolerance against SEU errors for the FIFOs in the Interface Layer of the SpaceFibre and the WizardLink codecs. The core supports the same configurations as above.	0 - 2, 4 - 5	0
ft_dma_data	Enable fault-tolerance against SEU errors for the TX/RX FIFOs in the DMA layer. The core supports the same configurations as above.	0 - 2, 4 - 5	0
ft_dma_bc	Enable fault-tolerance against SEU errors for the SpaceFibre Broadcast Channel FIFOs in the DMA layer. The core supports the same configurations as above.	0 - 2, 4 - 5	0
scantest	Enable Scan Test support.	0 - 1	0
ahbbits	Width of AHB read/write data buses and maximum access size. Only present in the AHB wrapper.	32, 64, 128	AHBDW
lendian	Select endianness of the system: 0 for big-endian, 1 for little-endian. Only present in the AXI wrapper.	0 - 1	1
spfi	If set, the SpaceFibre codec is instantiated in the design.	0 - 1	1
wizl	If set, the WizardLink codec is instantiated in the design.	0 - 1	0
rmap	If set, every DMA channel will feature a hardware RMAP target and RMAP CRC logic. Only used by the SpaceFibre layer.	0 - 1	0
rmapcrc	Include RMAP CRC logic in every DMA channel without implementing a full RMAP target. Only used by the SpaceFibre layer.	0 - 1	0
ccsdscrc	Enable CCSDS/CCITT CRC-16 and 16-bit ISO-checksum (J.G. Fletcher, ISO 8473-1:1998) logic in every DMA channel. It requires that either *rmap* or *rmapcrc* is also set to 1. Only used by the SpaceFibre layer.	0 - 1	0
nodeaddr	When set to 0 - 254: this generic specifies the reset value for the default address field. When set to 255: Reset value for the node address is taken from the CFG.NODEADDR input signal. Only used by the SpaceFibre layer.	0 - 255	254
destkey	Reset value for the core destination key. Only used by the SpaceFibre layer.	0 - 255	0
numextvc	Number of external FIFO interfaces that can be used to bypass the DMA engine when transmitting and receiving packets over a specific virtual channel. Only used by the SpaceFibre layer.	0 - 32	0
numextbc	If set, implement an external FIFO interface that can be used to bypass the DMA engine when transmitting and receiving broadcast messages. Only used by the SpaceFibre layer.	0 - 1	0
numextwc	If set, implement an external FIFO interface that can be used to bypass the DMA engine when transmitting and receiving WizardLink data. Only used by the WizardLink layer.	0 - 1	0

63.7 Signal descriptions

Table 907 shows the interface signals of the core when using the wrapper for AMBA AHB 2.0 (VHDL ports).

Table 907. Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	System reset	Low
CLK	N/A	Input	AMBA clock	-
CORE_RSTN	N/A	Input	Core reset	Low
CORE_CLK	N/A	Input	Core clock	-
CORE_TXCLK	N/A	Input	Optional core transmit clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
AHBMI	*	Input	Array of AHB master input signals	-
AHBMO	*	Output	Array of AHB master output signals	-
HSSLI	RX_CLK	Input	SerDes receive clock	-
	RX_DATA	Input	Receive Data	-
	RX_KFLAGS	Input	Receive K-Flags	-
	RX_SERROR	Input	Receive Error Flags	High
	NO_SIGNAL	Input	SerDes No Signal Flag	High
HSSLO	TX_DATA	Output	Transmit Data	-
	TX_KFLAGS	Output	Transmit K-Flags	-
	TX_EN	Output	SerDes Transmitter Enable Flag	High
	RX_EN	Output	SerDes Receiver Enable Flag	High
	INV_POL	Output	SerDes Invert Polarity Flag	High
	TX_DATA_DBG	Output	Unencoded 32-bit transmit data	-
	TX_KFLAGS_DBG	Output	4-bit transmit K-Flags	-
	RX_DATA_DBG	Output	Unencoded 32-bit receive data	-
	RX_KFLAGS_DBG	Output	4-bit receive K-Flags	-
CFG	RMAPEN	Input	Reset value of the RMAP Enable bit of the SpaceFibre Virtual Channel 0. Used only if the *rmap* generic is also set.	High
	NODEADDR	Input	Reset value for the SpaceFibre default address when the *nodeaddr* VHDL generic = 255. Unused if *nodeaddr* /= 255.	-
EXTVCI[31:0]	TX_DATA	Input	Virtual channel transmit data when using the external FIFO interface.	-
	TX_KFLAGS	Input	Virtual channel transmit K-flags when using the external FIFO interface.	-
	TX_WEN	Input	Virtual channel transmit write enable when using the external FIFO interface.	High
	RX_REN	Input	Virtual channel receive read enable when using the external FIFO interface.	High

Table 907. Signal descriptions

Signal name	Field	Type	Function	Active
EXTVCO [31:0]	TX_FULL	Output	Virtual channel transmit full flag when using the external FIFO interface.	High
	RX_DATA	Output	Virtual channel receive data when using the external FIFO interface.	-
	RX_KFLAGS	Output	Virtual channel receive K-flags when using the external FIFO interface.	-
	RX_VALID	Output	Virtual channel receive valid flag when using the external FIFO interface.	High
EXTBCI	TX_DATA	Input	Broadcast transmit data when using the external FIFO interface.	-
	TX_CHANNEL	Input	Broadcast transmit channel when using the external FIFO interface.	-
	TX_BTYPE	Input	Broadcast transmit broadcast type when using the external FIFO interface.	-
	TX_DELAYED	Input	Broadcast transmit delayed flag when using the external FIFO interface.	High
	TX_LATE	Input	Broadcast transmit late flag when using the external FIFO interface.	High
	TX_WEN	Input	Broadcast transmit write enable when using the external FIFO interface.	High
EXTBCO	TX_ACK	Output	Broadcast transmit acknowledge flag when using the external FIFO interface.	High
	RX_DATA	Output	Broadcast receive data when using the external FIFO interface.	-
	RX_CHANNEL	Output	Broadcast receive channel when using the external FIFO interface.	-
	RX_BTYPE	Output	Broadcast receive broadcast type when using the external FIFO interface.	-
	RX_VALID	Output	Broadcast receive valid flag when using the external FIFO interface.	High
	RX_DELAYED	Output	Broadcast receive delayed flag when using the external FIFO interface.	High
	RX_LATE	Output	Broadcast receive late flag when using the external FIFO interface.	High
EXTWCI	TX_DATA	Input	WizardLink transmit data when using the external FIFO interface.	-
	TX_KFLAGS	Input	WizardLink transmit K-flags when using the external FIFO interface.	-
	TX_WEN	Input	WizardLink transmit write enable when using the external FIFO interface.	High
	RX_REN	Input	WizardLink receive read enable when using the external FIFO interface.	High
EXTWCO	TX_FULL	Output	WizardLink transmit full flag when using the external FIFO interface.	High
	RX_DATA	Output	WizardLink receive data when using the external FIFO interface.	-
	RX_KFLAGS	Output	WizardLink receive K-flags when using the external FIFO interface.	-
	RX_VALID	Output	WizardLink receive valid flag when using the external FIFO interface.	High

Table 907. Signal descriptions

Signal name	Field	Type	Function	Active
MTESTI**	SPFI	Input	Memory BIST input signals to the FIFOs in the SpaceFibre codec. Please refer to the GRSPFI_CODEC section of this document for further details about the fields of this record	-
	WIZL	Input	Memory BIST input signals to the FIFOs in the WizardLink codec. Please refer to the GRWIZL section of this document for further details about the fields of this record	-
	DMA_TX	Input	Memory BIST input signal to the DMA TX FIFO	-
	DMA_RX	Input	Memory BIST input signal to the DMA RX FIFO	-
	DMA_BC	Input	Memory BIST input signal to the DMA Broadcast FIFO	-
MTESTO**	SPFI	Output	Memory BIST output signals from the FIFOs in the SpaceFibre codec. Please refer to the GRSPFI_CODEC section of this document for further details about the fields of this record	-
	WIZL	Output	Memory BIST output signals from the FIFOs in the WizardLink codec. Please refer to the GRWIZL section of this document for further details about the fields of this record	
	DMA_TX	Output	Memory BIST output signal from the DMA TX FIFO	-
	DMA_RX	Output	Memory BIST output signal from the DMA RX FIFO	-
	DMA_BC	Output	Memory BIST output signal from the DMA Broadcast FIFO	-
MTESTCLK **	N/A	Input	Memory BIST clock	-

* see GRLIB IP Library User's Manual

** not available in FPGA releases

63.8 Library dependencies

Table 908 shows the libraries used when instantiating the core (VHDL libraries).

Table 908. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GRLIB	GENERIC BM PKG	Signals, components	Package including the generic bus master interface and the bridges for converting to AHB and AXI
GAISLER	HSSL	Signals, component	GRHSSL component and signal declarations

63.9 Instantiation

This example shows how the core can be instantiated using the AMBA 2.0 AHB wrapper.

```
library  ieee;
use      ieee.std_logic_1164.all;
library  gaisler;
use      gaisler.hssl.all;

entity example is
```

```
   generic (
      tech:       in    integer := 0);
   port (
   );

-- Signal declarations
signal clk       : std_ulogic;
signal rstn      : std_ulogic;
signal hssl_clk  : std_ulogic;
signal hssl_rstn : std_ulogic;

signal ahbmi_hssl : ahb_mst_in_vector_type(1 downto 0);
signal ahbmo_hssl : ahb_mst_out_vector_type(1 downto 0);

signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
signal ahbmi : ahb_mst_in_type;
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);

signal hssli : grhssl_in_type;
signal hsslo : grhssl_out_type;

------------------------------------------------------------------------------
-- Component instantiation
------------------------------------------------------------------------------

hssl_dut : grhssl_ahb
      generic map (
        tech                => 0,
        hmindex             => 1,
        hsindex             => 7,
        haddr_spfi          => 16#A00#,
        hmask_spfi          => 16#FF0#,
        haddr_wizl          => 16#A10#,
        hmask_wizl          => 16#FFF#,
        hirq                => 1,
        use_8b10b           => 1,
        use_sep_txclk       => 0,
        sel_16_20_bit_mode  => 0,
        ticks_2us           => 125,
        tx_skip_freq        => 5000,
        prbs_init1          => 1,
        depth_rbuf_data     => 8,
        depth_rbuf_fct      => 4,
        depth_rbuf_bc       => 8,
        num_vc              => 4,
        fct_multiplier      => 1,
        depth_vc_rx_buf     => 10,
        depth_vc_tx_buf     => 10,
        remote_fct_cnt_max  => 9,
        width_bw_credit     => 20,
        min_bw_credit       => 52428,
        idle_time_limit     => 62500,
        num_dmach           => 2,
        num_txdesc          => 256,
        num_rxdesc          => 512,
        depth_dma_fifo      => 32,
        depth_bc_fifo       => 4,
        depth_wizl_tx_buf   => 10,
        depth_wizl_rx_buf   => 10,
        incr_hmindex        => 1,
        use_async_rxrst     => 1,
        spfi                => 1,
        wizl                => 1)
      port map (
        clk        => clk,
        rstn       => rstn,
        core_clk   => hssl_clk,
        core_rstn  => hssl_rstn,
        core_txclk => '0',               -- Not using separate TX clock
        -- AHB interface
```

```
ahbmi      => ahbmi_hssl,
ahbmo      => ahbmo_hssl,
ahbsi      => ahbsi,
ahbso      => ahbso(7),
-- Serdes interface
hssli      => hssli,
hsslo      => hsslo
);
```

64 GRIOMMU - AHB/AHB bridge with access protection and address translation

64.1 Overview

The core is used to connect two AMBA AHB buses clocked by synchronous clocks with any frequency ratio. The two buses are connected through an interface pair consisting of an AHB slave and an AHB master interface. AHB transfer forwarding is performed in one direction, where AHB transfers to the slave interface are forwarded to the master interface. The core can be configured to provide access protection and address translation for AMBA accesses traversing over the core. Access protection can be provided using a bit vector to restrict access to memory. Access protection and address translation can also be provided using page tables in main memory, providing full IOMMU functionality. Both protection strategies allow devices to be placed into a configurable number of groups that share data structures located in main memory. The protection and address translation functionality provides protection for memory assigned to processes and operating systems from unwanted accesses by units capable of direct memory access.

Applications of the core include system partitioning, clock domain partitioning, system expansion and secure software partitioning.

Features offered by the core include:

- Single and burst AHB transfer forwarding
- Access protection and address translation that can provide full IOMMU functionality
- Devices can be placed into groups where a group shares page tables / access restriction vectors
- Hardware table-walk
- Efficient bus utilization through (optional) use of SPLIT response, data prefetching and posted writes. **NOTE:** SPLIT responses require an AHB arbiter that allows assertion of HSPLIT during second cycle of SPLIT response. This is supported by GRLIB's AHBCTRL IP core.
- Read and write combining, improves bus utilization and allows connecting cores with differing AMBA access size restrictions.
- Deadlock detection logic enables use of two uni-directional bridges to build a bi-directional bridge. The core can be connected with an another instance of the core, or with a uni-directional AHB/AHB bridge core (AHB2AHB), to form a bi-directional bridge.

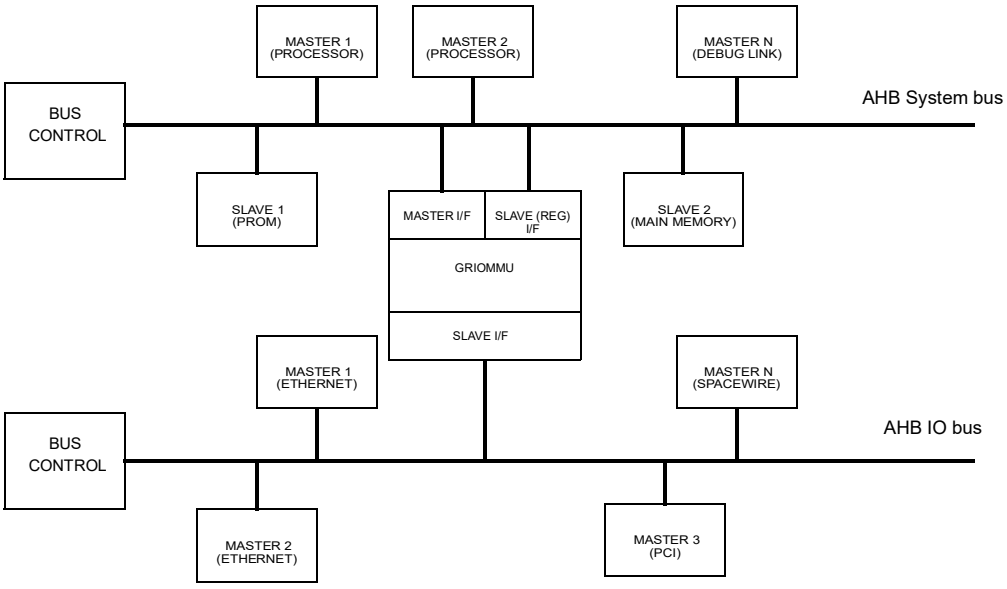


Figure 152. System with core providing access restriction/address translation for masters on AHB IO bus

64.2 Bridge operation

64.2.1 General

The first sub sections below describe the general AHB bridge function. The functionality providing access restriction and address translation is described starting with section 64.3. In the description of AHB accesses below the core propagates accesses from the IO bus to the System bus, see figure 152.

The address space occupied by the core on the IO bus is configurable and determined by Bank Address Registers in the slave interface's AHB Plug&Play configuration record.

The core is capable of handling single and burst transfers of all burst types. Supported transfer sizes (HSIZE) are BYTE, HALF-WORD, WORD, DWORD, 4WORD and 8WORD.

For AHB write transfers write data is always buffered in an internal FIFO implementing posted writes. For AHB read transfers the core uses GRLIB's AMBA Plug&Play information to determine whether the read data will be prefetched and buffered in an internal FIFO. If the target address for an AHB read burst transfer is a prefetchable location the read data will be prefetched and buffered.

The core can be implemented to use SPLIT responses or to insert wait states when handling an access. With SPLIT responses enabled, an AHB master initiating a read transfer to the core is always splitted on the first transfer attempt to allow other masters to use the slave bus while the core performs read transfer on the master bus. The descriptions of operation in the sections below assume that the core has been implemented with support for AMBA SPLIT responses. The effects of disabling support for AMBA SPLIT responses are described in section 64.2.11.

If interrupt forwarding is enabled the interrupts on the IO bus interrupt lines will be forwarded to the system bus and vice versa.

64.2.2 AHB read transfers

When a read transfer is registered on the slave interface connected to the IO bus, the core gives a SPLIT response. The master that initiated the transfer will be de-granted allowing other bus masters to use the slave bus while the core performs a read transfer on the master side. The master interface then requests the bus and starts the read transfer on the master side. Single transfers on the slave side are normally translated to single transfers with the same AHB address and control signals on the master side, however read combining can translate one access into several smaller accesses. Translation of burst transfers from the slave to the master side depends on the burst type, burst length, access size and core configuration.

If the read FIFO is enabled and the transfer is a burst transfer to a prefetchable location, the master interface will prefetch data in the internal read FIFO. If the splitted burst on the slave side was an incremental burst of unspecified length (INCR), the length of the burst is unknown. In this case the master interface performs an incremental burst up to a specified address boundary (determined by the VHDL generic *rburst).* The core can be configured to recognize an INCR read burst marked as instruction fetch (indicated on HPROT signal). In this case the prefetching on the master side is completed at the end of a cache line (the cache line size is configurable through the VHDL generic *iburst*). When the burst transfer is completed on the master side, the splitted master that initiated the transfer (on the slave side) is allowed in bus arbitration by asserting the appropriate HSPLIT signal to the AHB controller. The splitted master re-attempts the transfer and the core will return data with zero wait states.

If the read FIFO is disabled, or the burst is to non-prefetchable area, the burst transfer on the master side is performed using sequence of NONSEQ, BUSY and SEQ transfers. The first access in the burst on the master side is of NONSEQ type. Since the master interface can not decide whether the splitted burst will continue on the slave side or not, the system bus is held by performing BUSY transfers. On the slave side the splitted master that initiated the transfer is allowed in bus arbitration by asserting the HSPLIT signal to the AHB controller. The first access in the transfer is completed by returning read data. The next access in the transfer on the slave side is extended by asserting HREADY low. On the

master side the next access is started by performing a SEQ transfer (and then holding the bus using BUSY transfers). This sequence is repeated until the transfer is ended on the slave side.

In case of an ERROR response on the master side the ERROR response will be given for the same access (address) on the slave side. SPLIT and RETRY responses on the master side are re-attempted until an OKAY or ERROR response is received.

64.2.3 AHB write transfers

The core implements posted writes. During the AHB write transfer on the slave side the data is buffered in the internal write FIFO and the transfer is completed on the slave side by always giving an OKAY response. The master interface requests the bus and performs the write transfer when the master bus is granted. If the burst transfer crosses the write burst boundary (defined by VHDL generic *wburst*), a SPLIT response is given. When the core has written the contents of the FIFO out on the master side, the core will allow the master on the slave side to perform the remaining accesses of the write burst transfer.

Writes are accepted with zero wait states if the core is idle and the incoming access is not locked. If the incoming access is locked, each access will have one wait state. If write combining is disabled a non-locked BUSY cycle will lead to a flush of the write FIFO. If write combining is enabled or if the incoming access is locked, the core will not flush the write FIFO during the BUSY cycle.

64.2.4 Deadlock conditions

When two cores are used to form a bi-directional bridge, a deadlock situation can occur if the cores are simultaneously accessed from both buses. The core that has been configured as a slave contains deadlock detection logic which will resolve a deadlock condition by giving a RETRY response, or by issuing SPLIT complete followed by a new SPLIT response. When the core resolves a deadlock while prefetching data, any data in the prefetch buffer will be dropped when the core's slave interface issues the AMBA RETRY response. When the access is retried it may lead to the same memory locations being read twice.

Deadlock detection logic for bi-directional configurations may lead to deadlocks in other parts of the system. Consider the case where a processor on bus A on one side of the bidirectional bridge needs to perform an instruction fetch over the bridge before it can release a semaphore located in memory on bus A. Another processor on bus B, on the other side of the bridge, may spin on the semaphore waiting for its release. In this scenario, the accesses from the processor on bus B could, depending on system configuration, continuously trigger a deadlock condition where the core will drop data in, or be prevented from initiating, the instruction fetch for the processor on bus A. Due to scenarios of this kind the bridge should not be used in bi-directional configurations where dependencies as the one described above exist between the buses connected by the bridge.

Other deadlock conditions exist with locked transfers, see section 64.2.5.

64.2.5 Locked transfers

The core supports locked transfers. The master bus will be locked when the bus is granted and remain locked until the transfer completes on the slave side. Locked transfers can lead to deadlock conditions, the core's VHDL generic *lckdac* determines if and how the deadlock conditions are resolved.

With the VHDL generic *lckdac* set to 0, locked transfers may *not* be made after another read access which received SPLIT until the first read access has received split complete. This is because the core will return split complete for the first access first and wait for the first master to return. This will cause deadlock since the arbiter is not allowed to change master until a locked transfer has been completed. The AMBA specification requires that the locked transfer is handled before the previous transfer, which received a SPLIT response, is completed.

With *lckdac* set to 1, the core will respond with an AMBA ERROR response to locked access that is made while an ongoing read access has received a SPLIT response. With *lckdac* set to 2 the core will

save state for the read access that received a SPLIT response, allow the locked access to complete, and then complete the first access. All non-locked accesses from other masters will receive SPLIT responses until the saved data has been read out.

If the core is used to create a bi-directional bridge there is one more deadlock condition that may arise when locked accesses are made simultaneously in both directions. If the VHDL generic *lckdac* is set to 0 the core will deadlock. If *lckdac* is set to a non-zero value the slave bridge will resolve the deadlock condition by issuing an AMBA ERROR response to the incoming locked access.

64.2.6 Read and write combining

Read and write combining allows the core to assemble or split AMBA accesses on the core's slave interface into one or several accesses on the master interface. This functionality can improve bus utilization and also allows cores that have differing AMBA access size restrictions to communicate with each other. The functionality attained by read and write combining depends on the VHDL generics *rdcomb* (defines type of read combining), *wrcomb* (defines type of write combining), *slvmstaccsz* (defines maximum AHB access size supported by the core's slave interface) and *mstmaccsz* (defines maximum AHB access size that can be used by core's master interface). These VHDL generics are described in section 64.12.4. The table below shows the effect of different settings. BYTE and HALF-WORD accesses are special cases. The table does not list illegal combinations, for instance *mstmaccsz* /= *slvmaccsz* requires that *wrcomb* /= 0 and *rdcomb* /= 0.

Table 909. Read and write combining

Access on slave interface	Access size	wrcomb	rdcomb	Resulting access(es) on master interface
BYTE or HALF-WORD single read access to any area	-	-	-	Single access of same size
BYTE or HALF-WORD read burst to prefetchable area	-	-	-	Incremental read burst of same access size as on slave interface, the length is the same as the number of 32-bit words in the read buffer, but will not cross the read burst boundary.
BYTE or HALF-WORD read burst to non-prefetchable area	-	-	-	Incremental read burst of same access size as on slave interface, the length is the same as the length of the incoming burst. The master interface will insert BUSY cycles between the sequential accesses.
BYTE or HALF-WORD single write	-	-	-	Single access of same size
BYTE or HALF-WORD write burst	-	-	-	Incremental write burst of same size and length, the maximum length is the number of 32-bit words in the write FIFO.
Single read access to any area	Access size <= mstmaccsz	-	-	Single access of same size
Single read access to any area	Access size > mstmaccsz	-	1	Sequence of single accesses of mstmaccsz. Number of accesses: (access size)/mstmaccsz
Single read access to any area	Access size > mstmaccsz	-	2	Burst of accesses of size mstmaccsz. Length of burst: (access size)/mstmaccsz
Read burst to prefetchable area	-	-	0	Burst of accesses of incoming access size up to address boundary defined by rburst.
Read burst to prefetchable area	-	-	1 or 2	Burst of accesses of size mstmaccsz up to address boundary defined by rburst.
Read burst to non-prefetchable area	Access size <= mstmaccsz	-	-	Incremental read burst of same access size as on slave interface, the length is the same as the length of the incoming burst. The master interface will insert BUSY cycles between the sequential accesses.

Table 909. Read and write combining

Access on slave interface	Access size	wrcomb	rdcomb	Resulting access(es) on master interface
Read burst to non-prefetch-able area	Access size > mstmaccsz	-	1 or 2	Burst of accesses of size mstmaccsz. Length of burst: (incoming burst length)*(access size)/mstmaccsz
Single write	Access size <= mstmaccsz	-	-	Single write access of same size
Single write	Access size > mstmaccsz	1	-	Sequence of single access of mstmaccsz. Number of accesses: (access size)/mstmaccsz.
Single write	Access size > mstmaccsz	2	-	Burst of accesses of mstmaccsz. Length of burst: (access size)/mstmaccsz.
Write burst	-	0	-	Burst of same size as incoming burst, up to address boundary defined by VHDL generic wburst.
Write burst	-	1 or 2	-	Burst write of maximum possible size. The core will use the maximum size (up to mstmaccsz) that it can use to empty the write buffer.

Read and write combining prevents the bridge from propagating fixed length bursts and wrapping bursts. See section 64.2.7 for a discussion on burst operation.

Read and write combining with VHDL generics *wrcomb/rdcomb* set to 1 cause the core to use single accesses when dividing an incoming access into several smaller accesses. This means that another master on the bus may write or read parts of the memory area to be accessed by the core before the core has read or written all the data. In bi-directional configurations, an incoming access on the master core may cause a collision that aborts the operation on the slave core. This may cause the core to read the same memory locations twice. This is normally not a problem when accessing memory areas. The same issues apply when using an AHB arbiter that performs early burst termination. The standard GRLIB AHBCTRL core does not perform early burst termination.

To ensure that the core does not re-read an address, and that all data in an access from the core's slave interface is propagated out on the master interface without interruption the VHDL generics *rdcomb* and *wrcomb* should both be set to 0 or 2. In addition to this, the AHB arbiter may not perform early burst termination (early burst termination is not performed by the GRLIB AHBCTRL arbiter).

Read and write combining can be limited to specified address ranges. See description of the *combmask* VHDL generic for more information. Note that if the core is implemented with support for prefetch and read combining, it will not obey combmask for prefetch operations (burst read to prefetchable areas). Prefetch operations will always be performed with the maximum allowed size on the master interface.

64.2.7 Burst operation

The core can be configured to support all AMBA 2.0 burst types (single access, incrementing burst of unspecified length, fixed length incrementing bursts and wrapping bursts). Single accesses and incrementing bursts of unspecified length have previously been discussed in this document. An incoming single access will lead to one access, or multiple accesses for some cases with read/write combining, on the other side of the bridge. An incoming incrementing burst of unspecified length to a prefetchable area will lead to the prefetch buffer (if available) being filled using the same access size, or the maximum allowed access size if read/write combining is enabled, on the master interface.

If the core is used in a system where no fixed length bursts or incremental bursts will be used in accesses to the bridge, then set the *allbrst* generic to 0 and skip the remainder of this section.

The VHDL generic *allbrst* controls if the core will support fixed length and wrapping burst accesses. If *allbrst* is set to 0, the core will treat all burst accesses as incrementing of unspecified length. For fixed length and wrapping bursts this can lead to performance penalties and malfunctions. Support for

fixed length and wrapping bursts is enabled by setting *allbrst* to 1 or 2. Table 64.2.7 describes how the core will handle different burst types depending on the setting of *allbrst*.

Table 910. Burst handling

Value of allbrst generic	**Access type***	**Undefined length incrementing burst INCR**	**Fixed length incrementing burst INCR{4,8,16}**	**Wrapping burst WRAP{4,8,16}**
0	Reads to non-prefetchable area	Incrementing burst with BUSY cycles inserted. Same behaviour with read and write combining.	Fixed length burst with BUSY cycles inserted. If the burst is short then the burst may end with a BUSY cycle. If access combining is used the HBURST signal will get incorrect values.	Malfunction. Not supported
	Reads to prefetchable area	Incrementing burst of maximum allowed size, filling prefetch buffer, starting at address boundary defined by prefetch buffer.		Malfunction. Not supported
	Write burst	Incrementing burst	Incrementing burst, if write combining is enabled, and triggered, the burst will be translated to an incrementing burst of undefined length. VHDL generic *wrcomb* should not be set to 1 (but to 0 or 2) in this case	Write combining is not supported. Same access size will be used on both sides of the bridge.
1	Reads to non-prefetchable area	Incrementing burst with BUSY cycles inserted. Same behaviour with read and write combining.	Same burst type with BUSY cycles inserted. If read combining is enabled, and triggered by the incoming access size, an incremental burst of unspecified length will be used. If the burst is short then the burst may end with a BUSY cycle.	Same burst type with BUSY cycles inserted. If read combining is enabled, and triggered by the incoming access size, an incremental burst of unspecified length will be used. This will cause AMBA violations if the wrapping burst does not start from offset 0.
	Reads to prefetchable area	Incrementing burst of maximum allowed size, filling prefetch buffer.	For reads, the core will perform full (or part that fits in prefetch buffer) fixed/wrapping burst on master interface and then respond with data. No BUSY cycles are inserted. If the access made to the slave interface is larger than the maximum supported access size on the master interface then a incrementing burst of unspecified length will be used to fill the prefetch buffer. This (read combining) is not supported for wrapping bursts.	
	Write burst	Same as for allbrst = 0		
2	Reads to non-prefetchable area	Incrementing burst with BUSY cycles inserted. Same behaviour with read and write combining.	Reads are treated as a prefetchable burst. See below.	
	Reads to prefetchable area	Incrementing burst of maximum allowed size, filling prefetch buffer, starting at address boundary defined by prefetch buffer.	Core will perform full (or part that fits in prefetch buffer) fixed/wrapping burst on master interface and then respond with data. No BUSY cycles are inserted. If the access made to the slave interface is larger than the maximum supported access size on the master interface then a incrementing burst of unspecified length will be used to fill the prefetch buffer. This (read combining) is not supported for wrapping bursts.	
	Write burst	Same as for allbrst = 0		

* Access to prefetchable area where the core's prefetch buffer is ised (VHDL generic pfen /= 0).

64.2.8 Transaction ordering, starvation and AMBA arbitration schemes

The core is configured at implementation to use one of two available schemes to handle incoming accesses. The core will issue SPLIT responses when it is busy and on incoming read accesses. If the core has been configured to use first-come, first-served ordering it will keep track of the order of incoming accesses and serve the requests in the same order. If first-come, first-served ordering is disabled the core will give some advantage to the master it has a response for and then allow all masters in to arbitration simultaneously, moving the decision on which master that should be allowed to access the core to the bus arbitration.

When designing a system containing a core the expected traffic patterns should be analyzed. The designer must be aware how SPLIT responses affect arbitration and how the selected transaction ordering in the core will affect the system. The two different schemes are further described in sections 64.2.9 and 64.2.10.

64.2.9 First-come, first-served ordering

First-come, first served ordering is used when the VHDL generic *fcfs* is non-zero.

With first-come, first-served ordering the core will keep track of the order of incoming accesses. The accesses will then be served in the same order. For instance, if master 0 initiates an access to the core, followed by master 3 and then master 5, the core will propagate the access from master 0 (and respond with SPLIT on a read access) and then respond with SPLIT to the other masters. When the core has a response for master 0, this master will be allowed in arbitration again by the core asserting HSPLIT. When the core has finished serving master 0 it will allow the next queued master in arbitration, in this case master 3. Other incoming masters will receive SPLIT responses and will not be allowed in arbitration until all previous masters have been served.

An incoming locked access will always be given precedence over any other masters in the queue.

A burst that has initiated a pre-fetch operation will receive SPLIT and be inserted last in the master queue if the burst is longer than the maximum burst length that the core has been configured for.

It should be noted that first-come, first-served ordering may not work well in systems where an AHB master needs to have higher priority compared to the other masters. The core will not prioritize any master, except for masters performing locked accesses.

64.2.10 Bus arbiter ordering

Bus arbiter ordering is used when VHDL generic *fcfs* is set to zero.

When several masters have received SPLIT and the core has a response for one of these masters, the master with the queued response will be allowed in to bus arbitration by the core asserting the corresponding HSPLIT signal. In the following clock cycle, all other masters that have received SPLIT responses will also be allowed in bus arbitration as the core asserts their HSPLIT signals simultaneously. By doing this the core defers the decision on the master to be granted next to the AHB arbiter. The core does not show any preference based on the order in which it issued SPLIT responses to masters, except to the master that initially started a read or write operation. Care has been taken so that the core shows a consistent behavior when issuing SPLIT responses. For instance, the core could be simplified if it could issue a SPLIT response just to be able to change state, and not initiate a new operation, to an access coming after an access that read out prefetched data. When the core entered its idle state it could then allow all masters in bus arbitration and resume normal operation. That solution could lead to starvation issues such as:

T0: Master 1 and Master 2 have received SPLIT responses, the core is prefetching data for Master 1

T1: Master 1 is allowed in bus arbitration by setting the corresponding HSPLIT

T2: Master 1 reads out prefetch data, Master 2 HSPLIT is asserted to let Master 2 in to bus arbitration

T3: Master 2 performs an access, receives SPLIT, however the core does not initiate an access, it just stalls in order to enter its idle state.

T4: Master 2 is allowed in to bus arbitration, Master 1 initiates an access that leads to a prefetch and Master 1 receives a SPLIT response

T5: Master 2 performs an access, receives SPLIT since the core is prefetching data for master 1

T6: Go back to T0

This pattern will repeat until Master 1 backs away from the bus and Master 2 is able to make an access that starts an operation over the core. In most systems it is unlikely that this behavior would introduce a bus lock. However, the case above could lead to an unexpectedly long time for Master 2 to complete its access. Please note that the example above is illustrative and the problem does not exist in the core as the core does not issue SPLIT responses to (non-locked) accesses in order to just change state but a similar pattern could appear as a result of decisions taken by the AHB arbiter if Master 1 is given higher priority than Master 2.

In the case of write operations the scenario is slightly different. The core will accept a write immediately and will not issue a SPLIT response. While the core is busy performing the write on the master side it will issue SPLIT responses to all incoming accesses. When the core has completed the write operation on the master side it will continue to issue SPLIT responses to any incoming access until there is a cycle where the core does not receive an access. In this cycle the core will assert HSPLIT for all masters that have received a SPLIT response and return to its idle state. The first master to access the core in the idle state will be able to start a new operation. This can lead to the following behavior:

T0: Master 1 performs a write operation, does NOT receive a SPLIT response

T1: Master 2 accesses the core and receives a SPLIT response

T2: The core now switches state to idle since the write completed and asserts HSPLIT for Master 2.

T3: Master 1 is before Master 2 in the arbitration order and we are back at T0.

In order to avoid this last pattern the core would have to keep track of the order in which it has issued SPLIT responses and then assert HSPLIT in the same order. This is done with first-come, first-served ordering described in section 64.2.9.

64.2.11 AMBA SPLIT support

Support for AMBA SPLIT responses is enabled/disabled through the VHDL generic *split*. SPLIT support should be enabled for most systems. The benefits of using SPLIT responses is that the bus on the core's slave interface side can be free while the core is performing an operation on the master side. This will allow other masters to access the bus and generally improve system performance. The use of SPLIT responses also allows First-come, first-served transaction ordering.

For configurations where the core is the only slave interface on a bus, it can be beneficial to implement the core without support for AMBA SPLIT responses. Removing support for SPLIT responses reduces the area used by the core and may also reduce the time required to perform accesses that traverse the core. It should be noted that building a bi-directional core without support for SPLIT responses will increase the risk of access collisions.

If SPLIT support is disabled the core will insert wait states where it would otherwise issue a SPLIT response. This means that the arbitration ordering will be left to the bus arbiter and the core cannot be implemented with the First-come, first-served transaction ordering scheme. The core will still issue RETRY responses to resolve dead lock conditions, to split up long burst and also when the core is busy emptying it's write buffer on the master side.

The core may also be implemented with dynamic SPLIT support, this allows the use of SPLIT responses to be configurable via the core's register interface (see SP field in the core's Control register).

64.2.12 Core latency

The delay incurred when performing an access over the core depends on several parameters such as core configuration, the operating frequency of the AMBA buses, AMBA bus widths and memory access patterns. This section deals with latencies in the core's bridge function. Access protection mechanisms may add additional delays, please refer to the description of access protection for a description of any additional delays.

Table 911 below shows core behavior in a system where both AMBA buses are running at the same frequency and the core has been configured to use AMBA SPLIT responses. Table 912 further down shows core behavior in the same system without support for SPLIT responses.

*Table 911.*Example of single read with FFACT = 1, and SPLIT support

Clock cycle	Core slave side activity	Core master side activity
0	Discovers access and transitions from idle state	Idle
1	Slave side waits for master side, SPLIT response is given to incoming access, any new incoming accesses also receive SPLIT responses.	Discovers slave side transition. Master interface output signals are assigned.
2		If bus access is granted, perform address phase. Otherwise wait for bus grant.
3		Register read data and transition to data ready state.
4	Discovers that read data is ready, assign read data output and assign SPLIT complete	Idle
5	SPLIT complete output is HIGH	
6	Typically a wait cycle for the SPLIT:ed master to be allowed into arbitration. Core waits for master to return. Other masters receive SPLIT responses.	
7	Master has been allowed into arbitration and performs address phase. Core keeps HREADY high	
8	Access data phase. Core has returned to idle state.	

*Table 912.*Example of single read with FFACT = 1, without SPLIT support

Clock cycle	Core slave side activity	Core master side activity
0	Discovers access and transitions from idle state	Idle
1	Slave side waits for master side, wait states are inserted on the AMBA bus.	Discovers slave side transition. Master interface output signals are assigned.
2		Bus access is granted, perform address phase.
3		Register read data and transition to data ready state.
4	Discovers that read data is ready, assign HREADY output register and data output register.	Idle
5	HREADY is driven on AMBA bus. Core has returned to idle state	

While the transitions shown in tables 911 and 912 are simplified they give an accurate view of the core delay. If the master interface needs to wait for a bus grant or if the read operation receives wait states, these cycles must be added to the cycle count in the tables. The behavior of the core with a fre-

quency factor of two between the buses is shown in tables 913 and 914 (best case, delay may be larger depending on which slave clock cycle an access is made to the core).

*Table 913.*Example of single read with FFACT = 2, Master freq. > Slave freq, without SPLIT support

Slave side clock cycle	Core slave side activity	Master side clock cycle	Core master side activity
0	Discovers access and transitions from idle state	0	Discovers slave side transition. Master interface output signals are assigned.
1	Slave side waits for master side, wait states are inserted on the AMBA bus.		
2		1	Bus access is granted, perform address phase.
3			
4		2	Register read data and transition to data ready state.
5			
6	Discovers that read data is ready, assign HREADY output register and data output register.	3	Idle
7	HREADY is driven on AMBA bus. Core has returned to idle state		

*Table 914.*Example of single read with FFACT = 2, Master freq. > Slave freq, without SPLIT support

Slave side clock cycle	Core slave side activity	Master side clock cycle	Core master side activity
0	Discovers access and transitions from idle state	0	Idle
		1	
1	Slave side waits for master side, wait states are inserted on the AMBA bus.	2	Discovers slave side transition. Master interface output signals are assigned.
		3	Bus access is granted, perform address phase.
2	Discovers that read data is ready, assign HREADY output register and data output register.	4	Register read data and transition to data ready state.
		5	Idle
3	HREADY is driven on AMBA bus. Core has returned to idle state	6	
		7	

Table 915 below lists the delays incurred for single operations that traverse the bridge while the bridge is in its idle state. The second column shows the number of cycles it takes the master side to perform the requested access, this column assumes that the master slave gets access to the bus immediately and that each access is completed with zero wait states. The table only includes the delay incurred by traversing the core. For instance, when the access initiating master reads the core's prefetch buffer, each additional read will consume one clock cycle. However, this delay would also have been present if the master accessed any other slave.

Write accesses are accepted with zero wait states if the bridge is idle, this means that performing a write to the idle core does not incur any extra latency. However, the core must complete the write operation on the master side before it can handle a new access on the slave side. If the core has not transitioned into its idle state, pending the completion of an earlier access, the delay suffered by an access be longer than what is shown in the tables in this section. Accesses may also suffer increased delays during collisions when the core has been instantiated to form a bi-directional bridge. Locked accesses that abort on-going read operations will also mean additional delays.

If the core has been implemented to use AMBA SPLIT responses there will be an additional delay where, typically, one cycle is required for the arbiter to react to the assertion of HSPLIT and one clock cycle for the repetition of the address phase.

Note that if the core has support for read and/or write combining, the number of cycles required for the master will change depending on the access size and length of the incoming burst access. For instance, in a system where the bus in the core's master side is wider than the bus on the slave side, write combining will allow the core to accept writes with zero wait states and then combine several accesses into one or several larger access. Depending on memory controller implementation this could reduce the time required to move data to external memory, and will reduce the load on the master side bus.

Table 915. Access latencies

Access	Master acc. cycles	Slave cycles	Delay incurred by performing access over core
Single read	3	1	1 * clk_{slv} + 3 * clk_{mst}
Burst read with prefetch	2 + (burst length)[x]	2	2 * clk_{slv} + (2 + burst length)* clk_{mst}
Single write[xx]	(2)	0	0
Burst write[xx]	(2 + (burst length))	0	0

[x] A prefetch operation ends at the address boundary defined by the prefetch buffer's size

[xx] The core implements posted writes, the number of cycles taken by the master side can only affect the next access.

64.2.13 Endianness

The core is designed for big-endian systems.

64.3 General access protection and address translation

64.3.1 Overview

The core provides two types of access protection. The first option is to use a bit vector to implement access restriction on a memory page basis. The second option is to use a page-table to provide access restriction and address translation. Regardless of the protection strategy, the core provides means to assign masters on the IO bus in groups where each group can be associated with a data structure (access restriction vector or page table) in memory. The core can be implemented to support a dynamically configurable page size from 4 to 512 KiB, or a fixed page size of 4 KiB.

When a master on the IO bus initiates an access to be propagated by the core, the core will first look at the incoming master's group assignment setting to determine to which group the master belongs. When the group is known, the core can propagate or inhibit the access based on the group's attributes, or determine the address of the in-memory data structures to use for access checks (and possibly address translation). The in-memory data structure may be cached by the core, otherwise the information will be fetched from main memory.

Once the core has the necessary information to process the incoming access, the access will be either allowed to propagate through the core or, in case the access is to a restricted memory location, be inhibited. If the access is inhibited, the core will issue an AMBA ERROR response to the master if the incoming access is a read access. The core implements posted writes, therefore write operations will not receive an AMBA ERROR response. An interrupt can, optionally, be asserted when an access is inhibited. The AHB failing access register can be configured to log the first or most recent access that was inhibited.

When enabled, the core always checks access permissions when a master initiates an access. For the access protection and translation operation to be effective the masters are required to adhere to the AMBA 2.0 specification and not issue burst transfers that cross a 1 KiB address boundary.

It is possible for masters to access the core's register interface through the core. In this case the core will perform an access to itself on the System AHB bus. For configurations where the core is used to form a bi-directional core, any data structures read by the core must be located on the same bus as the core's master interface. The core cannot access data structures that are placed on the same bus as masters that the core protects against, in other words data structures must be accessible on a slave on the System bus, see figure 152.

64.3.2 Delays incurred from access protection

The time required for the core's master interface to start an access may be delayed by access protection checks. Table 916 below shows the added delays, please refer to section 64.2.12 for a description of delays from the core's bridge operation.

Table 916. Access protection check latencies

Protection mode	Delay in clock cycles on master side
Disabled	0
Write-protection only and read access	0
Master assigned to group in passthrough or inactive group	1
Access Protection Vector, cache hit	1
Access Protection Vector cache miss, cache disabled/not implemented	Minimum[x] 4 clock cycles
IOMMU Protection, cache hit	1
IOMMU Protection, TLB miss, TLB disabled/not implemented	Minimum[x] 4 clock cycles

[x] The core may suffer additional AMBA bus delays when accessing the vector in memory. 4 cycles is the minimum time required and assumes that the core is instantly granted access to the bus and that data is delivered with zero wait states.

64.4 Access Protection Vector

The Access Protection Vector (APV) consists of a continuous bit vector where each bit determines the access rights to a memory page. The bit vector provides access restriction on the full 4 GiB AMBA address space. The required size of the bit vector depends on the page size used by the core, see table below:

Table 917. Bit vector size vs. page size

Page size	Bit vector size
4 KiB	128 KiB
8 KiB	64 KiB
16 KiB	32 KiB
32 KiB	16 KiB
64 KiB	8 KiB
128 KiB	4 KiB
256 KiB	2 KiB
512 KiB	1 KiB

Each group can have a bit vector with a base address specified by a field in the group's Group Control Register. When a master performs an access to the core, the master's group number is used to select one of the available bit vectors. The AMBA access size used to fetch the vector is fixed at implementation time and can be read out from the core's Capability register 1. If the AMBA access size to use is 32-bits (WORD sized) and the page size is 4 KiB, bits 31:17 of the incoming address (HADDR) are used to index a word in the bit vector, and bits HADDR[16:12] are used to select one of the 32 bits in the word. For each increase in AMBA access size (DWORD, 4WORD, 8WORD), one bit less of the

physical address is used to index the vector and this bit is instead used to select one specific bit in the data read from memory. Similarly, for each increase in page size one bit less of the physical address is used.

The lowest page is protected by the most significant bit in the bit vector. This means that page 0 is protected by the most significant bit in byte 0 read from the bit vector's base address (using big endian addressing). When performing WORD accesses, the lowest page is protected by bit 31 in the accessed word (using the bit numbering convention used throughout this document).

If the bit at the selected position is '0', the access to the page is allowed and the core will propagate the access. If the selected bit is '1', and the access is an read access, an AMBA ERROR response is given to the master initiating the access. If the selected bit is '1', and the access is a write access, the write is inhibited (not propagated through the core).

64.4.1 Access Protection Vector cache

The core can be implemented with an internal memory that caches the Access Protection Vector. The cache size is configurable at implementation time and depends on a number of parameters that can be read out via Capability registers 0 and 1. If the core has been implemented with IOMMU functionality and a IOMMU Translation Lookaside Buffer (TLB), the RAMs in the APV cache will be shared with the IOMMU TLB.

The cache is implemented as a direct-mapped cache built up of one data RAM and one tag RAM. The number of locations in each RAM is the number of lines in the cache. The width of the data RAM (cache line size) is the same as the size of the AMBA accesses used to fetch the APV from main memory. The width and contents of the tag RAM depends on the number of supported groups, cache line size and number of lines in the cache.

The address used to select a position in the RAMs, called the set address, must have *log2(number of lines in the cache)* bits. The number of address bits taken from the physical address required to uniquely address one position in the bit vector depends on the cache line size. The number of required bits for each allowed cache line size is shown in table 918 below.

*Table 918.*Cache line size vs. physical address bits

Cache line size in bits	Bits of physical address needed to identify one position depending on page size							
	4 KiB	8 KiB	16 KiB	32 KiB	64 KiB	128 KiB	256 KiB	512 KiB
32	15	14	13	12	11	10	9	8
64	14	13	12	11	10	9	8	7
128	13	12	11	10	9	8	7	6
256	12	11	10	9	8	7	6	5

If the core has support for more than one group, the cache must also be tagged with the group ID. The number of bits required to uniquely select one group is *log2(number of groups)*.

This means that in order to be able to cache the full bit vectors for all supported groups the cache address (set address) must have *log2(number of groups) + (required physical address bits)* address bits. The number of required lines in the cache to be able to hold all vectors is:

*cache lines = (number of groups) * (2^{20} / (cache line size))*

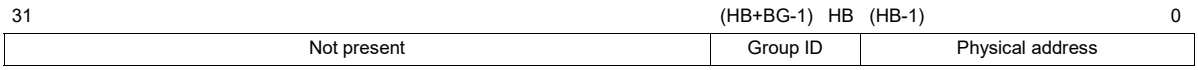
If the cache size is not large enough to hold a copy of each position in the bit vector, part of the physical address and group will be placed in the cache tag RAM instead. If the number of lines in the cache allows keeping a cached data of all positions in all bit vectors, the set address and tag data arrangement shown in table 919 will be used.

For the set address/tag RAM tables below the following values are used:

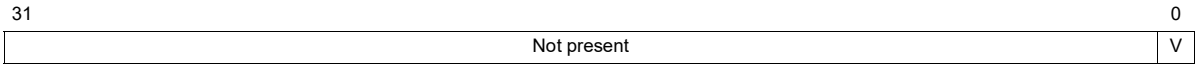
SB = Set address bits = log2(cache line size)
HB = Required number of bits of physical address = See table 918 above.
GB = Required number of bits to select one group = *log2(number of groups)*

Table 919. Set address bits = (group ID bits) + (Physical address bits)

Set address:



Contents of Tag RAM:

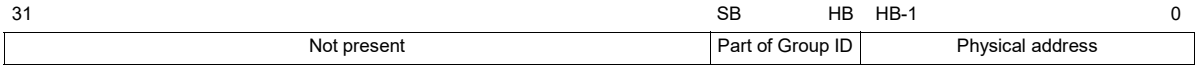


0 Valid (V) - Signals that addressed position in cache contains valid data

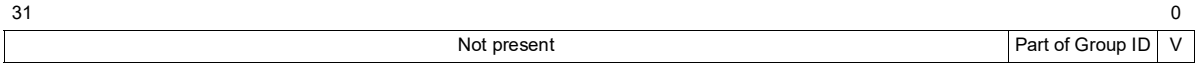
If the number of lines in the cache allows part of the group ID to be part of the set address, the arrangement will be:

Table 920. Set address bits < (group ID bits) + (Physical address bits)

Set address:



Contents of Tag RAM:

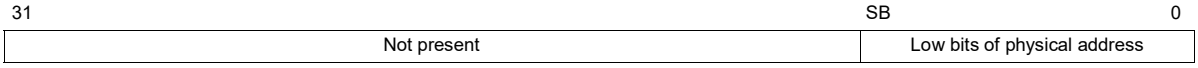


0 Valid (V) - Signals that addressed position in cache contains valid data

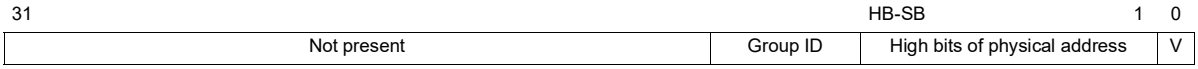
If the number of lines in the cache only allows part of the required physical address to be part of the set address, the arrangement will be:

Table 921. Set address bits < (group ID bits) + (Physical address bits)

Set address:



Contents of Tag RAM:



0 Valid (V) - Signals that addressed position in cache contains valid data

In the first arrangement, where *(set address bits) = (group ID bits) + (physical address bits)*, there will never be a collision in the cache. In the two other arrangements there is not room for all positions in the bit vector(s). This means that a cached copy for one memory page can be replaced with the bit vector for another memory page. Since the physical address is used as the set address, accesses from a master assigned to one group may evict cached bit vector data belonging to another group. This may not be wanted in systems where interference between groups of masters should be minimized. In order to minimize inter-group interference, the core can be implemented with support for using as much of the group ID as possible in the set address, this functionality is called group-set-addressing.

The core has support for group-set-addressing if the CA field in Capability register 0 is non-zero. If the number of set address bits (cache lines) is large enough to cache all bit vectors, the set address and

tag RAM arrangement will be as described by table 919. If the number of set address bits will allow the whole group ID to be part of the set address, the arrangement will be:

Table 922. Group set address: Set address bits < (group ID bits) + (Physical address bits)

Set address:

31	SB	GB-1	0
Not present	Low bits of physical address	Group ID	

Contents of Tag RAM:

31	1	0
Not present	High bits of physical address	V

0 Valid (V) - Signals that addressed position in cache contains valid data

If only part of the group ID can be used for the set address, the arrangement will be:

Table 923. Group set address: Set address bits < (group ID bits)

Set address:

31	GB-SB-1	0
Not present	Low part of Group ID	

Contents of Tag RAM:

31		1	0
Not present	Physical address	High part of group ID	V

0 Valid (V) - Signals that addressed position in cache contains valid data

Group-set-addressing is enabled via the GS field in the core's Control register.

64.4.2 Constraining the memory area covered by the APV cache

In a typical system, the normal case for an AMBA master core is to perform accesses to main memory. In order to reduce latency, the protection data for these accesses is ideally cached within the core. However, main memory is not likely to occupy the full AMBA address range. If accesses outside a certain access range is expected to be rare, and if it is not critical if these accesses suffer a higher latency, it can be beneficial to restrict the memory range for which the core caches the Access Protection Vector. The benefit of this is that the cache size can be reduced while the same hit rate is kept for the specified memory area, alternatively the hit rate could possibly be increased while keeping the cache size constant.

The core can be configured at implementation to only cache the bit vector for a specified memory range. Capability register 1 contains an address and a mask that describes this area. Bit vector data for the specified memory range will be cached by the core. Bit vector data for accesses made outside the memory range will not be placed in the cache, and will instead be fetched for memory on each access. The impact of having a non-zero mask in Capability register 1 is that for each '1' in the mask, one physical address bit can be removed from the cache set address in the examples given earlier in this section.

64.4.3 Access Protection Vector cache flush operation

If the contents of a vector is modified the core cache must be flushed by writing to the TLB/Cache Flush Register. The TLB/Cache Flush register contains fields to flush the entire cache or to flush the lines belonging to a specified group. In order to flush entries for a specific group, group-set-addressing must be implemented and enabled. Performing a group flush without group-set-addressing may only flush part of the cache and can lead to unexpected behavior.

The core will not propagate any transfers while a cache flush operation is in progress.

64.5 IO Memory Management Unit (IOMMU) functionality

The IOMMU functionality of the core provides address translation and access protection on the full 4 GiB AMBA address space. The size of the address range where addresses are translated is specified by the IOMMU Translation Range (ITR) field in the core's Control register:

*Size of translated address range in MiB = 16 MiB * 2^{ITR}*

The maximum allowed value of the ITR field is eight, which means that the IOMMU can provide address translation to an area of size $16*2^8$ = 4096 MiB, which is the full 32-bit address space. When ITR is set to eight and a page size of 4 KiB is used, bits 31:12 of the incoming IO address are translated to physical addresses, using IO Page Tables entries describes below. Bits 11:0 of the incoming access are propagated through the IOMMU. For each increase in page size one more bit will be directly propagated through the IOMMU instead of being translated.

If ITR is less then eight the most significant bits of the IO address must match the value of the TMASK field in Capability register 2. If an access is outside the range specified by TMASK the access will be inhibited. Table 924 shows the effect of different ITR values. As an example, with ITR set to 2, the IOMMU will perform address translation for a range that spans 64 MiB. This range will be located at offset TMASK[31:26]. Accesses to addresses that do not have their most significant bits set to match TMASK[31:26] will be inhibited. The table also shows the number of pages within the decoded range and the memory required to hold the translation information (page tables) in main memory. The *pgsz* value is the value of the PGSZ field in the control register.

*Table 924.*Effects of IOMMU Translation Range setting

ITR	Size of translated range	TMASK bits used	Number of pages	Size of page tables
0	16 MiB	TMASK[31:24]	4096 / 2^{pgsz}	16 / 2^{pgsz} KiB
1	32 MiB	TMASK[31:25]	8192 / 2^{pgsz}	32 / 2^{pgsz} KiB
2	64 MiB	TMASK[31:26]	16384 / 2^{pgsz}	64 / 2^{pgsz}KiB
3	128 MiB	TMASK[31:27]	32768 / 2^{pgsz}	128 / 2^{pgsz} KiB
4	256 MiB	TMASK[31:28]	655536 / 2^{pgsz}	256 / 2^{pgsz} KiB
5	512 MiB	TMASK[31:29]	131072 / 2^{pgsz}	512 / 2^{pgsz} KiB
6	1024 MiB	TMASK[31:30]	262144 / 2^{pgsz}	1 / 2^{pgsz} MiB
7	2048 MiB	TMASK[31]	524288 / 2^{pgsz}	2 / 2^{pgsz} MiB
8	4096 MiB	TMASK not used	1048576 / 2^{pgsz}	4 / 2^{pgsz} MiB

64.5.1 IO Page Table Entry

Address translation is performed by looking up translation information in a one-level table present in main memory. Part of the incoming address is used to index the table that consists of IO Page Table Entries. The format of an IO Page Table Entry (IOPTE) is shown in table 925 below.

*Table 925.*IOMMU Page Table Entry (IOPTE)

31 … 8	7	6 … 5	4	3	2	1	0
PPAGE	C	R	BO	BS	W	V	R

31:8	Physical Page (PPAGE) - Bits 27:8 of this field corresponds to physical address bits 31:12 of the page. With a 4 KiB page size, PPAGE[27:8] is concatenated with the incoming IO address bits [11:0] to form the translated address. For each increase in page size one bit less of PPAGE is used and one bit more of the incoming IO address is used: this means that with a 16 KiB page size , PPAGE[27:10] will be concatenated with the incoming IO address bits [13:0] to form the translated address. Bits 31:27 of this field are currently discarded by the IOMMU and are present in the data structure for forward compatibility with systems using 36-bit AMBA address space.
7	Cacheable (C) - This field is currently not used by the IOMMU
6:5	RESERVED
4	Bus select Override (BO) - If this field is set to ‘1’ then the bus selection is made via the IOPTE.BS field.
3	Bus Select (BS) - Overrides master configuration register BS field when BO field in this PTE is set.
2	Writeable (W) - If this field is ‘1’ write access is allowed to the page. If this field is ‘0’, only read accesses are allowed.
1	Valid (V) - If this field is ‘1’ the PTE is valid. If this field is ‘0’, accesses to the page covered by this PTE will be inhibited.
0	RESERVED

When the core has IOMMU protection enabled all, incoming accesses from masters belonging to an active group, which is not in pass-through mode, will be matched against TMASK. If an access is outside the range specified by ITR/TMASK, the access will be inhibited and may receive an AMBA ERROR response (not applicable when the access is a posted write).

If the incoming access is within the range specified by ITR/TMASK, the core will use the incoming IO address to index the page table containing the address translation information for the master/IO address. The core may be implemented with an Translation Lookaside Buffer (TLB) that may hold a cached copy of the translation information. Otherwise the translation information will be fetched from main memory. The base address of the page table to use is given by the Group Configuration register to which the master performing the access is assigned. Please see the register description of the Group Configuration register for constraints on the page table base address. The core will use bits X:Y to index the table, where X depends on the value of the ITR field in the core’s Control register, and Y depends on the page size (Y = 12 + PGSZ field in Control register).

When the core has fetched the translation information (IOPTE) for the accesses page it will check the IOPTE’s Valid (V) and Writeable (W) fields. If the IOPTE is invalid, the access will be inhibited. If the Writeable (W) field is unset and the access is a write access, the access will be inhibited. Otherwise the core will, for a page size of 4 KiB, use the IOPTE field PPAGE, bits 27:8, and bits 11:0 of the incoming IO address to form the physical address to use when the access is propagated by the core (physical address: PPAGE[27:8] & IOADDR[11:0]).

If the valid (V) bit of the IOPTE is ‘0’ the core may or may not store the IOPTE in the TLB (if implemented). This is controlled via the SIV field in the core’s Control register.

64.5.2 Prefetch operations and IOMMU protection

During normal bridge operation, and with Access Protection Vector protection, the core determines if data for an access can be prefetched by looking at the IO address and the System bus plug and play

information. This operation cannot be done without introducing additional delays when the core is using IOMMU protection. The incoming IO address must first be translated before it can be determined if the access is to a memory area that can be prefetched. In order to minimize delays the core makes the assumption that any incoming burst access is to a prefetchable area. The result is that when using IOMMU protection all burst accesses will result in the core performing a prefetch operation.

64.5.3 Translation Lookaside Buffer operation

The core can be implemented with an internal memory that caches IO Page Table Entries. This memory is referred to as a Translation Lookaside Buffer (TLB). The TLB size is configurable at implementation time and depends on a number of parameters that can be read out via Capability registers 0 and 2. If the core has been implemented with Access Protection Vector functionality and an APV cache, the RAMs for the APV cache will be shared with the IOMMU TLB.

The TLB is implemented as a direct-mapped cache built up of one data RAM and one tag RAM. The number of locations in each RAM is the number of entries in the TLB. The width of the data RAM (entry size) is the same as the size of the AMBA accesses used to fetch page table entries from main memory. The width and contents of the tag RAM depends on the number of supported groups, entry size and number of entries in the TLB.

The address used to select a position in the RAMs, called the set address, must have *log2(number of entries in the TLB)* bits. The number of address bits taken from the physical address required to uniquely address one position in the TLB depends on the entry size. The number of required bits for each allowed entry size is shown in table 918 below, the values in the third column is the number of address bits that must be used to accommodate the largest translatable range (maximum value of ITR field in the core's Control register). Note that an entry size larger than 32 bits results in an TLB that holds multiple IOPTEs per entry.

Table 926. TLB entry size vs. physical address bits

Entry size in bits	Entry size in IOPTEs	Bits of physical address needed to identify one position depending on page size							
		4 KiB	8 KiB	16 KiB	32 KiB	64 KiB	128 KiB	256 KiB	512 KiB
32	1	20	19	18	17	16	15	14	13
64	2	19	18	17	16	15	14	13	12
128	4	18	17	16	15	14	13	12	11
256	8	17	16	15	14	13	12	11	10

If the core has support for more than one group, the TLB entries must also be tagged with the group ID. The number of bits required to uniquely select one group is *log2(number of groups)*.

This means that in order to be able to cache the page tables for all supported groups the TLB address (set address) must have *log2(number of groups) + (required physical address bits)* address bits. The number of required entries in the TLB to be able to hold all vectors is:

*TLB entries = (number of groups) * (2^{20} / (entry size))*

If the TLB is not large enough to hold a copy of each position in the page table, part of the physical address and group will be placed in the tag RAM. The core will implement the TLB depending on the parameters mentioned above. If the number of entries in the TLB allows keeping a copy of all positions in all page tables, the set address and tag data arrangement shown in table 927 will be used.

For the set address/tag RAM tables below the following values are used:

SB = Set address bits = log2(number of TLB entries)
HB = Required number of bits of physical address = See table 926 above.
GB = Required number of bits to select one group = *log2(number of groups)*

*Table 927.*Set address bits = (group ID bits) + (Physical address bits)
Set address:

31	(HB+BG-1) HB	(HB-1) 0
Not present	Group ID	Physical address

Contents of Tag RAM:

31	0
Not present	V

0 Valid (V) - Signals that addressed position in cache contains valid data

If the number of entries in the TLB allows part of the group ID to be part of the set address, the arrangement will be:

*Table 928.*Set address bits < (group ID bits) + (Physical address bits)
Set address:

31	SB HB	HB-1 0
Not present	Part of Group ID	Physical address

Contents of Tag RAM:

31	0	
Not present	Part of Group ID	V

0 Valid (V) - Signals that addressed position in cache contains valid data

If the number of entries in the TLB only allows part of the required physical address to be part of the set address, the arrangement will be:

*Table 929.*Set address bits < (group ID bits) + (Physical address bits)
Set address:

31	SB 0
Not present	Low bits of physical address

Contents of Tag RAM:

31		HB-SB 1	0
Not present	Group ID	High bits of physical address	V

0 Valid (V) - Signals that addressed position in cache contains valid data

In the first arrangement, where *(set address bits) = (group ID bits) + (physical address bits)*, there will never be a collision in the TLB. In the two other arrangements there is not room for all entries in the page table(s). This means that a cached IOPTE for one memory page can be replaced with the IOPTE for another memory page. Since the physical address is used as the set address, accesses from a master assigned to one group may evict cached IOPTE's belonging to another group. This may not be wanted in systems where interference between groups of masters should be minimized. In order to minimize inter-group interference, the core can be implemented with support for using as much of the group ID as possible in the set address, this functionality is called group-set-addressing.

The core has support for group-set-addressing if the IT field in Capability register 0 is non-zero. If the number of set address bits (TLB entries) is large enough to cache all page tables, the set address and

tag RAM arrangement will be as described by table 927. If the number of set address bits will allow the whole group ID to be part of the set address, the arrangement will be:

Table 930. Group set address: Set address bits < (group ID bits) + (Physical address bits)

Set address:

31	SB	(GB-1)	0
Not present	Low bits of physi-cal address	Group ID	

Contents of Tag RAM:

31	1	0
Not present	High bits of physical address	V

0 Valid (V) - Signals that addressed position in cache contains valid data

If only part of the group ID can be used for the set address, the arrangement will be:

Table 931. Group set address: Set address bits < (group ID bits)

Set address:

31	GB-SB-1	0
Not present	Low part of group ID	

Contents of Tag RAM:

31		1	0
Not present	Physical address	High part of Group ID	V

0 Valid (V) - Signals that addressed position in cache contains valid data

Group-set-addressing is enabled via the GS field in the core's Control register.

64.5.4 TLB flush operation

If the contents of a page table is modified the TLB must be flushed by writing to the TLB/Cache Flush Register. The TLB/Cache Flush register contains fields to flush the entire TLB or to flush the entries belonging to a specified group. In order to flush entries for a specific group, group-set-addressing must be implemented and enabled. Performing a group flush without group-set-addressing may only flush part of the TLB and can lead to unexpected behavior.

When working in IOMMU mode, the core can be configured to not store a IOPTE in the TLB if the IOPTE's valid (V) bit is cleared. This behavior is controller via the SIV field in the core's Control register.

The core will not propagate any transfers while a flush operation is in progress.

64.6 Fault-tolerance

In order to attain fault-tolerance the core should be implemented with inferred memory technology for the read buffer, write buffer and any FCFS buffer. This will implement the buffers in flip-flops and the core must then be implemented using techniques such as radiation hardened registers or TMR insertion.

The Access Protection Vector cache and IOMMU TLB can be implemented with the same options as above or using non-protected memory cells. When using non-protected memory cells the core can be implemented to use byte-parity to protect entries in the cache/TLB. If an error is detected it will be processed as a cache/TLB miss and the data will be re-read from main memory. A detected error will also be reported via the core's status register and the core also has support for signaling errors via its statistic output.

Errors can be injected in the Access Protection Vector cache and IOMMU TLB via the Data and Tag RAM Error Injection registers.

64.7 Statistics

In order to record statistics, a LEON4 Statistics Unit should be connected to the core. The core has the following statistics outputs:

*Table 932.*IOMMU Statistics

Output	Description
hit	High for one cycle during TLB/cache hit.
miss	High for one cycle during TLB/cache miss
pass	High for one cycle during passthrough access
accok	High for one cycle during access allowed
accerr	High for one cycle during access denied
walk	High while core is busy performing a table walk or accessing the access protection vector
lookup	High while core is performing cache lookup/table walk
perr	High for one cycle when core detects a parity error in the APV cache

64.8 Multi-bus bridge

The core can be instantiated in a version with two AHB master interfaces. These interfaces can be connected to separate AHB buses. The top-level entity griommu_mb contains additional signals for the second AHB master interface. Using the griommu_mb entity will enable bus select fields in the core's master configuration registers and the LB field in the core's control register. The bus select fields in the Master configuration registers allows the user to select which AHB master interface that should be used for accesses initiated by a specific master. The control register field LB selects which AHB master interfaces that should be used when the core fetches IOPTEs or APV bit vector data from memory.

Note that the bus selection in registers is active even if the core is disabled through the core's control register.

64.9 ASMP support

In some systems there may be a need to have separated instances of software each controlling a group of masters. In this case, sharing of the IOMMU register interface may not be wanted as it would allow software to modify the protection settings for a group of masters that belongs to another software instance. The core can be implemented with ASMP support to support systems where software entities are separated by address space. In this case, the core's register interface is mirrored on different 4 KiB pages. Different write protection settings can be set for each mirrored block of registers. This allows use of a memory management unit to control that software running can write to one, and only one, subset of registers.

When ASMP support is enabled, the field NARB in Capability register 0 is non-zero. The value of NARB tells how many ASMP register blocks that are available. Each ASMP register block mirrors the standard register set described in section 64.10 with the addition that some registers may be write protected. Table 933 contains a column that shows if a register is writable when accessed from an ASMP register block. The core's Control register, Master configuration register(s), Diagnostic cache registers, the ASMP access control register(s) can never be written via ASMP register block. These registers are only available in the first register set starting at the core register set base address. ASMP register block *n* is mapped at an offset *n**0x1000 from the core's register base address.

Software should first set up the IOMMU and assign the masters into groups. Then the ASMP control registers should be configured to constrain which registers that can be written from each ASMP block. After this initialization is done, other parts of the software environment can be brought up.

As an example, consider the case where OS A will control masters 0, 1 and 4 while OS B will control masters 2 and 3. In this case it may be appropriate to map masters 0, 1 and 4 to group 0 and master 2 and 3 to group 1. The ASMP access control registers can then be configured to only allow accesses to the Group control register for group 0 from ASMP register block 1 and likewise only allow accesses to the Group control register for group 1 from ASMP register block 2.

OS A will then map in ASMP register block 1 (registers within page located at core base offset + 0x1000) and OS B will then map in ASMP register block 2 (registers within page located at core base offset + 0x2000). This way OS a will be able to change the base address and the properties of group 0, containing its masters, without being able to change the protection mechanisms of group 1 belonging to OS B. Note that since an OS is able to flush the TLB/cache it is able to impact the I/O performance of masters assigned to other OS instances. Also note that care must be taken when clearing status bits and setting the mask register that controls interrupt generation.

64.10 Registers

The core is programmed through registers mapped into AHB I/O address space. All accesses to register address space must be made with word (32-bit) accesses.

*Table 933.*GRIOMMU registers

AHB address offset	Register	Writable in ASMP block
0x00	Capability register 0	No
0x04	Capability register 1	No
0x08	Capability register 2	No
0x0C	Reserved	-
0x10	Control register	No
0x14	TLB/cache flush register	Yes, protected**
0x18	Status register	Yes, protected**
0x1C	Interrupt mask register	Yes, protected**
0x20	AHB Failing Access register	No
0x24 - 0x3C	Reserved, must not be accessed	-
0x40 - 0x7C	Master configuration registers. Master n configuration register is located at offset 0x40 + n*0x4.	No
0x80-0xBC	Group control registers. Group n's control register is located at offset 0x80 + n*0x4.	Yes, protected**
0xC0	Diagnostic cache access register	No
0xC4 - 0xE0	Diagnostic cache access data registers 0 - 7	No
0xE4	Diagnostic cache access tag register	No
0xE8	Data RAM error injection register	No
0xEC	Tag RAM error injection register	No
0xF0 - 0xFF	Reserved, must not be accessed	No
0x100 - 0x13F	ASMP access control registers. The control register for ASMP block n is located at offset 0x100+n*0x4.	No

* Register is duplicated in ASMP register block at offset 0x1000 + register offset. The number of ASMP register blocks is given by the NARB field in Capability register 0. ASMP register block *n* starts at offset *n**0x1000. Register is only writable if allowed by the corresponding ASMP access control register field.

64.10.1 Capability Register 0

*Table 934.*0x00 - LAP0 - Capability register 0

31	30	29	28	27 24	23 20	19	18 17	16	15	14	13	12	11 9	8	7 4	3 0
A	AC	CA	CP	RESERVED	NARB	CS	FT	ST	I	IT	IA	IP	RESERVED	MB	GRPS	MSTS
*	*	*	*	0	*	*	*	*	*	*	*	*	0	*	*	*
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

31 Access Protection Vector (A) - If this bit is ‘1’, the core has support for Access Protection Vector

30 Access Protection Vector Cache (AC) - If this bit is ‘1’, the core has a internal cache for Access Protection vector lookups.

29 Access Protection Vector Cache Addressing (CA):
0: Core only supports standard addressing, group number is used as tag
1: Core supports using group ID as part of cache set address

28 Access Protection Vector Cache Pipeline (CP) - If this bit is set to ‘1’ the core has a pipeline stage added on the APV cache’s address. This means one cycle additional latency.

27:24 RESERVED

23:20 ASMP Register Blocks (NARB) - This field contains the number of ASMP register blocks that the core implements. If this field is non-zero the core has NARB ASMP register blocks with the first block starting at offset 0x1000 and the last block starting at offset NARB*0x1000.

19 Configurable Page Size (CS) - If this bit is ‘1’ the core supports several page sizes and the size is set via the Control register field PGSZ. If this bit is ‘0’, a fixed page size of 4 KiB is used.

18:17 Fault Tolerance (FT) - “00” - No fault tolerance, “01” - APV cache and/or IOMMU TLB is protected by parity

16 Statistics (S) - If this field is ‘1’ the core collects statistics

15 IOMMU functionality enable (I) - If this bit is ‘1’, the core has support for IOMMU functionality.

14 IOMMU TLB (IT) - If this bit is ‘1’, the core has an IOMMU Translation Lookaside Buffer (TLB)

13 IOMMU Addressing (IA):
0: Core only supports standard addressing, group number is used as tag
1: Core supports using group ID as part of TLB set address

12 IOMMU TLB Address Pipeline (IP) - If this bit is set to ‘1’ the core has a pipeline stage added on the TLB’s address. This means one cycle additional latency.

11:9 RESERVED

8 Multi-bus (MB) - Set to 1 if core is connected to two system buses.

7:4 Number of groups (GRPS) - Number of groups that the core has been implemented to support - 1.

3:0 Numbers of masters (MSTS) - Number of masters that the core has been implemented to support - 1.

Reset value: Implementation dependent

64.10.2 Capability Register 1

*Table 935.*0x04 - LAP1 - Capability register 1

31 20	19 16	15 8	7 5	4 0
CADDR	CMASK	CTAGBITS	CISIZE	CLINES
*	*	*	*	*
r	r	r	r	r

31:20 Access Protection Vector Cacheable Address (CADDR) - If the CMASK field of this register is non-zero the CADDR and CMASK fields specify the base address of the memory area protected by the part of the bit vector that can be cached by the core.

19:16 Access Protection Vector Cacheable Mask (CMASK) - Number of '1's in the Access Protection Vector Cachable mask. If the core is implemented with a Access Protection Vector cache and this value is non-zero, the CMASK field together with the CADDR field specify a memory area protected by a part of the bit vector that can be cached by the core. The CMASK value corresponds to the number of most significant bits of the CADDR field that are matched against the incoming AMBA address when determining if the protection bits for the memory area should be cached. As an example, if CMASK is 1 and CADDR is 0x000, the core will cache protection information for the address range 0x00000000 - 0x7FFFFFFF. With the same mask and CADDR = 0x800, the core would cache protection information for the address range 0x80000000 - 0xFFFFFFFF.

15:8 Access Protection Vector Cache Tag bits (CTAGBITS) - The width in bits of the Access Protection Vector cache's tags.

7:5 Access Protection Vector Access size (CSIZE) - This field indicates the AMBA access size used when accessing the Access Protection Vector in main memory. This is also the cache line size for the APV cache (if enabled). The values are:
000: 32-bit (4 byte)
001: 64-bit (8 byte)
010: 128-bit (16 byte)
011: 256-bit (32-byte)

4:0 Access Protection Vector Cache Lines (CLINES) - Number of lines in the Access Protection Vector cache. The number of lines in the cache is 2^{CLINES}.

64.10.3 Capability Register 2

*Table 936.*0x04 - CAP2 - Capability register 2

31 24	23 20	19 18	17 16	15 8	7 5	4 0
TMASK	RESERVED	MTYPE	TTYPE	TTAGBITS	ISIZE	TLBENT
*	0	*	*	*	*	*
r	r	r	r	r	r	r

31:24 Translation Mask (TMASK) - The incoming IO address bits IOADDR[31:24] must match this field, depending on the setting of the ITR field in the core's Control register, for an address translation operation to be performed.

23:20 RESERVED

19:18 IOMMU Type (MTYPE) - Shows IOMMU implementation type. This field is always 0, other values are reserved for future versions of the core. If this field is non-zero, it should trigger a software alert as future versions of the core may not be backward compatible.

17:16 TLB Type (TTYPE) - Show implementation of Translation Lookaside Buffer. This field is always 0, other values are reserved for future versions of the core. If this field is non-zero, it should trigger a software alert as future versions of the core may not be backward compatible.

15:8 TLB Tag bits (TTAGBITS) - The width in bits of the TLB tag.

7:5 IOMMU Access size (ISIZE) - This field indicates the AMBA access size used when accessing page tables in main memory. This is also the line size for the TLB (if enabled). The values are:
000: 32-bit (4 byte)
001: 64-bit (8 byte)
010: 128-bit (16 byte)
011: 256-bit (32-byte)

4:0 TLB entries (TLBENT) - Number of entries in the TLB. The number of entries is 2^{TLBENT}.

Reset value: implementation dependent

64.10.4 Control Register

*Table 937.*0x10 - CTRL - Control register

31 21	20 18	17	16	15 12	11	10	9 8	7	6	5	4	3	2 1	0
RESERVED	PGSZ	LB	SP	ITR	DP	SIV	HPROT	AU	WP	DM	GS	CE	PM	EN
0	0	0	0	0	0	0	0	0	0	0	0	0	*	0
r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31:21 RESERVED

20:18 Page Size (PGSZ) - The value in this field determines the page size mapped by page table entries and bit vector positions. Valid values are:

000: 4 KiB
001: 8 KiB
010: 16 KiB
011: 32 KiB
100: 64 KiB
101: 128 KiB
110: 256 KiB
111: 512 KiB

This field is only writable if the CS field in Capability register 0 is non-zero.

17 Lookup bus (LB) - The value of this bit controls AHB master interface to use for fetching bit vector and/or page table entries from memory when the core has been implemented with support for multiple buses (multiple AHB master interfaces). If this field is '0', the first master interface will be used for vector/table lookups. If this field is '1', the second master interface will be used for lookups.

This field is only writable if the MB field in Capability register 0 is non-zero.

Table 937. 0x10 - CTRL - Control register

16	SPLIT support (SP) - The value of this bit controls if the core can issue AMBA SPLIT responses to masters on the IO bus. If this bit is '1' the core will use AMBA SPLIT responses. If this bit is '0', the core will insert waitstates and not issue AMBA SPLIT responses. This bit is read-only if the core has been implemented with support for only one response mode. If this bit is writable, software must make sure that the IO bus is free and that the core is not handling any ongoing accesses before changing the value of this bit. The core performs rudimentary checks in order to determine if the slave side is idle before changing SPLIT behavior. Therefore AMBA SPLIT responses may not be disabled or enabled immediately after this bit is written.
15:12	IOMMU Translation Range (ITR) - This field defines the size of the address range translated by the core's IOMMU functionality. The size of the decoded address range is 16 MiB * 2^{ITR} and the decoded memory area is located on an address with the most significant bits specified by the TMASK field in Capability register 2, unless ITR = 8 in which case the whole address space is covered by the translated range.
11	Disable Prefetch (DP) - When this bit is '1' the core will not perform any prefetch operations. This bit is read only if the core has been implemented without support for prefetching data. During normal operation prefetch of data improves performance and should be enabled (the value of this bit should be '0'). Prefetching may need to be disabled in scenarios where IOMMU protection is enabled, which leads to a prefetch operation on every incoming burst access, and when the core is used in bi-directional bridge configurations where dead locks may be resolved by the core dropping prefetch data.
10	Save Invalid IOPTE (SIV) - If this field is '1' the core will save IOPTEs that have their valid (V) bit set to '0' if the core has been implemented with a TLB. If this field is '0' the core will not buffer an IOPTE with valid (V) set to '0' and perform an page table lookup every time the page covered by the IOPTE is accessed. If the value of this field is changed, a TLB flush must be made to remove any existing IOPTEs from the core's internal buffer. Also if this field is set to '0', any diagnostic accesses to the TLB should not set the IOPTE valid bit to '0' unless the Tag valid bit is also set to '0'. This field is only accessible if the core has support for IOMMU protection and is implemented with a Translation Lookaside Buffer (TLB).

Table 937. 0x10 - CTRL - Control register

9:8	HPROT encoding (HPROT) - The value of this field will be assigned to the AMBA AHB HPROT signal bits 3:2 when the core is fetching protection data from main memory. HPROT(3) signals if the access is cacheable and HPROT(2) signals if the access is bufferable. This field is only used when the core has been implemented with support for Access Protection Vector or IOMMU functionality.
7	Always Update (AU) - If this bit is set to '0' the AHB failing access register will only be updated if the Access Denied (AD) bit in the Status register is '0' when the access is denied. Otherwise the AHB failing access register will be updated each time an access is denied, regardless of the Access Denied (AD) bit's value.
6	Write Protection only (WP) - If this bit is set to '1' the core will only used the Access Protection Vector to protect against write accesses. Read accesses will be propagated over the core without any access restriction checks. This will improve the latency for read operations. This field has no effect when the core is using IOMMU protection (PM field = "01"). When using IOMMU protection all accesses to the range determined by TMASK and ITR will be checked against the page table, unless the access is from a master that is assigned to an inactive group or a group in pass-through mode.
5	Diagnostic Mode (DM) - If this bit is set to '1' the core's internal buffers can be accessed via the Diagnostic interface (see Diagnostic cache access register) when the DE field of the Status register has been set by the core. Set this bit to '0' to leave Diagnostic mode. While in this mode the core will not forward any incoming AMBA accesses.
4	Group-Set-addressing (GS) - When this bit is set to '1', the core will use the group number as part of the Access Protection Vector cache set address. This bit can only be set if fields A and CA, or I and IA, of Capability register 0 are non-zero.
3	Cache/TLB Enable (CE) - When this bit is set to '1', the core's internal cache/TLB is enabled. Note that the core can be implemented without internal cache/TLB. Capability register 0, fields AC and IT show if the core has internal cache.
2:1	Protection Mode (PM) - This value selects the protection mode to use. "00" selects Group Mode and/or Access Protection Vector mode (if available). "01" selects IOMMU mode. This field is read only if the core only has support for one mode setting.
0	Enable (EN) - Core enable. If this bit is set to 1 the core is enabled. If this bit is set to 0 the core is disabled and in pass-through mode. After writing this bit software should read back the value. The change has not taken effect before the value of this bit has changed. The bit transition may be blocked if the core is in diagnostic access mode or otherwise occupied.

64.10.5 TLB/cache Flush Register

*Table 938.*0x14 - FLUSH - TLB/cache flush register

31 8	7 4	3 2	1	0
RESERVED	FGRP	RES	GF	F
0	0	0	0	0
r	rw	r	rw	rw

31:1 RESERVED

7:4 Flush Group (FGRP) - This field specifies the group to be used for a Group Flush, see GF field below.

3:2 RESERVED

1 Group Flush (GF) - When this bit is written to '1' the cache entries for the group selected by the FGRP field will be flushed. More precisely the core will use the FGRP field as (part of the) set address when performing the flush. This flush option is only available if the core has support for group set addressing (CA field of Capability register 1 is non-zero). This flush option must only be used if the GS bit in the Control register is set to '1', otherwise old data may still be marked as valid in the Access Protection Vector cache or IOMMU TLB. This bit will be reset to '0' when a flush operation has completed. A flush operation also affects the FL and FC fields in the Status register.

0 Flush (F) - When this bit is written to '1' the core's internal cache will be flushed. This bit will be reset to '0' when a flush operation has completed. A flush operation also affects the FL and FC fields in the Status register.

64.10.6 Status Register

*Table 939.*0x18 - STAT - Status register

31 6	5	4	3	2	1	0
RESERVED	PE	DE	FC	FL	AD	TE
0	0	0	0	0	0	0
r	uc	uc	uc	uc	uc	uc

31:6 RESERVED

5 Parity Error (PE) - The core sets this bit to '1' when it detects a parity error in the tag or data RAM of the APV cache. This field is cleared by writing '1' to this position, writes of '0' have no effect.

4 Diagnostic Mode Enabled (DE) - If this bit is set to '1' the core is in Diagnostic Mode where the core's internal buffers can be accessed via the Diagnostic access registers. While in this mode the core will not forward any incoming AMBA accesses.

3 Flush Completed (FC) - The core sets this bit to '1' when a flush operation completes. This field is cleared by writing '1' to this position, writes of '0' have no effect.

2 Flush started (FL) - The core sets this bit to '1' when a Flush operation has started. This field is cleared by writing '1' to this position, writes of '0' have no effect.

1 Access Denied (AD) - The core denied an AMBA access. This field is cleared by writing '1' to this position, writes of '0' have no effect.

0 Translation Error (TE) - The core received an AMBA ERROR response while accessing the bit vector or page tables in memory. This also leads to the incoming AMBA access being inhibited. Depending on the status of the Control register's AU field and this register's AD field this may also lead to an update of the AHB Failing Access register.

64.10.7 Interrupt Mask Register

*Table 940.*0x1C - IMASK - Interrupt mask register

31 6	5	4	3	2	1	0
RESERVED	PEI	R	FCI	FLI	ADI	TEI
0	0	0	0	0	0	0
r	rw	rw	rw	rw	rw	rw

31:6 RESERVED

5 Parity Error Interrupt (PEI) - If this bit is set to '1' an interrupt will be generated when the PE bit in the Status register transitions from '0' to '1'.

4 RESERVED

3 Flush Completed Interrupt (FCI) - If this bit is set to '1' an interrupt will be generated when the FC bit in the Status register transitions from '0' to '1'.

2 Flush Started Interrupt (FLI) - If this bit is set to '1' an interrupt will be generated when the FL bit in the Status register transitions from '0' to '1'..

1 Access Denied Interrupt (ADI) - If this bit is set to '1' an interrupt will be generated when the AD bit in the Status register transitions from '0' to '1'.

0 Translation Error Interrupt (TEI) - If this bit is set to '1' an interrupt will be generated when the TE bit in the Status register transitions from '0' to '1'.

Reset value: 0x00000000

64.10.8 Failing Access Register

*Table 941.*0x20 - AHBFAS - failing access register

31 5	4	3 0
FADDR[31:5]	FW	FMASTER
0	0	0
r	r	r

31:5 Failing Address (FADDR[31:5]) - Bits 31:5 of IO address in access that was inhibited by the core. This field is updated depending on the value of the Control register AU field and the Status register AD field.

4 Failing Write (FW) - If this bit is set to '1' the failed access was a write access, otherwise the failed access was a read access. This field is updated depending on the value of the Control register AU field and the Status register AD field.

3:0 Failing Master (FMASTER) - Index of the master that initiated the failed access. This field is updated depending on the value of the Control register AU field and the Status register AD field.

Reset value: 0x00000000

64.10.9 Master Configuration Register(s)

*Table 942.*0x40-0x7C - MSTLFGO-15 - Master configuration register(s)

31 24	23 12	11 5	4	3 0
VENDOR	DEVICE	RESERVED	BS	GROUP
*	*	0	0	0
r	r	r	rw	rw

31: 24 Vendor ID (VENDOR) - GRLIB Plug'n'play Vendor ID of master

23: 12 Device ID (DEVICE) - GRLIB Plug'n'play Device ID of master

11: 5 RESERVED

Table 942. 0x40-0x7C - MSTLFGO-15 - Master configuration register(s)

4	Bus select for master (BS) - Master n's bus select register is located at register address offset 0x40 + n*0x4. This field specifies the the bus to use for accesses initiated by AHB master n. This field is only available if the MB field in Capability register 0 is non-zero. Bus selection is active even if the core control register enable bit is zero.
3:0	Group assignment for master - Master n's group assignment field is located at register address offset 0x40 + n*0x4. This field specifies the group to which a master is assigned.

64.10.10 Group Control Register(s)

*Table 943.*0x80-0xAC - GRPCTRCO-15 - Group control register(s)

31	1	0
BASE[31:2+SIZE]	P	AG
0	0	0
rw	rw	rw

31: 2 Base address (BASE) - Group n's control register is located at offset 0x80 + n*0x4. This field contains the base address of the data structure for the group.

The number of bits writeable in the data structure base address depends on the access size used to fetch entries in the Access Protection Vector and/or the IOMMU page table. The access size is given in the ISIZE and CSIZE Capability register fields.

This field is only writable if the core has been implemented with support for Access Protection Vector and/or IOMMU functionality.

1 Pass-through (P) - If this bit is set to '1' and the group is active (see bit 0 below) the core will pass-through all accesses made by master in this group and not use the address specified by BASE to perform look-ups in main memory. Note that this also means that the access will pass through untranslated when the core is using IOMMU protection (even if the access is outside the translated range defined by TMASK in Capability register 2).

If this bit is set to '0', the core will use the contents in its cache, or in main memory, to perform checks and possibly address translation on incoming accesses.

If the core has been implemented without support for Access Protection Vector and IOMMU, this field is disabled.

0 Active Group (AG) - Indicates if the group is active. If this bit is set to '0', all accesses made by masters assigned to this group will be blocked.

If the core has been implemented without support for Access Protection Vector and IOMMU, accesses will be propagated if this bit is set to '1'. If the core has been implemented with support for Access Protection Vector and/or IOMMU the core will check the P field of this register and possibly also the in-memory data structure before allowing or blocking the access.

64.10.11 Diagnostic Cache Access Register

*Table 944.*0xC0 - DIAGCTRL - Diagnostic cache access register

31	30	29 22	21	20	19	18 0
DA	RW	RESERVED	DP	TP	R	SETADDR
0	0	0	0	0	0	NR
rw	rw	r	rw	rw	r	rw*

31 Diagnostic Access (DA) - When this bit is set to '1' the core will perform a diagnostic operation to the cache address specified by the SETADDR field. When the operation has finished this bit will be reset to '0'.

30 Read/Write (RW) - If this bit is '1' and the A field is set to '1' the core will perform a read operation to the cache. The result will be available in the Diagnostic cache access tag and data register(s). If this bit is set to '0' and the A field is set to '1', the core will write the contents of the Diagnostic cache access tag and data registers to the internal cache.

29:22 RESERVED

21 Data Parity error (DP) - This bit is set to '1' if a parity error has been detected in the word read from the cache's data RAM. This bit can be set even if no diagnostic cache access has been made and it can also be set after a cache write operation. This bit is read-only.

20 Tag Parity error (TP) - This bit is set to '1' if a parity error has been detected in the word read from the cache's tag RAM. This bit can be set even if no diagnostic cache access has been made and it can also be set after a cache write operation. This bit is read-only.

19 RESERVED

*Table 944.*0xC0 - DIAGCTRL - Diagnostic cache access register

18:0	Cache Set Address (SETADDR) - Set address to use for diagnostic cache access. When a read operation has been performed, this field should not be changed until all wanted data has been read from the Diagnostic cache access data and tag registers. Changing this field invalidates the contents of the data and tag registers.

* This register can only be accessed if the core has an internal cache and the DE bit in the Status register is set

64.10.12 Diagnostic Cache Access Data Register 0 - 7

Table 945. 0xC4-0xE0 - DIAG0 - Diagnostic cache access data register 0 - 7

31	0
CDATAn	
NR	
rw*	

31:0 Cache data word n (CDATAn) - The core has 8 Diagnostic cache access data registers. Diagnostic cache access data register n holds data bits [31+32*n:32*n] in the cache line.

* This register can only be accessed if the core has an internal cache and the DE bit in the Status register is set
Reset value: Undefined

64.10.13 Diagnostic Cache Access Tag Register

Table 946. 0xE4 - DIAG 7 - Diagnostic cache access tag register

31	0
TAG	V
NR	
rw*	

31:1 Cache tag (TAG) - The size of the tag depends on cache size. The contents of the tag depends on cache size and addressing settings.

0 Valid (V) - Valid bit of tag

* This register can only be accessed if the core has an internal cache and the DE bit in the Status register is set
Reset value: Undefined

64.10.14 Data RAM Error Injection Register

Table 947. 0xE8 - DERRI - Data RAM error injection register

31	0
DPERRINJ	
0	
rw	

31:0 Data RAM Parity Error Injection (DPERRINJ) - Bit DPERRINJ[n] in this register is XOR:ed with the parity bit for data bits [*7+8*n*:*8*n*] in the data RAM.

* This register can only be accessed if the core has an internal cache and the FT field in Capability register 0 is non-zero
Reset value: 0x00000000

64.10.15 Tag RAM Error Injection Register

Table 948. 0xEC - TERRI - Tag RAM error injection register

31	0
TPERRINJ	
0	
rw	

0 Tag RAM Parity Error Injection (TPERRINJ) - Bit TPERRINJ[n] in this register is XOR:ed with the parity bit for tag bits [*7+8*n*:*8*n*] in the tag RAM.

* This register can only be accessed if the core has an internal cache and the FT field in Capability register 0 is non-zero
Reset value: 0x00000000

64.10.16 ASMP Access Control Register

*Table 949.*0x100-0x10C - ASMPCTRL - ASMP access control register(s)

31 19	18	17	16	15 0
RESERVED	FC	SC	MC	GRPACCSZCTRL
0	0	0	0	0
r	rw	rw	rw	rw

31: 19	RESERVED
18	Flush register access control (FC) - If this bit is set to '1' in the ASMP control register at offset 0x100 + n*0x4 then the TLB/cache flush register in ASMP register block n is writable. Otherwise writes to the TLB/cache flush register in ASMP register block n will be inhibited.
17	Status register access control (SC) - If this bit is set to '1' in the ASMP control register at offset 0x100 + n*0x4 then the Status register in ASMP register block n is writable. Otherwise writes to the Status register in ASMP register block n will be inhibited.
16	Mask register access control (MC) - If this bit is set to '1' in the ASMP control register at offset 0x100 + n*0x4 then the Master register in ASMP register block n is writable. Otherwise writes to the Mask register in ASMP register block n will be inhibited.
15:0	Group control register access control (GRPACCSZCTRL) - ASMP register block n's group access control field is located at register address offset 0x100 + n*0x4. This field specifies which of the Group control registers that are writable from an ASMP register block. If GRPACCSZCTRL[i] in the ASMP access control register at offset 0x100 + n*0x4 is set to '1' then Group control register i is writable from ASMP register block n.

Reset value: 0x00000000

64.11 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x04F. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

If implemented, the core's second AHB master interface has 0x01 (Frontgrade Gaisler) and device identifier 0x010.

64.12 Implementation

64.12.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

64.12.2 Technology mapping

The core has two technology mapping generics *memtech* and fcfsmtech. *memtech* selects which memory technology that will be used to implement the FIFO memories. *fcfsmtech* selects the memory technology to be used to implement the First-come, first-served buffer, if FCFS is enaled.

64.12.3 RAM usage

The core instantiates one or several *syncram_2p* blocks from the technology mapping library (TECHMAP). If prefetching is enabled max(*mstmaccsz*, *slvaccsz*)/32 *syncram_2p* block(s) with organization (max*(rburst,iburst*)-max(*mstmaccsz*, *slvaccsz*)/32) x 32 is used to implement read FIFO (max*(rburst,iburst*) is the size of the read FIFO in 32-bit words). max(*mstmaccsz*, *slvaccsz*)/32 *syn-*

cram_2p block(s) with organization *(wburst* - max(*mstmaccsz*, *slvaccsz*)/32) x 32, is always used to implement the write FIFO (where *wburst* is the size of the write FIFO in 32-bit words).

If the core has support for first-come, first-served ordering then one *fcfs* x 4 *syncram_2p* block will be instantiated, using the technology specified by the VHDL generic *fcfsmtech*.

If the core has an Access Protection Vector cache and/or IOMMU TLB, the cache/TLB will be implemented using one *syncramft* block for the tag RAM and one *syncramft* block for the data RAM.

64.12.4 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

64.13 Configuration options

Table 950 shows the configuration options of the core (VHDL generics).

*Table 950.*Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
memtech	Memory technology		
iohsindex	Slave I/F AHB index on IO bus	0 to NAHBMST-1	0
syshmindex	Master I/F AHB index on System bus	0 to NAHBMST-1	0
syshmindex2	Master I/F AHB index for second AHB interface. Only available if the entity griommu_mb is instantiated.	0 to NAHBMST-1	0
syshsindex	Index for register slave AHB I/F connected to same bus as core Master I/F	0 to NAHBMST-1	0
syshmaddr	ADDR field of AHB slave BAR 0 on system bus	0 - 16#FFF#	0
syshmask	MASK field of AHB slave BAR 0 on system bus. The required value of this generic depends on the setting of generic narb (see below).	0 - 16#FFF#	16#FFF#
syshirq	Interrupt line to use for IOMMU interrupts	1 - NAHBIRQ-1	1
dir	0 - clock frequency on the master bus is lower than or equal to the frequency on the slave bus 1 - clock frequency on the master bus is higher than or equal to the frequency on the slave bus (for VHDL generic *ffact* = 1 the value of dir does not matter)	0 - 1	0
ffact	Frequency scaling factor between AHB clocks on master and slave buses.	1 - 15	2
slv	Slave bridge. Used in bi-directional bridge configuration where *slv* is set to 0 for master bridge and 1 for slave bridge. When a deadlock condition is detected slave bridge (*slv*=1) will give RETRY response to current access, effectively resolving the deadlock situation. This generic must only be set to 1 for a bridge where the frequency of the bus connecting the master interface is higher or equal to the frequency of the AHB bus connecting to the bridge's slave interface. Otherwise a race condition during access collisions may cause the bridge to deadlock.	0 - 1	0
pfen	Prefetch enable. Enables read FIFO.	0 - 1	0

*Table 950.*Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
irqsync	Interrupt forwarding. Forward interrupts from slave interface to master interface and vice versa. 0 - no interrupt forwarding, 1 - forward interrupts 1 - 15, 2 - forward interrupts 0 - 31. Since interrupts are forwarded in both directions, interrupt forwarding should be enabled for one bridge only in a bi-directional AHB/AHB bridge.	0 - 2	0
wburst	Length of write bursts in 32-bit words. Determines write FIFO size and write burst address boundary. If the wburst generic is set to 2 the bridge will not perform write bursts over a 2x4=8 byte boundary. This generic must be set so that the buffer can contain two of the maximum sized accesses that the bridge can handle.	2 - 32	8
iburst	Instruction fetch burst length. This value is only used if the generic *ibrsten* is set to 1. Determines the length of prefetching instruction read bursts on the master side. The maximum of (iburst,rburst) determines the size of the core's read buffer FIFO.	4 - 8	8
rburst	Incremental read burst length. Determines the maximum length of incremental read burst of unspecified length (INCR) on the master interface. The maximum of *rburst* and *iburst* determine the read burst boundary. As an example, if the maximum value of these generics is 8 the bridge will not perform read bursts over a 8x4=32 byte boundary. This generic must be set so that the buffer can contain two of the maximum sized accesses that the bridge can handle. For systems where AHB masters perform fixed length burst (INCRx , WRAPx) *rburst* should not be less than the length of the longest fixed length burst.	4 - 32	8
bar0	Address area 0 decoded by the bridge's slave interface. Appears as memory address register (BAR0) on the slave interface. The generic has the same bit layout as bank address registers with bits [19:18] suppressed (use functions ahb2ahb_membar and ahb2ahb_iobar in gaisler.misc package to generate this generic).	0 - 1073741823	0
bar1	Address area 1 (BAR1)	0 - 1073741823	0
bar2	Address area 2 (BAR2)	0 - 1073741823	0
bar3	Address area 3 (BAR2)	0 - 1073741823	0
sbus	The number of the AHB bus to which the slave interface is connected. The value appears in bits [1:0] of the user-defined register 0 in the slave interface configuration record and master configuration record.	0-3	0
mbus	The number of the AHB bus to which the master interface is connected. The value appears in bits [3:2] of the user-defined register 0 in the slave interface configuration record and master configuration record.	0-3	0

Table 950. Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
ioarea	Address of the I/O area containing the configuration area for AHB bus connected to the bridge's master interface. This address appears in the bridge's slave interface user-defined register 1. In order for a master on the slave interface's bus to access the configuration area on the bus connected to the bridge's master interface, the I/O area must be mapped on one of the bridge's BARs. If this generic is set to 0, some tools, such as Frontgrade Gaisler's GRMON debug monitor, will not perform Plug'n'Play scanning over the bridge.	0 - 16#FFF#	0
ibrsten	Instruction fetch burst enable. If set, the bridge will perform bursts of *iburst* length for opcode access (HPROT[0] = '0'), otherwise bursts of *rburst* length will be used for both data and opcode accesses.	0 - 1	0
lckdac	Locked access error detection and correction. Locked accesses may lead to deadlock if a locked access is made while an ongoing read access has received a SPLIT response. The value of *lckdac* determines how the core handles this scenario: 0: Core will deadlock 1: Core will issue an AMBA ERROR response to the locked access 2: Core will allow both accesses to complete. If the core is used to create a bidirectional bridge, a deadlock condition may arise when locked accesses are made simultaneously in both directions. With *lckdac* set to 0 the core will deadlock. With *lckdac* set to a non-zero value the slave bridge will issue an ERROR response to the incoming locked access.	0 - 2	0
slvmaccsz	The maximum size of accesses that will be made to the bridge's slave interface. This value must equal *mstmaccsz* unless *rdcomb* /= 0 and *wrcomb* /= 0.	32 - 256	32
mstmaccsz	The maximum size of accesses that will be performed by the bridge's master interface. This value must equal *mstmaccsz* unless *rdcomb* /= 0 and *wrcomb* /= 0.	32 - 256	32
rdcomb	Read combining. If this generic is set to a non-zero value the core will use the master interface's maximum AHB access size when prefetching data and allow data to be read out using any other access size supported by the slave interface. If slvmaccsz > 32 and mstmaccsz > 32 and an incoming single access, or access to a non-prefetchable area, is larger than the size supported by the master interface the bridge will perform a series of small accesses in order to fetch all the data. If this generic is set to 2 the core will use a burst of small fetches. If this generic is set to 1 the bridge will not use a burst unless the incoming access was a burst. Read combining is only supported for single accesses and incremental bursts of unspecified length.	0 - 2	0

*Table 950.*Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
wrcomb	Write combining. If this generic is set to a non-zero value the core may assemble several small write accesses (that are part of a burst) into one or more larger accesses or assemble one or more accesses into several smaller accesses. The settings are as follows: 0: No write combining 1: Combine if burst can be preserved 2: Combine if burst can be preserved and allow single accesses to be converted to bursts (only applicable if slvmaccsz > 32) Only supported for single accesses and incremental bursts of unspecified length	0 - 2	0
combmask	Read/write combining mask. This generic determines which ranges that the core can perform read/write combining to (only available when rdcomb respectively wrcomb are non-zero). The value given for combmask is treated as a 16-bit vector with LSB bit (right-most) indicating address 0x0 - 0x10000000. Making an access to an address in an area marked as '0' in combmask is equivalent to making an access over a bridge with rdcomb = 0 and wrcomb = 0. However, combmask is not taken into account when the core performs a prefetch operation (see pfen generic). When a prefetch operation is initiated, the core will always use the maximum supported access size (when rdcomb /= 0).	0 - 16#FFFF#	16#FFFF#
allbrst	Support all burst types 2: Support all types of burst and always prefetch for wrapping and fixed length bursts. 1: Support all types of bursts 0: Only support incremental bursts of unspecified length See section 64.2.7 for more information. When allbrst is enabled, the core's read buffer (size set via rburst/iburst generics) must have at least 16 slots.	0 - 2	0
ifctrlen	Interface control enable. When this generic is set to 1 the input signals *ifctrl.mstifen* and *ifctrl.slvifen* can be used to force the AMBA slave respectively master interface into an idle state. This functionality is intended to be used when the clock of one interface has been gated-off and any stimuli on one side of the bridge should not be propagated to the interface on the other side of the bridge. When this generic is set to 0, the ifctrl.* input signals are unused.	0 - 1	0
fcfs	First-come, first-served operation. When this generic is set to a non-zero value, the core will keep track of the order of incoming accesses and handle the requests in the same order. If this generic is set to zero the bridge will not preserve the order and leave this up to bus arbitration. If FCFS is enabled the value of this generic must be higher or equal to the number of masters that may perform accesses over the bridge.	0 - NAHBMST	0
fcfsmtech	Memory technology to use for FCFS buffer. When VHDL generic *fcfs* is set to a non-zero value, the core will instantiate a 4 bit x *fcfs* buffer to keep track of the incoming master indexes. This generic decides the memory technology to use for the buffer.	0 - NTECH	0 (inferred)

*Table 950.*Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
scantest	Enable scan support	0 - 1	0
split	Use AMBA SPLIT responses. When this generic is set to 1 the core will issue AMBA SPLIT responses. When this generic is set to 0 the core will insert waitstates instead and may also issue AMBA RETRY responses. If this generic is set to 0, the *fcfs* generic must also be set to 0, otherwise a simulation failure will be asserted.	0 - 1	1
dynsplit	Dynamic SPLIT responses. If this generic is non-zero the Control register field SP will be writable. This allows software to control if the core should use AMBA SPLIT responses or waitstates on the IO bus. The VHDL generic *split* must be set to 1 if this generic is set to 1.	0 - 1	0
nummst	Number of masters connected to the bus that the core's slave interface connects to.	1 - NAHBMST-1	1
numgrp	Number of groups	1 - NAHBMST-1	1
stat	Enable statistics outputs	0 - 1	0
apv	Include support for Access Protection Vector (APV). Setting this generic to 1 includes support.	0 - 1	1
apvc_en	Access Protection Vector cache. 0: disabled, 1: enabled.	0 - 1	0
apvc_ways	Number of ways in Access Protection Vector cache	1 - 1	1
apvc_lines	Number of lines in each way of the Access Protection vector cache. The total size of the data cache in bytes will be apvc_ways * tbw_accsz/8 * apvc_lines. This value must be a power of two. If the core is implemented with an IOMMU TLB, the maximum value of this generic and VHDL generic *tlb_num* determines the number of lines in the cache.		16
apvc_tech	Access Protection Vector cache memory technology. This generic decides the technology setting for the cache's tag and data RAM.	0 - NTECH	0 (inferred)
apvc_gseta	Allow use of group ID as part of cache set address. This allows cache addressing scheme 2 to be used. The value 0 disables the use of group ID as part of the cache set address and 1 enables the functionality. If the core is implemented with an IOMMU TLB and VHDL generic *tlb_gseta* is set to non-zero value, this will also enable *apvc_gseta*. This generic may only be set to a non-zero value if VHDL generic *numgrp* > 1.	0 - 1	0
apvc_caddr	If generic *apvc_cmask* is non-zero this generic specifies the base address of the memory area that the core will cache protection information for.The area is specified in the same way as addresses for AHB slaves. To cache protection information for the block 0x40000000 - 0x7FFFFFFF, set this generic to 0x400 and *apvc_cmask* to 0xC00.	0 - 16#FFF#	0
apvc_cmask	Specifies size of memory block for which protection information will be cached by the core. If this generic is zero the core will cache protection information for the full AMBA address range. If this generic is 0x800 the core will cache information for half the AMBA address range, the base address for the cacheable area is specified by *apvc_caddr*.	0 - 16#FFF#	0

*Table 950.*Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
apvc_pipe	Insert pipelining registers on APV cache. The master -> group -> cache path may become critical in a design. If there are timing problem on the tag or cache RAM address inputs, set this generic to 1 and suffer one cycle in additional penalty on cache accesses. If the core is implemented with an IOMMU TLB and VHDL generic *tlb_pipe* is set to non-zero value, this will also enable *apvc_pipe*.	0 - 1	0
iommu	Enable IOMMU functionality	0 - 1	0
iommutype	Selects type of IOMMU functionality. Set to 0	0 - 1	0
tlb_num	Number of entries in the IOMMU translation lookaside buffer (TLB). A value of zero here implements the core without a TLB. The width of an entry is determined through the VHDL generic tbw_accsz. The total size of the TLB in bytes will be tbw_accsz/8 * tlb_num. This value must be a power of two. If the core has been implemented with an APV cache, the maximum value of this generic and VHDL generic *apvc_lines* determines the number of entries in the TLB.	0 - 64	0
tlb_type	Selects TLB implementation. Set to 0.	0 - 1	0
tlb_tech	TLB memory technology. This generic decides the technology setting to use for implementing the TLB. In the current version of the bridge this generic and VHDL generic *apvc_tech* must have the same value.	0 - NTECH	0 (inferred)
tlb_gseta	Allow use of group ID as part of TLB set address. This allows cache addressing scheme 2 to be used. The value 0 disables the use of group ID as part of the TLB set address and 1 enables the functionality. If the core is implemented with an APV cache and VHDL generic *apvc_gseta* is set to non-zero value, this will also enable *tlb_gseta*. This generic may only be set to a non-zero value if VHDL generic *numgrp* > 1.	0 - 1	0
tlb_pipe	Insert pipelining registers on TLB address. The master -> group -> TLB path may become critical in a design. If there are timing problem on the tag or cache RAM address inputs, set this generic to 1 and suffer one cycle in additional penalty on cache accesses. If the core is implemented with an APV cache and VHDL generic *apvc_pipe* is set to non-zero value, this will also enable *tlb_pipe*.	0 - 1	0
tmask	Translation mask. Specifies the value that the most significant bits of the IO address must have for an address to be translated. Bits 7:0 of this value specified TMASK[31:24]. The default value 0xff is recommended. However, this may not work well if the IO bus has a GRLIB plug'n'play area. Note that tmask must specify an address range that is covered by one of the core's memory bars. Otherwise the core will not be selected by the AHB bus controller when the tmask area is accessed.	0 - 16#ff#	16#ff#

*Table 950.*Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
tbw_accsz	AMBA access size to use when fetching entries of the Access Protection Vector and/or the IOMMU page table. This value also sets the Access Protection Vector cache line size and the TLB entry size. This value must not exceed the maximum access size for the AHB master interface.	32 - *mstmaccsz*	32
dpagesz	Support for dynamic page size. If this generic is set to 1 the core will support selecting the page size via the Control register. If this generic is set to 0, the page size is fixed to 4 KiB.	0 - 1	0
ft	Fault tolerance. This setting determines if the APV cache and/or TLB tag and data RAMs should be protected against faults. Possible values are: 0 - disabled, 1 - byte parity	0 - 1	0
narb	Number of ASMP register blocks. The core will be implemented with narb ASMP register blocks. the required syshmask settings for different narb values are: narb 0 : hmask 0xfff, narb 1 : hmask 0xfe, narb 2 -3 : hmask 0xfc, narb 4-7 : hmask 0xf80, narb 8 - 15 : hmask 0xf00		
multiirq	Enable interrupt propagation for second AHB master interface. This generic is only available if the the entity griommu_mb is instantiated. If this generic is set to '1', interrupt propagation, as configured via the irqsync generic, will also be done for the second AHB master interface.	0 - 1	0

64.14 Signal descriptions

Table 951 shows the interface signals of the core (VHDL ports).

*Table 951.*Signal descriptions (VHDL ports)

Signal name	Field	Type	Function	Active
RST		Input	Reset	Low
HCLKSYS		Input	AHB system bus clock	-
HCLKIO		Input	AHB IO bus clock	-
IO_AHBSI	*	Input	AHB slave input signals	-
IO_AHBSO	*	Output	AHB slave output signals	-
IO_AHBPNP	*	Input	AHB master output vector signals (on io/slave i/f side). Used to decode plug'n'play vendor/device ID of masters so that these values can be visible in the Master control register(s).	-
SYS_AHBMI	*	Input	AHB master input signals	-
SYS_AHBMO	*	Output	AHB master output signals	-
SYS_AHBPNP	*	Input	AHB slave input vector signals (on system/master i/f side). Used to decode cachability and prefetchability Plug&Play information on bus connected to the bridge's master interface.	-
SYS_AHBMI2	*	Input	AHB master input signals, second interface. Only available on griommu_mb entity.	-
SYS_AHBMO2	*	Output	AHB master output signals, second interface. Only available on griommu_mb entity.	-
SYS_AHBPNP2	*	Input	AHB slave input vector signals (on system/master i/f side). Used to decode cachability and prefetchability Plug&Play information on bus connected to the bridge's second master interface. Only available on griommu_mb entity.	-
SYS_AHBSI	*	Input	AHB slave input signals	-
SYS_AHBSO	*	Output	AHB slave output signals	-
WLK_AHBMI	*	Input	AHB master input signals	-
WLK_AHBMO	*	Output	AHB master output signals	-
LCKI	slck blck mlck	Input	Used in systems with multiple AHB/AHB bridges (e.g. bi-directional AHB/AHB bridge) to detect deadlock conditions. Tie to "000" in systems with only uni-directional AHB/AHB bus.	High
LCKO	slck blck mlck	Output	Indicates possible deadlock condition	High
STATO (clocked by HCLKSYS)	hit	Output	High for one cycle during TLB/cache hit.	High
	miss	Output	High for one cycle during TLB/cache miss	High
	pass	Output	High for one cycle during passthrough access	High
	accok	Output	High for one cycle during access allowed	High
	accerr	Output	High for one cycle during access OK	High
	walk	Output	High while core is busy performing a table walk or accessing the access protection vector	High
	lookup	Output	High while core is performing cache lookup/ table walk	High
	perr	Output	High for one cycle when core detects a parity error in the APV cache	High

*Table 951.*Signal descriptions (VHDL ports)

Signal name	Field	Type	Function	Active
IFCTRL	mstifen	Input	Enable master interface. This input signal is unused if the VHDL generic *ifctrlen* is 0. If VHDL generic *ifctrlen* is 1 this signal must be set to '1' in order to enable the core's AMBA master interface, otherwise the master interface will always be idle and will not respond to stimuli on the core's AMBA slave interface.	High
	slvifen	Input	Enable slave interface. This input signal is unused if the VHDL generic *ifctrlen* is 0. If VHDL generic *ifctrlen* is 1 this signal must be set to '1' in order to enable the core's AMBA slave interface, otherwise the interface will always be ready and the bridge will not propagate stimuli on the core's AMBA slave interface to the core's AMBA master interface.	High

* see GRLIB IP Library User's Manual

64.15 Library dependencies

Table 952 shows the libraries used when instantiating the core (VHDL libraries).

*Table 952.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

64.16 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.iommu.all;

entity griommu_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    ...  -- other signals
    );
end;

architecture rtl of griommu_ex is

  -- AMBA signals, system bus
  signal proc_ahbsi       : ahb_slv_in_type;
  signal proc_ahbso       : ahb_slv_out_vector;
  signal proc_ahbmi       : ahb_mst_in_type;
  signal proc_ahbmo       : ahb_mst_out_vector;

  -- AMBA signals, IO bus
  signal io_ahbsi         : ahb_slv_in_type;
  signal io_ahbso         : ahb_slv_out_vector;
```

```
  signal io_ahbmi         : ahb_mst_in_type;
  signal io_ahbmo         : ahb_mst_out_vector;

  signal nolock           : griommu_ctrl_type;
  signal noifctrl         : griommu_ifctrl_type;
  signal dbgifctrl        : griommu_ifctrl_type;
  signal griommu_stato    : griommu_stat_type;

begin

nolock <= griommu_ctrl_none;
noifctrl <= griommu_ifctrl_none


-- Instantiate clock generators and AHBCTRL cores here
....
....
-- GRIOMMU
iommu: griommu
    generic map (
      memtech     => memtech,
      iohsindex   => 0,
      syshmindex  => 4,
      syshsindex  => 4,
      syshaddr    => 16#200#,
      syshmask    => 16#FFE#,
      syshirq     => 1,
      slv         => 0,
      dir         => 1,
      ffact       => 1,
      pfen        => 1,
      wburst      => 8,
      iburst      => 8,
      rburst      => 8,
      irqsync     => 0, -- No interrupt synchronization
      bar0        => ahb2ahb_membar(16#000#, '0', '0', 16#800#),
      bar1        => ahb2ahb_membar(16#800#, '0', '0', 16#800#),
      sbus        => 1,
      mbus        => 0,
      ioarea      => 16#FFF#,
      ibrsten     => 0,
      lckdac      => 0,
      slvmaccsz   => 32,  -- Maximum allowed access size by masters on io bus
      mstmaccsz   => 128, -- Maximum allowed access size on system bus
      rdcomb      => 2,
      wrcomb      => 2B,
      allbrst     => 0,
      ifctrlen    => 0,
      fcfs        => IO_NAHBM*CFG_IOMMU_FCFS,
      fcfsmtech   => 0,
      scantest    => scantest,
      split       => CFG_IOMMU_FCFS,
      nummst      => IO_NAHBM, -- Number of masters to support
      numgrp      => CFG_IOMMU_NUMGRP,
      stat        => CFG_IOMMU_STAT,
      apv         => CFG_IOMMU_APV,
      apv_accsz   => CFG_IOMMU_APVACCSZ,
      apvc_en     => CFG_IOMMU_APVCEN,
      apvc_ways   => 1,                  -- Only valid value
      apvc_lines  => CFG_IOMMU_APVCLINES,
      apvc_tech   => CFG_IOMMU_APVCTECH,
      apvc_gseta  => CFG_IOMMU_APVCGSETA,
      apvc_caddr  => CFG_IOMMU_APVCCADDR,
      apvc_cmask  => CFG_IOMMU_APVCCMASK,
      apvc_pipe   => CFG_IOMMU_APVCPIPE,
      iommu       => CFG_IOMMU_IOMMU,
      iommutype   => CFG_IOMMU_IOMMUTYPE,
      tlb_num     => CFG_IOMMU_TLBNUM,
      tlb_type    => CFG_IOMMU_TLBTYPE,
      tlb_tech    => CFG_IOMMU_TLBTECH,
      tlb_gseta   => CFG_IOMMU_TLBGSETA,
```

```
        tlb_pipe    => CFG_IOMMU_TLBPIPE,
        tmask       => 16#ff#,
        tbw_accsz   => CFG_IOMMU_TBWACCSZ,
        ft          => CFG_IOMMU_FT)
      port map (
        rstn        => rstn,
        hclksys     => clkm,
        hclkio      => clkm,
        io_ahbsi    => io_ahbsi,
        io_ahbso    => io_ahbso(0),
        io_ahbpnp   => io_ahbmo(IO_NAHBM-1 downto 0),
        sys_ahbmi   => sys_ahbmi,
        sys_ahbmo   => sys_ahbmo(4),
        sys_ahbpnp  => sys_ahbso,
        sys_ahbsi   => io_ahbsi,
        sys_ahbso   => io_ahbso(4),
        lcki        => nolock,
        lcko        => open,
        stato       => griommu_stato,
        ifctrl      => noifctrl);

end;
```

65 GRIOMMU2 - AHB/AHB bridge with RISC-V IOMMU & IOPMP capabilities

65.1 Overview

GRIOMMU2 is an AHB/AHB bridge that incorporates translation capabilities and physical memory protection based on the RISC-V IOMMU specification version 1.0 and RISC-V IOPMP specification version 0.8. It comes in two versions: the *standard* version with only a traditional AHB 2.0 interface and an *advanced* version with both a traditional AHB 2.0 interface and also an striped AHB interconnect optimized for higher throughput. GRIOMMU2 consists of five main blocks: the RISC-V IOMMU, the RISC-V IOPMP, the register interface, the I/O bridge and the striped controller (only included for the advanced variant of the IP). Using VHDL generics it is possible to individually configure if the core should include the IOMMU and the IOPMP or not.

DMA master AHB signals connect directly to GRIOMMU2, with arbitration handled internally by the core. This design enables the core to accept and queue multiple translations at once. Address translations adhere to the RISC-V IOMMU specification, utilizing structures such as Device ID, Process ID, and standard RISC-V MMU page tables (Sv39 and Sv32).

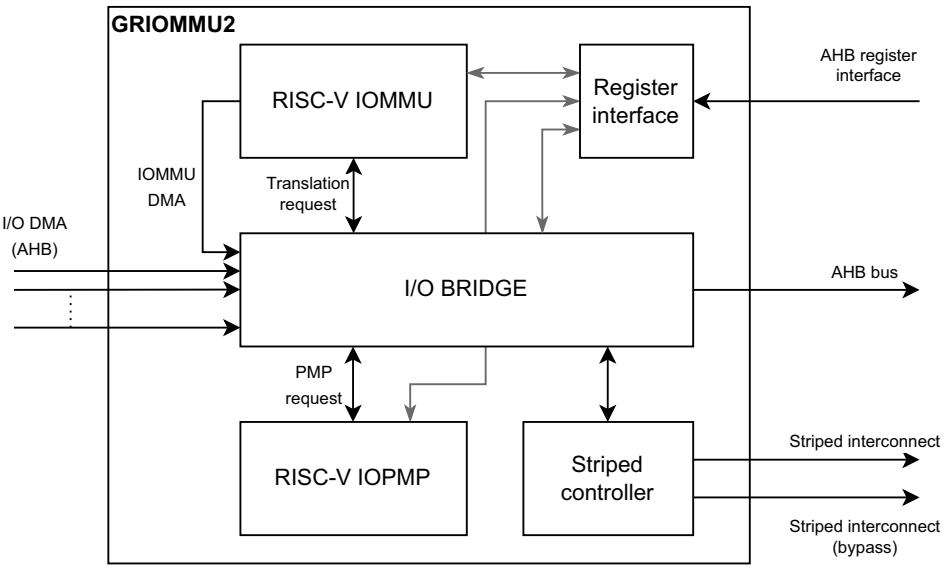


Figure 153. Block diagram of GRIOMMU2 using the higher throughput striped AHB interconnect bridge.

65.2 Bridge operation

65.2.1 General

The I/O bridge acts as the control hub of the GRIOMMU2 core. The main functionality of the block is to handle bridging between the connected DMA devices and the outgoing bus resources, i.e. the conventional AHB bus and, when the *advanced* variant of the IP is used, the striped buses. By utilizing the connected IOMMU and IOPMP components, advanced memory protection and memory visualization can also be provided by the I/O bridge.

For each connected DMA device there is a small state machine which handles all communication with the rest of the core as well as handling the AHB response back to the DMA device. It makes sure that the required protection is granted before requesting access to a bus resource, and once a bus resource has been granted, that data is sent and received properly

For the conventional AHB bus resource, there is a state machine which directly handles the communication with the outgoing conventional AHB bus.

For the striped interface the setup is slightly more complicated. That is because there is a FIFO buffer in between the controller state machine inside the I/O bridge and the actual AHB master controller for the stripe. The reason for this setup is to allow for easy clock domain crossing and pipelining in the striped interface since this interface is often timing critical in more complicated designs. The number of stripes included in the design is configurable with the *nstripes* VHDL generic.

In figure 154 a block diagram of the I/O bridge can be seen. The area marked in the dotted red box is logic only included in the *advanced* variant of the IP. The bus controller for the conventional AHB bus differs between the two variants, details about the differences can be read in chapter 65.2.4. The *crossbar* box in the figure does not represents any logic, only that every input is directly connected to every output to and from the block.

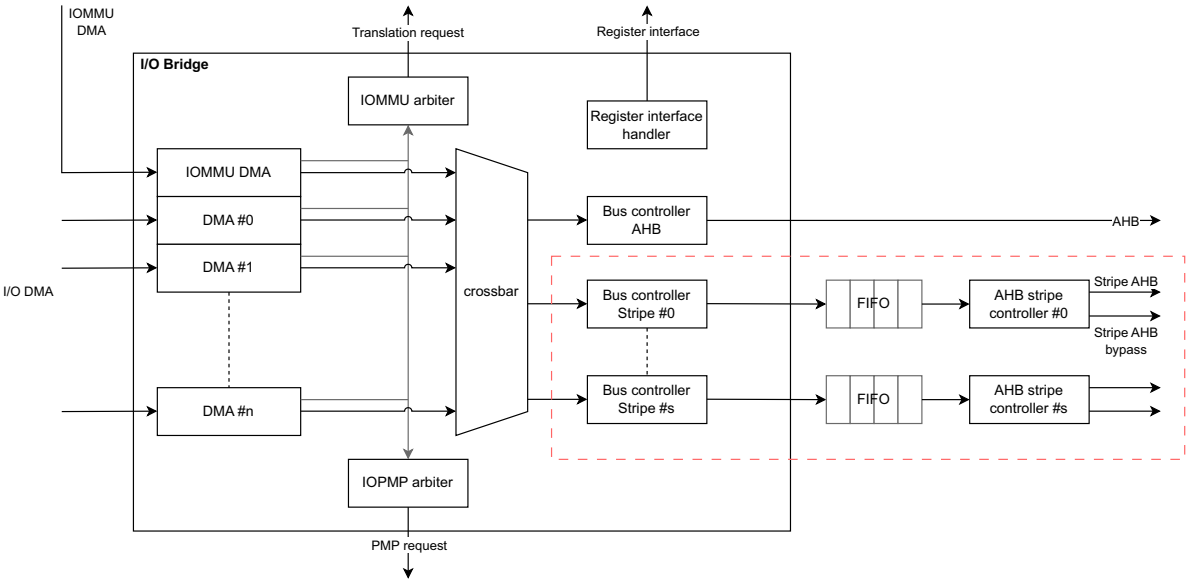


Figure 154. Block diagram of the GRIOMMU2 I/O Bridge. Area marked in red is logic exclusive for the *advanced* variant.

65.2.2 Access flow

The RISC-V IOMMU specification defines five operating modes: "OFF," "BARE," "LVL1," "LVL2," and "LVL3". The mode is set through the AHB register interface and is propagated through the RISCV IOMMU to the I/O bridge.

In "OFF" mode, all DMA accesses immediately return an error response. In "BARE" mode, no translations are performed, and the bridge uses the address provided by the DMA master directly. For the other modes, a translation request is sent to the IOMMU core. After the translation process has completed in the IOMMU core, an answer is sent back to the bridge which contains the outcome of the translation. If an error occurred during the translation process, or if the access had insufficient permissions, then an AHB error response is sent back to the original access. Otherwise, if the translation was successful, an translated address is returned.

If the IOPMP unit is enabled, an PMP check needs to happen before the access can continue. If the PMP checks fails an AHB error response is returned to the original access, otherwise it continues as normal.

The bridge then request access to the bus associated with the address retrieved from the previous step. For the *standard* version of the IP, there is only one bus available and all accesses will be sent there. For the *advanced* version there is five different bus resources that exists, the four buses associated with the striped bus and the conventional AHB bus. All accesses in the range *0x80000000 - 0xFFFFFFFF* is sent to the conventional bus. For addresses outside of this range, one of the striped buses will be selected. The stripe selection is configured using the stripe configuration signal.

Once a DMA device has been granted access to a bus, then the access can be propagated through that bus to the rest of the system. For burst accesses, the process of address translation, IOPMP check, and bus arbitration normally only happens once for the first access. Any burst that spans over a 64 byte address boundary however will be treated as a new request internally in the I/O bridge and will need to repeat the process of gaining access to a bus.

When the IOMMU is configured to not be included, then a operational mode of BARE is always sent to the I/O bridge. Likewise, when the IOPMP is configured not to be included, a value of "IOPMP not enabled" is always sent to the I/O bridge.

In figure 155, the flow diagram of the entire process from an access first being requested by a DMA device to the bridge requesting the use of a bus can be seen.

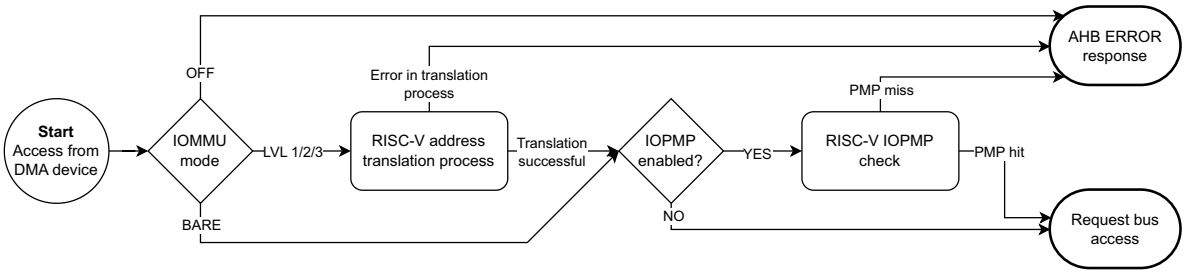


Figure 155. Flow diagram of how the I/O bridge handles an incoming DMA access.

65.2.3 Connecting DMA masters

The GRIOMMU2 supports connecting AHB master signals directly from the DMA devices instead of going through a AHBCTRL. The reason for connecting the signals directly is to allow for work to happen in parallel, as for example, while one master is accessing an outgoing bus, another master can be in the IOMMU address translation process. Additionally, in the *advanced version* of the IP using the striped bus, each stripe can be accessed independently allowing several DMA devices to do accesses with potentially no interference between each other.

Since it is a direct connection between the DMA devices and the GRIOMMU2, some AHB signals becomes unnecessary. The *hindex* signal is not used by the GRIOMMU2, when returning an answer to any master, the *hgrant* will always be set to all ones. The core will never either response with a *SPLIT* or *RETRY*, instead wait states with *hready* low is always issued while processing a access request.

In addition to the AHB signals, a *device ID (DID)*, *process ID (PID)*, and *process ID valid* signals needs to be provided for each connected DMA device through the *did, pid* and *pid_v* signals respectively. One set these signals are associated per DMA device, and these signals are in turn used by the RISC-V IOMMU translation process which is described in chapter 65.4.

The GRIOMMU2 allows for at most 62 direct AHB masters connected to it.

65.2.4 Access patterns

How accesses from the DMA devices are forwarded through the I/O bridge to the conventional and striped buses depends on the original access type and what bus the access is sent too.

For the *advanced* version of the IP, all accesses to the conventional bus will happen as single accesses of the same size as the original access. Accesses to the striped area is optimized for higher throughput and is designed to work together with a L2 cache using 64-byte cachelines. When a burst read is detected to this area, the entire 64-byte cacheline which the first access hits is prefetched as a burst using the highest allowed access size, the prefetched data is then used to serve the burst access. If the burst spans over several 64-byte cachlines, then this process is repeated for the following cachelines until the burst has finished. Burst writes to the striped bus will be absorbed and packed in the I/O bridge and be sent out as single accesses.

For the *standard* version of the IP, which only has one outgoing AHB bus, then all accesses are propagated as is, with the exception that bursts are split up into single accesses on the outgoing bus.

For all writes for both variants of the IP, the bridge utilizes the *posted writes* optimisation and always returns an AHB OK response immediately after access to a bus in the I/O bridge has been granted.

65.2.5 Error response from bridge

In the process of handling a access request from a DMA master there are many things that can go wrong which will cause the I/O bridge to return a AHB ERROR response. Depending on the type of access and where in the GRIOMMU2 the error happened, how the error is handled internally in the bridge will vary.

If the error happened before the DMA master requested access to one of the external buses, i.e. because of a fault in the IOMMU address translation, a IOPMP check failed, or the IOMMU mode is set to OFF, then a error response will always be sent. If the access was a burst, then the entire burst will get a error response.

If the error instead happened because of a access faults for a access propagated from the DMA masters, then the behavior will be different depending on what bus the access was sent to, and the type of the access. For **any write** access which caused an access fault, the error will always be masked by the bridge and an AHB OKAY response will be returned. The reason for this is because of the *posted writes* optimisation, mentioned in chapter 65.2.4. The response for **single reads** will always be propagated properly. The behavior of **burst reads** depends on which bus the access was to. For the conventional AHB bus, each individual access in the burst will get the proper response. For the striped bus, since it prefetches 64 bytes of data for every burst request, any error detected to that 64 byte area will result in all DMA accesses using that data getting a error response.

For accesses generated by the IOMMU core there is extra logic added in the I/O bridge so faults that were generated by writes will be properly recorded by the IOMMU core.

65.2.6 Bus arbitration

Access arbitration between DMA masters is handled within the I/O bridge. There are up to three separate places where arbitration is necessary depending on the current state of the IP. These are: requesting to send a address translation request, requesting an address to get a IOPMP check, and requesting access to a bus. In all three cases a round robin scheme is used to do the arbitration.

Each bus controller handles its own arbitration locally, meaning when the *advanced* variant of the IP is used, several DMA masters can have accesses going on in parallel without any interference as long as the accesses does not require use of the same bus. When the *standard* version of the IP is used, this scenario is not possible since there is only one outgoing bus.

65.2.7 IRQ forwarding

The I/O bridge possesses the capability to forward IRQ signals from the DMA devices to the AHB system I/O bus. In certain scenarios, a DMA device may wish to send an IRQ after a write operation has been completed. However, if the GRIOMMU2 were to immediately propagate these signals, there exists a possibility that the IRQ could become visible to the external system before the memory write due to the *posted writes* optimisation.

To mitigate this issue, the GRIOMMU2 has an option of enabling IRQ synchronization. This functionality ensures that when an IRQ is received from a DMA device, the I/O bridge latches the IRQ bits and initiates a fence operation that exclusively affects that DMA device. The IRQ is propagated only after it has been confirmed that all accesses for that DMA device have been fully handled. The fencing logic is only triggered on the rising edge of a interrupt, this is done so level interrupts also are forwarded as intended. During the fencing operation, any new accesses from the DMA device will be stalled.

65.2.8 Clock domain crossing

There are two clock inputs to the GRIOMMU2, the *ioclk* and *strclk*. The *ioclk* is used by the I/O bridge, IOMMU core and the IOPMP, while the *strclk* is only used by the striped bus controllers. To handle the clock domain crossing a FIFO buffer is used.

65.2.9 AMBA SPLIT support

GRIOMMU2 supports AMBA split and retry on the outgoing buses. The DMA side will never create a split or retry response since they can not block each other anyways.

65.3 IO Physical Memory Protection (IOPMP) functionality

65.3.1 General

In the GRIOMMU2, a RISC-V IOPMP unit can be included which provides extra physical memory protection (PMP). The configuration of the IOPMP, or if it should be included at all, can be configured using VHDL generics to the GRIOMMU2.

All resources used by the IOPMP is stored locally in registers, as oppose to the IOMMU, which utilizes external memory extensively.

A PMP request to the IOPMP includes the address that wants to be accessed, the permissions required (read, write and execute), and the Request Role ID (RRID) associated with the requester. Each DMA device connected to the GRIOMMU2 is associated with an RRID which is the same as the index it is connected on the *dmami/dmamo* vector. The answer that is sent back is if permission to do the access has been granted or not.

The address that is sent from the I/O bridge to the IOPMP has a granularity of 4KiB, meaning only the address bits in position 12 and above is actually used. Any access started by the I/O bridge to within a 4KiB area, requesting identical permissions, should get the same PMP answer back.

In a full IOPMP implementation there would be three separate internal data structures used for the PMP check: *SRCMD table*, *MDCFG table*, and the *PMP entry array*. What the IOPMP in the GRIOMMU2 instead implements is an simplification of this full model which only makes use of the *SRCMD table* and the *PMP entry array*. In the RISC-V IOPMP specification this setup is called *Rapid-K*.

In the *Rapid-K* implementation, the *PMP entry array* is split up into equally large *memory domains* of size K PMP entries. The *SRCMD table* contains, for each requester, a bit vector which says if a memory domain should be checked against or not. In the process of handling a PMP request, the first step is therefore always to fetch the memory domain bit vector from the *SRCMD table* which corresponds

with the requesting user so it is known which PMP entries should be matched against. In the figure 156 this process can be seen. The size of K can be configured with the *iopmp_k_n* VHDL generic, where K=2^*iopmp_k_n*, and the number of supported *memory domains* can be configured with the VHDL generic *iopmp_num_mds*.

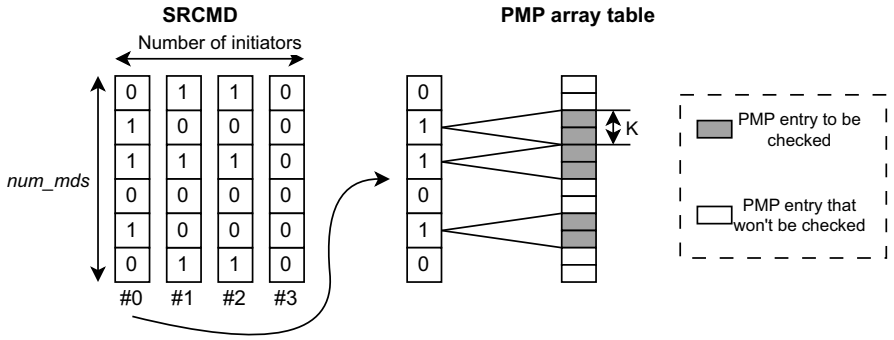


Figure 156. Process of selecting which PMP entries are valid depending on SRCMD table for a requester with id #0.

The PMP entries can either be of type *priority* or *non-priority*, with all *priority* entries being located in the beginning of *PMP entry array*. The PMP entries of the *PMP entry array* is matched against the request in order. If the address of the request matches a *priority entry*, then the process stops and the request is either granted or not granted permission depending on the permissions given in the matched PMP entry. For *non-priority entries*, permission can only be granted and not removed. Which means that an address match to a *non-priority entry* only halts the matching logic if it also actually grants permission for the request, otherwise it continues until there is no more entries to match against. More details about the matching logic can be read in the RISC-V IOPMP specification.

Any access generated by the IOMMU core will also be subject to PMP checks if the IOPMP is enabled. The IOMMU has a RRID value of *ndmamst*.

65.3.2 PMP entry Chunks

In a IOPMP implementation, large amounts of PMP entries may be needed. To be able to have an implementation that scales well with many entries, the core implements a system where the *PMP entry array* is split up into chunks and are processed one by one. The size of the chunks is set with the *iopmp_chunk_s* VHDL generic. While it is not strictly needed, it is recommended that the chunk size is set to evenly divide the total number of PMP entries in the system.

A chunk can either be defined as containing only *priority entries* or only *non-priority entries*. How many of the chunks should be *priority chunks* is set with the VHDL generic *iopmp_p_chunks*.

In the matching logic of the IOPMP, one chunk will be checked against each cycle. Once there is a match and access has been granted, of if there is a match in a *non-priority entry* which restricts permission, the matching logic will stop and any chunks not yet checked will be skipped. The number of clock cycles this matching logic will take therefore depends on how many chunks needs to be checked before termination.

That the PMP matching is handled with this chunk system should be totally transparent to a user configuring the IOPMP. The only noticeable difference from outside the core should be the execution time to handle a PMP request which varies depending on the chunk hit.

65.3.3 PMP entry cache

To speed up the PMP checking logic, a PMP cache is included in the core. After a PMP request has been granted permission to do an access, then the information from that PMP request is saved in the

cache. To get a hit in the cache the RRID, all the permissions, and all of the address bits needs to match exactly.

For every new PMP request, the cache is always checked first. Only if there is a miss in the cache will the request be sent to the main PMP checker. A response can be sent back from both the PMP cache and the main PMP checker at the same cycle.

These requirements to get a hit in the cache are very strict, in practice the main benefit of the cache is to speed up the PMP request for a device which is streaming large amounts of data sequentially. Then only a PMP request needs to be fully processed once for every 4KiB area, instead of the start of every access.

Every time there is a write to any part of the *PMP entry array* or *SRCMD table*, the entire PMP cache is invalidated.

65.4 IO Memory Management Unit (IOMMU) functionality

65.4.1 General

The IOMMU core is implemented following the RISC-V IOMMU 1.0 specification and provides features which can be used to implement virtual addresses and memory protection for accesses from external DMA Devices. A block diagram of the IOMMU core can be seen in figure 157.

The IOMMU core interacts with the I/O bridge through two interfaces, the translation request interface and a DMA interface.

The translation request interface passes a virtual address together with a Device ID, Process ID and other access related information to the IOMMU core in order for it to translate the virtual address into a physical address.

The IOMMU core uses the DMA interface to access memory, where many of the structures that it works with reside. The DMA interface is handled as an ordinary I/O DMA master with the exception that accesses originating from it never loop back into the IOMMU.

Once an address translation request is received by the IOMMU core, the access will move through a three stage translation lookaside buffer (TLB) checker to see if the translation has been cached. If the TLB checker finds a full match, it will return the translated request and no table walks will be generated. If the TLB checker misses in one or more stages, the entry will be put into a circular buffer which will translate and respond to the request in order. TLB lookups are made during page walking as well, so partial hits will still speed up the translation process significantly.

The main state machine within the IOMMU core monitors the circular buffer and translate the entries in order. During the translation, faults can occur. This triggers an error response to the IOMMU bridge and, if the fault-queue is enabled, writes a fault-queue entry to main memory. Similarly, an in-memory command-queue can be used to send commands to the IOMMU.

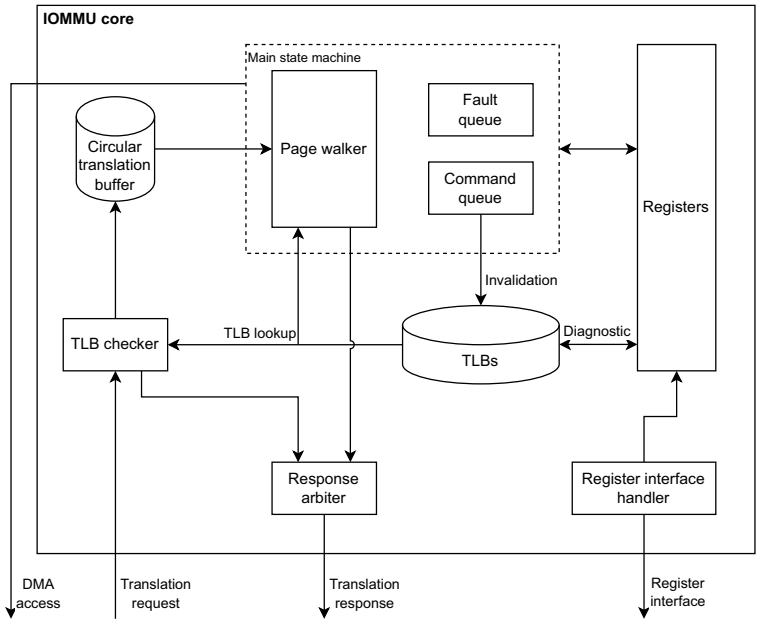


Figure 157. GRIOMMU2 RISC-V IOMMU block diagram

65.4.2 Translation interface

A translation interface is utilized by the I/O bridge to request translations from the IOMMU core. One translation request can at most be received and processed per cycle. If the circular buffer is full, the IOMMU core signals to the I/O bridge that it can receive no more translation requests.

The translation request from the I/O bridge to the IOMMU core contains the following information: *DMA master index, device ID, process ID, process ID valid, address, write data, privilege request, execute intent, write request, and size of access*. The translation response from the IOMMU core to the I/O bridge then contains the following information: *DMA master index, translated address response, address response mask, striped bypass interface to be used, signal error current transaction, and signal abort current transaction*.

There are two places in the IOMMU core which can generate address translation responses, the three stage TLB checker and the main state machine page walker. Only one answer can be sent back to the I/O bridge at a time. In case of a conflict, the TLB checker always takes priority and the page walker stalls.

65.4.3 DMA interface

For much of its functionality, the IOMMU core requires the capability to read and write to memory. This is handled through a DMA interface which sends generic requests for accessing memory. In the I/O bridge these requests are then transformed to the AHB format and are treated as any other DMA device with the exception that accesses from the IOMMU core never require address translation. Appropriate access size and burst mode are used based on the data being fetched.

As an optimisation, after a memory read, instead of only being able to fetch 32b at a time, like any other DMA device, the IOMMU DMA interface can immediately transmit the entire internal 512b data buffer used by the bus controllers.

65.4.4 Translation structure

The translation process involves three components: Device ID (DID), Process ID (PID), and Virtual Address (VA). Initially, the IOMMU utilizes the DID and a base address specified by the *ddtp.ppn* register to do a page walk on the device-directory-table (DDT) in order to fetch a device context (DC) from memory. The DC contains information on how to handle the translation in subsequent steps.

If specified by the DC, a similar process is followed to perform a page walk on the process-directory-table (PDT), retrieving a process context (PC) using the PID and a base address specified in the DC. Additionally, if enabled in the DC, the PC decoding undergoes a guest translation step, enabling a virtual machine to establish its own PID decoding structure. More information about the DDT and PDT page walks can be found in the RISC-V IOMMU specification.

Once the decoding process of the DC and, if applicable, the PC is complete, the VA is used to perform a lookup for the corresponding physical address (PA). This lookup is carried out using the structures described in the RISC-V Privileged Specification, with GRIOMMU2 supporting either Sv32 or Sv39 formats for address translation.

The RISC-V address translation is carried out in two main steps, *first stage address translation* (S-stage), and *guest stage address translation* (G-stage). If both steps is utilized, then the translation process is refereed to as *two-stage address translation* (SV-stage). Each step consist of a at most three, or two level page walk depending on Sv39 or Sv32 is used respectively. When a *page table entry (PTE) leaf* is found, the page walk is completed. The size covered by the PTE depends on which level in the page walk the *leaf* is found. In Sv39 the three possible sizes are: 1 GiB, 2 MiB, and 4KiB. TLB pages larger than 4KiB are collectively called *superpages*. During the SV-stage page walk, when both kinds of translations is enabled, each lookup in the S-stage needs to go through a G-stage translation. In figure 158 a diagram of a two-stage Sv39 address translation is shown. The *hypervisor guest address translation and protection (*HGATP) value controls the G-stage translation and is found in the DC, the *supervisor address translation and protection* (SATP) value controls the S-stage translation and is found in either the DC or, if PDT walk is enabled, the PC.

In the RISC-V IOMMU specification, one additional translation step is mentioned, that is the *messaged signal interrupt (MSI) address translation*. After the SV/S-stage translation and before the last G-stage translation, if MSI translation is enabled as specified in the DC, a check is done using a *address pattern* and *address mask* also found in the DC to see if the current address matches that of a virtual interrupt file. If a match is detected, then the last G-stage step will be skipped and instead a lookup happens to a MSI page table using the MSI page table root pointer found in the DC and a interrupt file number extracted from the current address. The MSI page table is always only made out of one level, therefor the page walk only requires one memory access.

The MSI page table lookup only happen if the original access was a 32b write. If the access is not of that type, then the MSI translation step is canceled and a *abort* signal is sent to the I/O bridge. The I/O bridge will then send a AHB OK response back to the DMA master associated with the access. If the access was a read the returned data will be all zeroes.

In the MSI PTE, two different kinds of modes can be specified: *basic MSI translation*, and *memory-resident interrupt file (mrif)*. For a MSI PTE which is set to *basic MSI translation mode*, the PTE contains a 4KiB PPN which is directly returned as the translated address to the I/O bridge. When the mode is set to *mrif mode*, the IOMMU first generates an atomic read-modify-write to the destination MRIF file, which location is specified in the MSI PTE. After the access has finished, a notice MSI is sent out as a 32b write, the destination of the write is also specified in the MSI PTE. When both of theses actions have happened, a *abort* signal is sent to the I/O bridge, canceling the access from the DMA.

More details about the MSI translation can be found in the RISC-V IOMMU and AIA specifications.

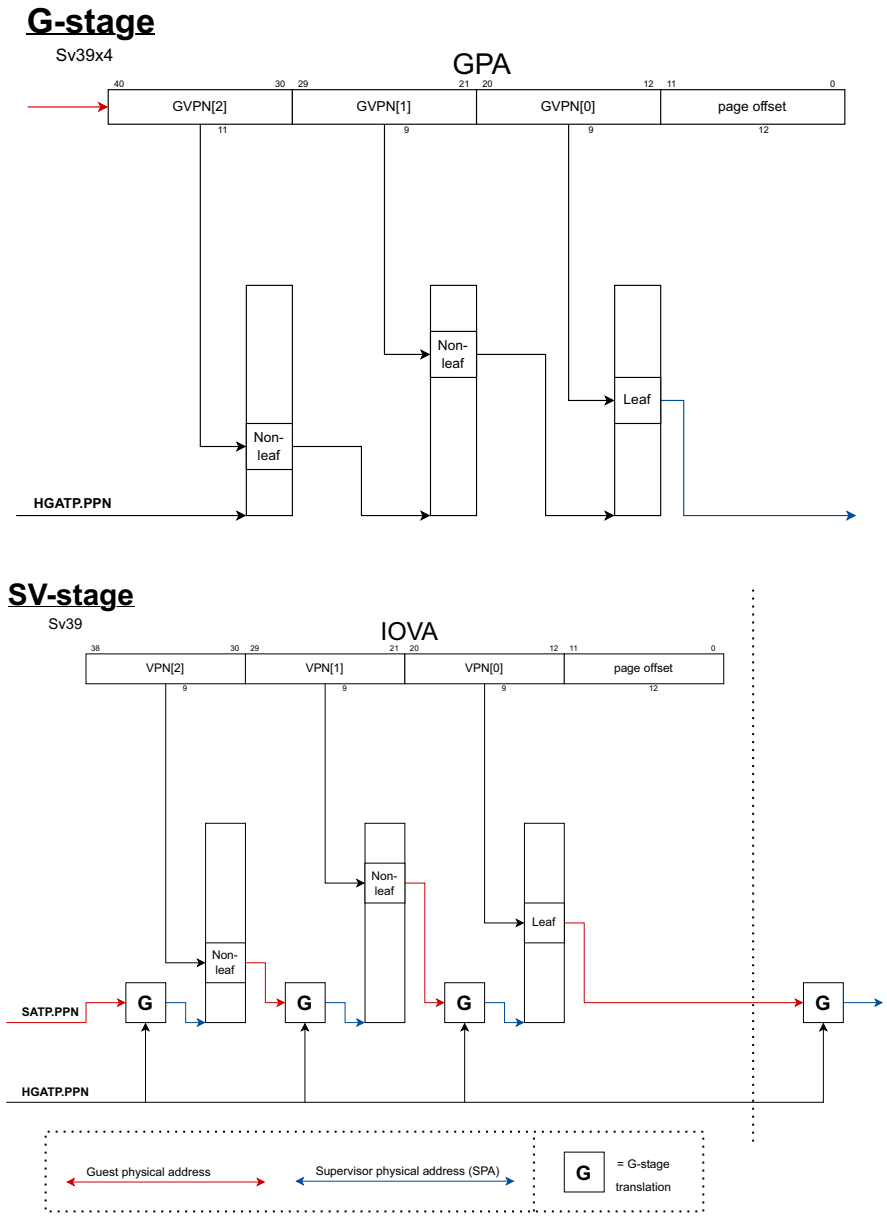


Figure 158. RISC-V two-stage address translation using Sv39

65.4.5 Translation lookaside buffers

The GRIOMMU2 makes us of translation lookaside buffers (TLB) to cache the result from the various page walks. The TLB sizes are determined using VHDL generics, providing flexibility in configuring the number of entries. Each TLB can be enabled and disabled separately through the *fctl* register. The replacement policy utilized is pseudo LRU, which selects the least recently used entry for eviction. All of the TLBs implemented cache the entirety of the respective page walks per TLB entry, and not per individual memory lookup.

The GRIOMMU2 has four separate TLBs implemented:

Context-TLB:

The context-TLB serves as a combined cache for the results of DDT and PDT page walks.

When an entry in the TLB does not have a valid PID, it is disregarded in the matching logic, assuming that the translation request also lacks a valid PID. The relevant information obtained from the DID and PID decoding is stored in the TLB, and if a hit occurs, this information is utilized in subsequent decoding steps.

SV-TLB:

The SV-TLB serves as a cache for the outcomes of both S-stage and VS-stage translations. Looking at figure 158, the SV-TLB caches everything up until the last G-stage translation, or the dotted horizontal line under the SV-stage sub-figure.

The TLB stores information related to the virtual address, PSCID, and GSCID, provided there is a valid GSCID available. During the matching process, the SV TLB compares these parameters against the incoming virtual address to determine if there is a hit. PSCID and GSCID are obtained as part of the DC and PC decoding steps.

G-TLB:

The G-TLB caches the results of only the guest stage translations. Looking at figure 158, what is cached is the entirety of the G-stage translation block.

Some G-stage lookups are implicitly covered by both the context-TLB and SV-TLB, since during the process of a PDT or SV walk, G-stage address translations can be active but it will be covered by the respective TLB. However, if there is a miss in these TLBs, then the G-TLB will be used for each of the separate G-stage lookups. Additionally, the last G-stage lookup in the two stage address translation is not covered by any other TLB than the G-TLB.

The G-TLB matches the GSCID and the GPA during the translation process. The GSCID is obtained during Device decoding, and the GPA is determined during the page walk.

MSI-TLB:

The MSI TLB caches the results of accesses to obtain a MSI-PTE used in the process to do MSI translation. The TLB matches against the GPA obtained in the two-stage address translation step together with the GSCID found in the DC. The MSI translation step only ever requires one access to happen, this means the potential performance gains from the MSI-TLB is in many cases much smaller than from the other TLB types.

TLB Invalidation:

TLB invalidation can be performed either through the command-queue using invalidations commands as specified by the RISC-V IOMMU specification, or using the TLB diagnostics interface which is accessible through the AHB register interface. Additionally, when a TLB is turned off through the *fctl* registers then all TLB entries belonging to that TLB type is invalidated.

65.4.6 Diagnostics interface

It is possible to probe or change the contents of the four TLBs through a diagnostic interface. For each of the four TLBs, a register area is set up where one entire TLB entry can be read, containing all the information that is cached. To set which specific TLB entry is being looked at, a different index register is used.

Each TLB entry in the diagnostic interface has a *locked* bit which can only be set through the diagnostic interface. When the locked bit is set, the TLB entry will never be overwritten by the IOMMU in normal address translation operations. If the locked entries should be affected by invalidation commands is determined by the *fctl.invl* bit. When the bit is set, a invalidation command will reset both

the valid and locking bit of the TLB entry. If the bit is not set, then the locked TLB entry will be left untouched.

65.4.7 Fault-queue

The reporting of faults during the IOMMU address translation process is handled through an in-memory queue data structure called the fault-queue. For each fault detected a 32 byte fault record is created and stored in memory. The fault records are stored in a circular buffer which has its base address and size as determined by the *fault-queue base (fqb)* register. The tail of the queue can be seen in the *fault-queue tail (fqt)* register and it indicates the index in the buffer where the next fault record will be written by the IOMMU. The head of the queue is set in the *fault-queue head (fqh)* register and indicates which fault record that software should process next. After a record has been processed, the head register should be advanced by writing to it. When the fault-queue is full, an overflow bit is set and no more entries will be written until it has been reset.

In the 32 byte fault record, information about what caused the fault can be read. Some noteworthy fields that are stored here are: a twelve bit *cause* encoding, the DID, the PID and the address of the translation request that triggered the fault. It is possible to enable the fault-queue to send interrupts. If enabled, an interrupt will be sent after a fault record has been written to memory or when the fault-queue has overflowed.

For more information about the fault-queue, see the RISC-V IOMMU specification.

65.4.8 Command-queue

With a similar structure as the fault-queue, the command-queue serves as a memory-based queue where processors can write commands to be executed by the IOMMU. Each command entry occupies 16 bytes of memory.

The base address and size of the command-queue are determined by a *command-queue base (cqb)* register, specifying the starting location and total size of the queue in memory. Additionally, the command-queue has a head and tail pointer. The *command-queue tail (cqt)* register indicates the location in memory from where the IOMMU should read the next command for execution. Processors update the *cqt* register after writing a command to memory, to notify the IOMMU of the new command. The command is signaled to be accepted by the IOMMU once the *command-queue head (cqh)* has been incremented by the IOMMU. Commands are always executed the order they are submitted.

There are four kinds of commands described in the RISC-V IOMMU specification, three of which is supported by the GRIOMMU2. These are: *IOMMU page-table cache invalidation commands (IOTINVAL)*, *IOMMU command-queue fence commands (IOFENCE)*, and *IOMMU directory cache invalidation commands (IOTDIR)*. The unsupported command is the *IOMMU PCIe ATS commnds (ATS)*, which was omitted since the GRIOMMU2 does not support any of PCIe specific features mentioned in the specification.

The *IOTINVAL* and *IOTDIR* commands are used to invalidate the TLBs. Where the *IOTINVAL* command can be configured to invalidate entries in either of SV-TLB or G-TLB, or both at the same time. The IOTDIR command is used to invalidate entries in the CTX-TLB.

The *IOFENCE* command is used to issue a fence instruction to the GRIOMMU2. Once a read or write fence instruction has been issued, the I/O bridge will stall all new accesses of that type from the DMA masters, giving wait states with the AHB HREADY signal low. The I/O bridge signals at all times to the IOMMU core if it currently processing any read or write accesses. Once the IOMMU core has been informed that all accesses of the type being fenced is handled, then it signals that the fence is completed and the I/O bridge is free to accept new accesses again. The signals between the IOMMU core and I/O bridge are pipelined, so to avoid certain race conditions with the fencing logic, a minimum time of four cycles needs to pass before the fencing operation is allowed to finish. A *IOFENCE* command can be configured to send a wired interrupt and/or a memory write at completion. That the

command has been accepted by the IOMMU with the increment of the *cqh* does not mean the fence is done, only that it has been started.

For more information about the command-queue, see the RISC-V IOMMU specification

There exist a custom command registers in the GRIOMMU2 which allows commands to also be written directly to the register interface called *cmd_reg*. To trigger the IOMMU to run the command stored in the register interface, the *cqcsr.n_reg_cmd* bit needs to be set. Once the command has been accepted, the register bit will be reset.

65.4.9 Interrupts

There are two different kinds of interrupts that can be generated by the IOMMU as specified by the RISC-V IOMMU specification, *wire-signaled interrupts (WSI)* and *message-signaled interrupts (MSI).* The GRIOMMU2 has support for both.

A WSI is communicated to the rest of the system by toggling a interrupt signal high for a cycle. While a MSI is communicated by sending a 32 bit write using a address and data as configured in the IOMMU. The main purpose of having MSI is to be able to use them with a RISC-V *incoming message-signaled interrupt controller (IMSIC)*, but, as the MSI is simply a normal 32b write, it can be configured in whatever sense the user sees fit.

Interrupts can be configured to be generated in various scenarios, including after faults have been reported, when a fence command has been completed, or when an Hardware Performance Monitoring counter has reached its overflow condition.

How the interrupt should be forwarded to the rest of the system depends on how the IOMMU is configured. The register bit *fctl.wsi* is used as a global toggle to pick between if a WSI or MSI should be generated. Each source which can generate an interrupt has some kind of interrupt enable bit, for example, the fault-queue has the *fqcsr.fqi* register bit. It is possible to map faults from different sources too different locations using the *interrupt-cause-to-vector (icvec)* register. For a MSI, depending on how the *icvec* is configured, different entries in the *MSI configuration table (msi_cfg_tbl)* register will be used when choosing what address and data the MSI should make use of. When a WSI is triggered, there is two signals which will be affected. The outgoing *irq* vector, where the *icvec* is used to pick which bit on the vector to toggle, and the *ahbso.hirq* vector, which always toggles the bit with the index as determined by the *hirq* VHDL generic.

65.4.10 Hardware Performance Monitor

The GRIOMMU2 implements hardware performance monitor (HPM) registers used to count the number of occurrences of certain events as specified by the RISC-V IOMMU specification. The number of counters implemented and the width of the counters is specified by the *hpm* and *hpm_ctr_width* VHDL generic respectively.

For each *performance-monitoring event counter (iophmctr1-31)* implemented, there is an associated *performance-monitoring event selectors (iohpmevt1-31)* registers which is used to configure the trigger condition. What is configurable is: the event-id out of a predefined list of events, DID/GSCID filtering, and PID/PSICID filtering.

There are eight events described in the specification. How they behave on the GRIOMMU2 is as follows:

- **Do not count (0):** Never increment counter
- **Untranslated requests (1):** Increment counter every time a translation request has been fully processed. Both translation requests from the I/O Bridge and translation-request debug interface are counted.
- **Translated requests (2):** Not used. No way of using translated requests with the GRIOMMU2.

- **ATS Translation requests (3):** Not used. PCIe related feature are not implemented.
- **TLB miss (4):** Increment counter when the three stage TLB checker is missed and the request is sent to the IOMMU core state machine. If the TLB checker is disabled via the *fctl.bpen* register bit, then the counter will always incremented for incoming translation requests. Due to how the event is implemented, the *idt* setting of '1', where *GSCID* and *PSCID* filtering is used, is not supported.
- **Device Directory Walks (5):** Increment counter for every memory access request in the DDT page walk (counter is not incremented if there is a hit in the CTX-TLB).
- **Process Directory Walks (6):** Increment counter for every memory access request in the PDT page walk (counter is not incremented if there is a hit in the CTX-TLB).
- **First-stage Page Table Walks (7):** Increment counter for every memory access request in the S/SV-stage page walk (counter is not incremented if there is a hit in the SV-TLB).
- **Second-stage Page Table Walks (8):** Increment counter for every memory access request in the G-stage page walk (counter is not incremented if there is a hit in the G-TLB).

65.4.11 Bridge HPM

In addition to the default RISC-V IOMMU HPM counters, custom *bridge HPM* counters can be enabled with the *b_hpm* VHDL generic to count events in the I/O bridge. There are five 64b counters defined, each locked to a certain event.

The events are:

- **Read:** Increment counter when the state machine associated with a DMA master has been granted access to a bus with the intent of doing a read request. The counter is only incremented once at the start of a access, single or burst.
- **Write:** Increment counter when the state machine associated with a DMA master has been granted access to a bus with the intent of doing a write request. The counter is only incremented once at the start of a access, single or burst.
- **L2C bypass:** Increment counter when the state machine associated with a DMA master has been granted access to a bus and the access is configured to use the secondary bus interface (L2C bypassed). If the *advanced* version of the IP is used and the access is to the conventional AHB bus, the counter is never incremented. The counter is only incremented once at the start of a access, single or burst.
- **Error:** Increment counter when the state machine associated with a DMA master has gotten into the error state. This can happen e.g. if the IOMMU core responds that the address translation failed, or if there was an access fault for a read to memory. The error counter is only incremented once per DMA bus request. When the counter is incremented, that means that there has at least been one AHB ERROR response sent to a DMA master, and more could have been sent if the DMA bus request was a burst. More information about the error response behavior can be read in chapter 65.2.5.
- **Abort:** Increment counter when the state machine associated with a DMA master has gotten a *abort* transaction signal from the IOMMU core. Information about when *abort* signals are sent can be read in chapter 65.4.4.

Each counter has a DMA index mask, allowing the user to filter which DMA masters are sampled. All DMA sources can be counted at the same time if the mask is set to all ones.

The read and write counters also have an address filter and mask to allow the user to filter on address. The filter is applied after the address translation step.

65.4.12 Translation-request debug interface

In the IOMMU core there is a translation-request debug interface which can be used to probe the address translation results for a given address. It is defined using three registers: the *Translation-request IOVA (tr_req_iova)* register, which is used to set the VA to be translated, the *Translation-request control (tr_req_ctl)* register, which is used to set the PID, DID, and the requested permissions for the access, and lastly the *Translation-response (tr_response)* register, where the answer of the translation request is stored. The translation-request debug interface is only enabled if the *dbg* VHDL generic is set.

Accesses generated by the translation-request debug interface will be placed directly into the circular translation buffer, skipping the initial TLB checker. While being processed in the page walker, the translation request is treated in the same way as a translation request from the I/O bridge with two exceptions: if the translation is determined to be that of a *virtual interrupt file* (see chapter 65.4.4) a fault will occur and the translation stop, and new TLB entries discovered as part of the translation will only be cached if the *tr_req_ctl.fcache* register bit is set. Other behaviors, such as updating of the Accessed (A) and Dirty (D) PTE bits, incrementation of HPM counters, and the usage of the TLB will still happen.

If the result from the address translation is a *superpage* (i.e. a page of size larger than 4 KiB), then the *tr_response.s* bit will be set and a mask can be read out from the *tr_response.ppn* by counting the sequence of ones starting from the first bit position.

More information about the translation-request debug interface can be read in the RISC-V IOMMU specification.

65.4.13 Omitted/optional features

The GRIOMMU2 IP does not implement certain optional features described in the RISC-V IOMMU specification. These features include: any PCIe-related functionalities, runtime Bi-endian support (*capabilities.end=0)*, the svpbmt extension, or Sv48/Sv57 virtual memory systems.

The *fctl.gxl* register bit, which defines the supported address-translation scheme for guest physical address translation, is defined to not be writable in the GRIOMMU2. Instead it has a hardcoded value depending if the IOMMU core is set to either Sv32 or Sv39 with the *Sv32* VHDL generic. The consequence of this is that the GRIOMMU2 is not reconfigurable and can therefor not change between Sv32 and Sv39 mode, a choice is made at design time where only one can be picked. There is not either the possibility of using Sv39 for G-stage and Sv32 for S-stage, which would otherwise be possible.

65.5 Registers

The core is programmed through registers mapped into AHB I/O address space. Each of the three sub-blocks, *I/O Bridge, IOMMU,* and *IOPMP,* has their own AHB BAR which contains the relevant registers for respective block. A register interface state machine exists in the GRIOMMU2 which handles the register access requests between the AHB interface and the three sub-blocks. When either the IOMMU or IOPMP is configured not to be included, their respective AHB BAR is not included either.

To the register interface, accesses of 32b are allowed to the entire memory region while 64b accesses are only allowed to registers explicitly specified to be 64b in size. 64b accesses always need to use aligned addresses for the access to be allowed. Accesses that are not allowed receive a AHB ERROR response.

The VHDL generic *be_reg_order* enables a register mode which makes it so 32b accesses to 64b registers depend on endianness. That is, when a 32b access is received at the address of a 64b register, the returned value depends on the endianness of the system, determining whether the upper or lower 32b of the register will be returned. This mode goes against the RISC-V IOMMU specification, where instead 64b registers should always be treated as little endian in the case for 32b accesses.

65.5.1 I/O Bridge registers

*Table 953.*I/O Bridge registers

AHB address offset	Register	Not implemented when
0x0000	Capabilities register	-
0x0004	Control register	-
0x0100 - 0x01FC	Device ID area	-
0x0200 - 0x02AC	Bridge HPM	If *B_HPM* generic is 0
0x1000 - 0x17FC	HCONFIG area	-

65.5.2 Capabilities register

*Table 954.*0x0000 - CAP - Capabilities register

31 16	15 8	7 5	4	3	2	1	0
RESERVED	NDMAMST	RESERVED	BHPM	END	BREG	HPMP	HMMU
0	*	0	*	-	*	*	*
r	r	r	r	r	r	r	r

31:16	RESERVED
15:8	Number DMA masters connected (NDMAMST): Shows the total number of DMA masters connected the core has been configured to support with the *ndmamst* VHDL generic.
7:5	RESERVED
4	Has Bridge HPM functionality (BHPM) - The core includes the Bridge-HPM register. Mirrors the value of the *b_hpm* VHDL generic.
3	Current endianess (END) - Endianness used by the core. When '0' big endian is used, when '1' little endian is used. The value of *EN* is directly set by the *cpumi.endian* signal.
2	Big endian registers supported (BREG) - When set 32b accesses to 64b registers will respect endianess. Mirrors the value of the *be_reg_order* VHDL generic.
1	Has IOPMP (HPMP) - The core includes the IOPMP block. Mirrors the value of the *iopmp_included* VHDL generic.
0	Has IOMMU (HMMU) - The core includes the IOMMU block. Mirrors the value of the *iommu_included* VHDL generic.

Reset value: Implementation dependent

65.5.3 Control register

*Table 955.*0x0004 - CTRL - Control register

31 1	1	0
RESERVED	irqfFE	irqfEN
0	1	1
r	rw	rw

31:2	RESERVED
1	IRQ forwarding fencing enabled (IRQFFE) - Enables the IRQ forwarding to fence the interrupt until all transactions by the DMA device is done before forwarding the interrupt. This setting will stop all incoming transactions as well until the interrupt is handled.
0	IRQ forwarding enabled (IRQFEN) - Enable IRQ forwarding for interrupt from the DMA AHB signals.

Reset value: 0x00000003

65.5.4 Device ID area

Each DMA device has a 23b device ID (DID) associated with it. The device ID is set with the input signal *DID,* which is an array of length *ndmamst* containing these DID values. The *Device ID area* in the register interface shows what DID is associated with each DMA. The DIDs are in ascending order

with regards to which DMA they are associated with in the register interface (bits [7:2] of the address shows the corresponding DMA index).

Table 956. 0x0100 - 0x01FC - DID - Device ID area

31 24	23 0
RESERVED	DID
0	-
r	r

31:24 RESERVED

23:0 Device ID value (DID) -

Reset value: Implementation dependent

65.5.5 Bridge hardware performance monitor

There are five events counted by the bridge HPM, each having its own 64b counter register and one or more 64b filter registers. The register layout is described in table 957.

Table 957. 0x0200 + n*0x20 - BHPM - Bridge hardware performance monitor register layout

63 0	Byte offsett
Counter	+000h
DMA mask	+008h
Address filter	+010h
Address filter mask	+018h

The five available events and the register address for each event are: *Read access* (0x0200), *write access* (0x0220), L2C *bypass* (0x0260), *error* (0x0280) and *abort* (0x02a0). The events are described in chapter 65.4.11. Only the *read access* and *write access* events have the address filter and mask registers defined, for the other events these areas are reserved

Table 958. +0x00 - BHPM counter

63 0
CNT
0x0
rw

63:0 Counter (CNT) - Counts number of times an event has been triggered. Only events triggered by DMA masters masked by the *DMA mask* register as well which matches the address filter defined by the *Address filter* and *Address filter mask* registers are counted. Bit 63 of the BHPM counter is sticky and will not reset once the counter overflows.

Reset value: 0x0

Table 959. +0x08 - BHPM DMA mask

63 0
MSK
0x0
rw

63:0 BHPM counter DMA mask (MSK) - Every bit in the *BHPM DMA mask* corresponds to one DMA device connected to the GRIOMMU2. When the bit corresponding to a device is set, then read accesses from that device will be counted in the corresponding *BHPM counter*. The IOMMU core itself is also a master on the IOMMU bridge and has index *ndmamst*. Bits [63:*ndmamst*+1] are read only, with a value of zero.

Reset value: 0x0

Table 960. +0x10 - BHPM address filter

63	2	1	0
FLTR		RESERVED	
0x0		0x0	
rw		0	

63:2 BHPM counter address filter (FLTR) - Holds a 4-byte aligned 64b address used as a filter for the *BHPM counter*. If the event associated with the *BHPM counter* is triggered, the counter will only increment if the following condition holds:

addr[31:2] and *BHPM address filter mask* = *BHPM address filter*

Bits [63:paddrw] are not writable

1:0 RESERVED

Reset value: 0x0

Table 961. +0x18 - BHPM address filter mask

63	2	1	0
FMSK		RESERVED	
0x0		0x0	
rw		0	

63:2 BHPM counter address filter mask (FMSK) - Holds a 4-byte aligned 64b address mask used as a filter for the *BHPM counter*. If the event associated with the *BHPM counter* is triggered, the counter will only increment if the following condition holds:

addr[31:2] and *BHPM address filter mask* = *BHPM address filter*

Bits [63:paddrw] are not writable

1:0 RESERVED

Reset value: 0x0

65.5.6 HCONFIG area

Each DMA device has a 8x32 bit HCONFIG associated with its AHB master signal. The HCONFIG vector contains information about the DMA device, such as *GRLIB Device and Vendor ID* and interrupt number. The *HCONFIG area* propagates this value for each DMA device. They are ordered in ascending order with regards to which DMA they are associated with in the register interface (bits [10:5] of the address shows the corresponding DMA index).

65.5.7 IOMMU registers

*Table 962.*GRIOMMU2 registers

AHB address offset	Register	Not implemented when
0x0000 - 0x0004	Capabilities (cap)	-
0x0008	Features-control register (fctl)	-
0x0010 - 0x0014	Device-directory-table pointer (ddtp)	-
0x0018 - 0x001C	Command-queue base (cqb)	-
0x0020	Command-queue head (cqh)	-
0x0024	Command-queue tail (cqt)	-
0x0028 - 0x002C	Fault-queue base (fqb)	-
0x0030	Fault-queue head (fqh)	-
0x0034	Fault-queue tail (fqt)	-
0x0048	Command-queue CSR (cqcsr)	-
0x004C	Fault-queue CSR (fqcsr)	-
0x0054	Interrupt pending status register (ipsr)	-
0x0058	Performance-monitoring counter overflow status (iocountovf)	If *HPM* generic is 0
0x005C	Performance-monitoring counter inhibits (iocountinh)	If *HPM* generic is 0
0x0060 - 0x0064	Performance-monitoring cycles counter (iohpmcycles)	If *HPM* generic is 0
0x0068 - 0x015C	Performance-monitoring event counters (iohpmctr1-31)	If *HPM* generic is 0
0x0160 - 0x0254	Performance-monitoring event selectors (iohpmevt1-31)	If *HPM* generic is 0
0x0258 - 0x025C	Translation-request IOVA (tr_req_iova)	If *DBG* generic is 0
0x0260 - 0x0264	Translation-request control (tr_req_ctl)	If *DBG* generic is 0
0x0268 - 0x026C	Translation-response (tr_response)	If *DBG* generic is 0
0x02B0 - 0x02B4	In-register command 0 (rcmd0)	-
0x02B8 - 0x02BC	In-register command 1 (rcmd1)	-
0x02F8 - 0x02FC	Interrupt-cause-to-vector register (icvec)	-
0x0300 - 0x03FC	MSI configuration table (msi_cfg_tbl)	-
0x1000 - 0x1038	CTX TLB Diagnostics area	-
0x1040 - 0x1078	SV TLB Diagnostics area	-
0x1080 - 0x10B8	G TLB Diagnostics area	-
0x10C0 - 0x10F8	MSI TLB Diagnostics area	-

65.5.8 Capabilities

*Table 963.*0x0000-0x0004 - CAP - Capabilities

63	48
Reserved	
0x00	
r	

47	41	40	39	38	37	32
Reserved		PD 20	PD 17	PD8	PAS	
0x0		*	*	*	0x20	
r						

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
dbg	hpm	igs		end	T2GPA	ATS	amo_hwad	msi_mrif	msi_flat	amo_mrif	Reserved	sv57x4	sv48x4	sv39x4	sv32x4
*	*	0x2		0	0	0	1	1	*	1	0	0	0	*	*
r															

15	14	13	12	11	10	9	8	7	0
Svpbmt	Svr-sw60t59b	Reserved		sv57	sv48	sv39	sv32	Version	
0	*	0x0		0	0	*	*	0x10	
r									

63:41	RESERVED
40	PD20 - Three level PDT with 20-bit process_id supported.
39	PD17 - Two level PDT with 17-bit process_id supported.
38	PD8 - One level PDT with 8-bit process_id supported.
37:32	PAS. - Physical Address Size supported.
31	DBG - Translation-request debug interface supported.
30	HPM - Hardware performance monitor implemented.
29:28	IGS - Interupt generation supported (0: MSI. 1: WSI. 2: MSI&WSI). Both MSI and WSI is always supported by GRIOMMU2 so IGS is constant 0x2.
27	END - If '1' bi-endian is supported, where current endian used is defined in the fctl register. The core does not support this feature and endianness is instead set as a generic, bit is constant '0'.
26	T2GPA - Returning guest-physical-address in ATS translation completions is supported.
25	ATS - CIe Address Translation Services (ATS) and pagerequest interface (PRI) is supported.
24	AMO_HWAD - Atomic updates to PTE accessed (A) and dirty (D) bit is supported.
23	MSI_MRIF - MSI address translation using MRIF mode MSI PTE is supported.
22	MSI_FLAT - MSI address translation using Pass-through mode MSI PTE is supported.
21	AMO_MRIF - Atomic updates to MRIF is supported.
20	RESERVED
19	Sv57x4 - Page-based 59-bit virtual addressing for second-stage address translation is supported.
18	Sv48x4 - Page-based 50-bit virtual addressing for second-stage address translation is supported.
17	Sv39x4 - Page-based 41-bit virtual addressing for second-stage address translation is supported.
16	Sv32x4 - Page-based 34-bit virtual addressing for second-stage address translation is supported.
15	Svpbmt - Page-based memory types.
14	Svrsw60t59b - PTE Reserved-for-Software Bits 60-59. Supported for when core is not configured in Sv32 mode.
13:12	RESERVED
11	Sv57 - Page-based 57-bit virtual addressing is supported.
10	Sv48 - Page-based 48-bit virtual addressing is supported.
9	Sv39 - Page-based 39-bit virtual addressing is supported.
8	Sv32 -Page-based 32-bit virtual addressing is supported.
7:0	Version - Specifies the RISCV IOMMU specification version implemented. The low nibble holds the minor version of the specification and the upper nimble hold the major version of the specification.

65.5.9 Feature control

*Table 964.*0x0008 - FCTL - Feature control

31 22	21	20	19	18	17	16	15 3	2	1	0
Reserved	bp EN	msi EN	g EN	sv EN	ctx EN	inv L	Reserved	gxl	wsi	be
0x0	1	0	0	0	0	1	0x0	*	1	*
r	rw	rw	rw	rw	rw	rw	r	r	rw	r

31:22	RESERVED
21	Enable TLB bypass (BPEN) - Enable TLB bypass functionality. If all required data for an address translation is already cached, the page walk state machine of the IOMMU is bypassed and an answer is sent after three clock cycles.
20	Enable MSI TLB (MSIEN) - Enables the MSI TLB. Disabling the MSI TLB will clear all valid bits.
19	Enable G TLB (GEN) - Enables G-stage (second stage) TLB. Disabling the G-stage TLB will clear all valid bits.
18	Enable SV TLB (SVEN) - Enables SV-stage (two stage) TLB. Disabling the SV-stage TLB will clear all valid bits.
17	Enable CTX TLB (CTXEN) - Enable CTX (combined DDT & PDT stage) TLB. Disabling the CTX TLB will clear all valid bits.
16	Invalidate locked TLB entries (INVL) - Enable the invalidation commands to be able to invalidate locked TLB entries (also removes the lock on invalidation). When bit is '0', invalidation commands will not modify locked entries.
15:3	RESERVED
2	Guest address translation allowed (GXL) - Determines which translation scheme is available for the second stage translation step. If ‘0’ then Sv32, if ‘1’ then Sv39. The value of GXL is set by the *Sv32* generic.
1	Wire-Signaled-Interrupt (WSI) - Interrupt scheme used by the core. When ‘1’, interrupts generated by the IOMMU are signaled as wire-signaled-interrupts else they are signaled as message-signaled-interrupts. This setting does not affect the interrupt forwarding functionality.
0	Big endian (BE) - Endianness used by the core. When ‘0’ little endian is used, when ‘1’ big endian is used. The value of *BE* is set by the *endian* generic .

Reset value: Implementation dependent

65.5.10 Device directory table pointer

*Table 965.*0x0010-0x0014 - DDTP - Device directory table pointer

63 54	53 32
Reserved	PPN
0x0	0x0
r	rw

31 10	9 5	4	3 0
PPN	Reserved	BUSY	MODE
0x0	0x0	0	*

*Table 965.*0x0010-0x0014 - DDTP - Device directory table pointer

rw	r	r	rw

63:54	RESERVED
53:10	Physical Page Number (PPN) - Holds the PPN of the root page of the device-directory-table.
9:5	RESERVED
4	Busy (BUSY) - When the DDTP IOMMU mode is written to the core will issue a fence for accesses from all DMA devices and set the. BUSY bit to '1'. When the fence has completed, the IOMMU mode is changed and the BUSY bit is set to '0'.
3:0	IOMMU mode (MODE) - Determines the mode of the core. 0: Off, the core will send a AHB error reply for any access. 1: Bare. no translation or protection. The core operates purely as a bridge. 2: 1LVL, translation and protection enabled. One-level device-directory-table used. 3: 2LVL, translation and protection enabled. Two-level device-directory-table used. 4: 3LVL, translation and protection enabled. Three-level device-directory-table used. The MODE value is only writable while the BUSY bit is set to '0'.

Reset value: Implementation dependent

65.5.11 Command-queue base

Table 966. 0x0018-0x001C - CQB - Command-queue base

63 54	53 32
Reserved	PPN
0x0	0x0
r	rw*

31 10	9 5	4 0
PPN	Reserved	SIZE
0x0	0x0	0x0
rw*	r	rw*

63:54 RESERVED

53:10 Physical Page Number (PPN) - Holds the PPN of the root page of the in-memory command-queue.

9:5 RESERVED

4:0 LOG2SZ-1 (SIZE) - Holds the number of entries in the command-queue as a log to base 2 minus 1 (i.e. #entries = 2^(SIZE+1).

Reset value: 0x0

* Only writable if the command-queue is turned off (*CQCSR.CQON* = '0').

65.5.12 Command-queue head

Table 967. 0x0020 - CQH - Command-queue head

31 0
INDEX
0x0
r

31:0 Command-queue head index (INDEX) - Holds the index into the command-queue from where the next command will be fetched by the IOMMU.

Reset value: 0x0

65.5.13 Command-queue tail

Table 968. 0x0024 - CQT - Command-queue tail

31 0
INDEX
0x0
rw

31:0 Command-queue tail index (INDEX) - Holds the index into the command-queue where the next command for the IOMMU should be stored. Only *[CQB.SIZE*:0] bits are writable.

Reset value: 0x0

65.5.14 Fault-queue base

Table 969. 0x0028-0x002C - FQB - Fault-queue base

63 54	53 32
Reserved	PPN
0x0	0x0
r	rw*

31 10	9 5	4 0
PPN	Reserved	SIZE
0x0	0x0	0x0

*Table 969.*0x0028-0x002C - FQB - Fault-queue base

rw*	r	rw*

63:54	RESERVED
53:10	Physical Page Number (PPN) - Holds the PPN of the root page of the in-memory fault-queue.
9:5	RESERVED
4:0	LOG2SZ-1 (SIZE) - Holds the number of entries in the fault-queue as a log to base 2 minus 1 (i.e. #entries = 2^(SIZE+1). Each fault record is 32-bytes.

Reset value: 0x0

* Only writable if the fault-queue is turned off (*FQCSR.FQON* = '0').

65.5.15 Fault-queue head

Table 970. 0x0030 - FQH - Fault-queue head

31	0
INDEX	
0x0	
rw	

31:0 Fault-queue head index (INDEX) - Hold the index into the fault-queue from which the next fault record should be read. Only [FQB.SIZE:0] bits are writable

Reset value: 0x0

65.5.16 Fault-queue tail

Table 971. 0x0034 - FQT - Fault-queue tail

31	0
INDEX	
0x0	
r	

31:0 Fault-queue tail index (INDEX) - Holds the index into the fault-queue where IOMMU writes the next fault record

Reset value: 0x0

65.5.17 Command-queue CSR

Table 972. 0x0048 - CQCSR - Command-queue CSR

31	18	17	16
Reserved		BUSY	CQON
0x0		0	0
r		r	r

15	12	11	10	9	8	7	2	1	0
Reserved		FWIP	CMD_ ILL	CMD_ TO	CQMF	Reserved		CIE	CQEN
0x0		0	0	0	0	0x0		0	0
r		wc	wc	r	wc	r		rw	rw

31:18 RESERVED

17 Command.queue busy (BUSY) - When the command-queue is set to be turned off (write '0' to *cqen*) any commands already fetched will need to be processed first, until this happens the *busy* bit will be kept high.

Register writes to *cqcsr* will be ignored while the *busy* bit is asserted.

16 Command-queue on (CQON) - The command-queue is active if *cqon* is '1'.

15:12 RESERVED

11 IOFENCE.C completed (FWIP) - Goes high when a IOFENCE.C command which has been configured to send a WSI on completion is completed.

Table 972. 0x0048 - CQCSR - Command-queue CSR

10	Command ill (CMD_ILL) - If an illegal or unsupported command is fetched and decoded by the command-queue then the command-queue sets the *cmd_ill* bit and stops processing from the command-queue.
9	Command timeout (CMD_TO) - If the execution of a command leads to a timeout then the command-queue sets the *cmd_to* bit and stops processing from the command-queue. The core however supports no such commands and the *cmd_to* will therefore never be set.
8	Command-queue-memory-fault (CQMF) - If a command-queue access leads to a memory fault then the *cqmf* bit is set to 1 and the command-queue stalls until this bit is cleared. Type of accesses that can set *cqmf*: - Command entry memory read. - Fence command AV write.
7:2	RESERVED
1	Command-queue interrupt enable (CIE) - Enables generation of interrupts from command-queue when set to '1'.
0	Command-queue enabled (CQEN) - Enables the command-queue when set to '1'. Changing *cqen* from '0' to '1' sets the *cmd_ill*, *cmd_to*, *cqmf* and *fence_w_ip* bits to '0'. When *cqen* is changed from '1' to '0' the command-queue may stay active (*cqon* has value '1') with the *busy* bit asserted until any commands which has already been fetched has been completely handled. When the command-queue has been turned off, the *cqon* and *busy* bits reads '0'.

Reset value: 0x0

65.5.18 Fault-queue CSR

Table 973. 0x004C - FQCSR - Fault-queue CSR

31	18	17	16
Reserved		BUSY	FQON
0x0		0	0
r		r	r

15	10	9	8	7	2	1	0
Reserved		FQOF	FQMF	Reserved		FIE	FQEN
0x0		0	0	0x0		0	0
r		wc	wc	r		rw	rw

31:18 RESERVED

17 Fault.queue busy (BUSY) - When the fault-queue is set to be turned off (write '0' to *fqen*) there can be some cycles before it is actually disabled to allow the IOMMU core to finish its current address translation request, during this time the *busy* bit is set.

Register writes to *fqcsr* will be ignored while the *busy* bit is asserted.

16 Fault-queue on (FQON) - The fault-queue is active if *fqon* is '1'.

15:10 RESERVED

9 Fault-queue-overflow (FQOF) - The *fqof* bit is set if the IOMMU needs to queue a fault record but the fault-queue is full (i.e. *fqt* == *fqh-1*).

8 Fault-queue-memory-fault (FQMF) - If a fault-queue access leads to a memory fault then the *fqmf* bit is set to 1 and the command-queue stalls until this bit is cleared.

Type of accesses that can set *fqmf:*
- Write of fault entry to memory.

7:2 RESERVED

1 Fault-queue interrupt enable (FIE) - Enables generation of interrupts from fault-queue when set to '1'.

0 Fault-queue enabled (FQEN) - Enables the fault-queue when set to '1'.

Changing *fqen* from '0' to '1' sets the *fqof*, and *fqmf* bits to '0' and the *fqt* register to 0.

When *fqen* is changed from '1' to '0' the fault-queue may stay active (*fqon* has value '1') with the *busy* bit asserted until the next time the IOMMU is back in a idle state (in practice this means that the current address translation, command execution or fault write will need to be completed). When the fault-queue has been turned off, the *cqon* and *busy* bits reads '0'.

Reset value: 0x0

65.5.19 Interrupt pending status register

Table 974. 0x0054- IPSR - Interrupt pending status register

31	16
Reserved	
0x0	
r	

15	4	3	2	1	0
Reserve		PIP	PMIP	FIP	CIP
0x0		0	0	0	0

Table 974. 0x0054- IPSR - Interrupt pending status register

r	r	wc	wc	wc

31:4	RESERVED
3	Page-request-queue-interrupt (PIP) - Not used.
2	Performance-monitoring-interrupt-pending (PMIP) - The *pmip* bit is set to '1' when the *OF* bit in *iohpmcycles* or any of the *iohpmevt1-31* registers transitions from '0' to '1'.
1	Fault-queue-interrupt-pending (FIP) - The *fip* bit is set to '1' if *fqcsr.fie* is '1' and any of the following are true: - *fqcsr.fqof* is '1' - *fqcsr-fqmf* is '1'. - A new record is produced in the fault-queue.
0	Command-queue-interrupt-pending (CIP) - The *cip* bit is set to '1' if *cqcsr.cie* is '1' and any of the following are true: - *cqcsr.fence_w_ip* is '1'. - *cqcsr.cmd_ill* is '1'. - *cqcsr.cqmf* is '1'.

Reset value: 0x0

65.5.20 Performance-monitoring counter overflow status

Table 975. 0x0058- IOCOUNTOVF - Performance-monitoring counter overflow status

31	1	0
HPM		CY
0x0		0
r		r

31:1 HPM overflow (HPM) - Shadow of *iohpmevt[1-31].OF*

0 Free running clock overflow (CY) - Shadow of *iohpmcycles.OF*

Reset value: 0x0

65.5.21 Performance-monitoring counter inhibits

Table 976. 0x005C- IOCOUNTINH- Performance-monitoring counter inhibits

31	1	0
HPM		CY
0x0		0
rw		rw

31:1 HPM inhibit (HPM) - When bit X is set, then counting of events in *iohpmctrX* is inhibited.

0 Free running clock inhibit (CY) - When set, *iohpmcycles* counter is inhibited from counting.

Reset value: 0x0

65.5.22 Performance-monitoring cycles counter

Table 977. 0x0060-0x0064- IOHPMCYCLES- Performance-monitoring cycles counter

63	62	0
OF	COUNTER	
0	0x0	
rw	rw	

63 Overflow (OF) - When the *counter* overflows, the *of* bit is set and will remain set until cleared by software.

62:0 Free running counter (COUNTER) - Free running cycles counter. Value of the counter will increase by one every cycle the counter is enabled (*iocountinh.cy* = 0).

Reset value: 0x0

65.5.23 Performance-monitoring event counter

Table 978. 0x0068-0x015C - IOHPMCTR1-31- Performance-monitoring event counter

63	0
COUNTER	
0x0	
rw	

63:0 Event counter (COUNTER) - Event counter value. For counter with index X, it is incremented whenever the event condition set in the event selector register for that counter index (*iohpmevtX)* is triggered while it is not inhibited (*iocountinh.hpm[x]*='0').

Reset value: 0x0

65.5.24 Performance-monitoring event selector

Table 979. 0x0160-0x0254 - IOHPMEVT1-31 - Performance-monitoring event selector

63	62	61	60	59 … 36	35 … 0
OF	IDT	DV_GSCV	PV_PSCV	DID_GSCID	PID_PSCID
0	0	0	0	0x0	0x0
rw	rw	rw	rw	rw	rw

63 … 16	15	14 … 0
PID_PSCID	DMASK	EVENTID
0x0	0	0x0
rw	rw	rw

63 Overflow (OF) - Overflow status for the corresponding *iohpmctr* counter.

62 Filder ID type (IDT) - Indicates the type of ID to filter on. When ‘0’, the *did_gscid* field holds a *device_id* and the *pid_pscid* field holds a *process_id*. When ‘1’, the *did_gscid* field holds a *gscid* and the *pid_pscid* field holds a *pscid*.

61 Match against DID/GSICD (DV_GSCV) - If set, only transactions matching the value in *did_gscid* are counted..

60 Match against PID/PSICD (PV_PSCV) - If set, only transactions matching the value in *pid_pscid* are counted.

59-36 DID/GSCID value (DID_GSCID) - Contains either a *device_id* or *a gscid* depending on the value of *idt*.

35-16 PID/PSCID value (PID_PSCID) - Contains either a *process_id* or *a pscid* depending on the value of *idt*.

15 DID masking (DMASK) - When set to ‘1’, partial matching of the *did_gsdid* is performed for the transaction. The lower bits of the *did_gscid* all the way to the first low order 0 bit (including the 0 bit position itself) are masked.

14:0 Event ID (EVENTID) - Indicates the event to count. When set to 0, no events are counted.

Reset value: 0x0

65.5.25 Translation-request IOVA

Table 980. 0x0258-0x025C - TR_REQ_IOVA - Translation-request IOVA

63 … 12	11 … 0
VPN	Reserved
0x0	0x0
rw	r

63:12 IOVA Virtual page number (VPN) - .Virtual page number used by the translation-request debug interface. Only writable when there is no debug request currently being processed (tr_req_ctl.go=’0’).

11: RESERVED

Reset value: 0x0

65.5.26 Translation-request control

Table 981. 0x0260-0x0264 - TR_REQ_CTL- Translation-request control

63 … 40	39 … 37	36	35 … 33	32
DID	Reserved	CACHE	Reserved	PV
0x0	0x0	0	0x0	0
rw	r	rw	r	rw

31 … 12	11 … 4	3	2	1	0
PID	Reserved	NW	EXE	PRIV	GO
0x0	0x0	0	0	0	0
rw	r	rw	rw	rw	rw

63-40 Device ID (DID) - Device ID used by the debug translation request.

Table 981. 0x0260-0x0264 - TR_REQ_CTL- Translation-request control

39-37	RESERVED
36	Allow TLB caching (CACHE) - When set, the page requests associated with the debug translation request will be cached in the TLBs.
35-33	RESERVED
32	Process ID Valid (PV) - Process ID valid used by the debug translation request.
31-12	Process ID (PID) - Process ID used by the debug translation request.
11-4	RESERVED
3	Non write permission (NW) - If set to '1', read permission is requested with the debug translation request, else both read and write permission is requested.
2	Execute permission (EXE) - .If set to '1', execute permission is requested with the debug translation request, else execute permission is not requested.
1	Privileged mode (PRIV) - .If set to '1', privileged mode access is requested with the debug translation request, else no privileged mode access is requested.
0	Start debug translation / Busy (GO) - Set to '1' to indicates that a translation request has been set up in the *tr_req_iova and tr_req_ctl* registers and that the debug translation request should start. The IOMMU will clear the *go* bit when the translation request has finished. On completion the results of the translation request will be in the *tr_respones* register. While *go*='1', any write to the *tr_req_ctl* or the *tr_req_iova* registers will be ignored.

Reset value: 0x0

65.5.27 Translation-response

Table 982. 0x0268-0x026C - TR_RESPONSE- Translation-response

63 54	53 32
Reserved	PPN
0x0	0x0
r	rw

31 10	9	8 7	6 1	0
PPN	S	PBMT	Reserved	FAULT
0x0	0	0x0	0x0	0
r	r	r	r	r

63-54 RESERVED

53-10 Physical page number (PPN) - If the translation request did not result in a fault (*fault*=0), then this field contains a PPN which is the result of the address translation performed on the VPN stored in *tr_req_iova*.

If the *s* bit is '0', then the size of the translation is 4 KiB, a page.

If the *s* bit is '1', then the translation resulted in a superpage, and the size of the superpage is encoded in the PPN itself. If scanning from bit position 0 to bit position 43, the first bit with a value of 0 at position X, then the superpage size is 2^(X+1) * 4 KiB

9 Superpage (S) - The translation resulted in a superpage.

8-7 Page-based memory type (PBMT) - Not implemented. Always "00"

6-1 RESERVED

0 Fault in translation (FAULT) - If the debug translation process resulted in a fault, the *fault* bit is set.

Reset value: 0x0

65.5.28 Interrupt-cause-to-vector register

Table 983. 0x02F8-0x02FC - ICVEC- Interrupt-cause-to-vector register

63 32
Reserved
0x0
r

31 16	15 12	11 8	7 4	3 0
Reserved	PIV	PMIV	FIV	CIV
0x0	0x0	0x0	0x0	0x0
r	rw	rw	rw	rw

63-16 RESERVED

15-12 Page-request-queue-interrupt-vector (PIV) - Not implemented.

11-8 Performance-monitoring-interrupt-vector (PMIV) - Interrupt vector number assigned to the performance-monitoring-interrupt. The amount of bits which are writable is determined by the VHDL parameter *num_vec_bits* (bits [*num_vec_bits*:0]).

7-4 Fault-queue-interrupt-vector (FIV) - Interrupt vector number assigned to the fault-queue-interrupt. The amount of bits which are writable is determined by the VHDL parameter *num_vec_bits* (bits [*num_vec_bits*:0]).

3-0 Command-queue-interrupt-vector (CIV) - Interrupt vector number assigned to the command-queue-interrupt. The amount of bits which are writable is determined by the VHDL parameter *num_vec_bits* (bits [*num_vec_bits*:0]).

Reset value: 0x0

65.5.29 In-register command 0

Table 984. 0x02B0-0x02B4 - RCMD0 - In-register command 0

63	0
cmd	
0x0	
rw	

63:0	Command 0 (CMD) - . First 64-bit area of the in-register command. Command is executed when *cqcsr.reg_cmd* is set.

Reset value: 0x0

65.5.30 In-register command 1

Table 985. 0x02B8-0x02BC - RCMD1 - In-register command 1

63	0
cmd	
0x0	
rw	

63:0	Command 1(CMD) - . Second 64-bit area of the in-register command. Command is executed when *cqcsr.reg_cmd* is set.

Reset value: 0x0

65.5.31 MSI configuration table

Table 986. 0x0300-0x03FC - MSI_CFG_TBL - MSI configuration table

63	0	Byte offsett
Entry 0: Message address		+000h
Entry 0: Vector Control	Entry 0: Message Data	+008h
Entry 1: Message address		+010h
Entry 1: Vector Control	Entry 1: Message Data	+018h

The MSI configuration table is made up of several MSI configuration entries. Each entry is split up into three parts: *Message address*, *vector control* and *message data*. The word offset of the *Vector Control* and *Message Data* is always following the little endian convention, even when big endian is used.

Table 987. +0x00 - MSI_CFG_TBL Message address

63 56	55 2	1 0
Reserved	ADDR	zero
0x0	0x0	0
r	rw	r

63:56	RESERVED
55:2	Address (ADDR) - Holds the 4-byte aligned MSI address
1:0	Zero

Reset value: 0x0

Table 988. +0x08 - MSI_CFG_TBL Message Data

31	0
DATA	
0x0	
rw	

31:0 Data (DATA) - Hold the MSI data

Reset value: 0x0

Table 989. +0x0C - MSI_CFG_TBL Vector Control

31	1	0
RESERVED		M
0x0		0
r		rw

31:1 RESERVED

0 Interrupt mask (M) - When the mask bit M is 1, the corresponding interrupt vector is masked and the IOMMU is prohibited from sending the associated message. Pending messages for that vector are later generated if the corresponding mask bit is cleared to 0.

Reset value: 0x0

65.5.32 CTX TLB index selector

Table 990. 0x1000 - CTX_INDEX - CTX TLB index selector

31	16	15	0
NUM_CTX		IND	
*		0x0	
r		rw	

31:16 Number CTX TLBs (NUM_CTX) - Number of available CTX TLB entries. Shadows the *ctx_tlbnum* VHDL parameter.

15:0 CTX index selector (IND) - Selects which CTX TLB entry that is exposed in the *CTX TLB Diagnostic interface* register. If a value is written that is larger than the total amount of available CTX TLBs, then the value instead will be set to 0.

Reset value: 0x0

65.5.33 CTX TLB Diagnostic interface

With the CTX TLB Diagnostic interface one CTX TLB entry, selected by the *CTX TLB index selector* register, can be accessed at a time. The diagnostic area is split up into seven 64b registers.

Table 991. 0x1008-0x100C - CTX_DIAG0 - CTX TLB Diagnostic interface, word 0

63	62	46	45	44	25	24	1	0
LCK	RESERVED		PIDV	PID		DID		V
0	0		0	0x0		0x0		0
rw	r		rw	rw		rw		rw

Table 991. 0x1008-0x100C - CTX_DIAG0 - CTX TLB Diagnostic interface, word 0

63	Entry locked (LCK) - TLB entry locked bit. If set the TLB entry will never be overwritten by the IOMMU during the process of a address translation. If the *fctl.invl* bit is set, then the TLB entry can be cleared through the command-queue with the *INVAL_DDT* or *INVAL_PDT* commands.
62:46	RESERVED
45	Process ID valid (PIDV) - Process ID valid bit used as a tag for the TLB lookup.
44:25	Process ID field (PID) - Process ID used as a tag for the TLB lookup.
25:1	Device ID (DID) - Device ID used as a tag for the TLB lookup.
0	Entry Valid (V) - TLB entry valid bit

Reset value: 0x0

Table 992. 0x1010-0x1014 - CTX_DIAG1 - CTX TLB Diagnostic interface, word 1

63 48	47 44	43 0
RESERVED	SATP_M	SATP_PPN
r	rw	rw
0x0	0x0	0x0

63:48	RESERVED
47:44	IOSATP mode (SATP_M) - TLB entry IOSATP mode value used.
43:0	IOSATP PPN (SATP_PPN) - TLB entry IOSATP PPN value used.

Reset value: 0x0

Table 993. 0x1018-0x101C - CTX_DIAG2 - CTX TLB Diagnostic interface, word 2

63 48	47 44	43 0
RESERVED	GATP_M	GATP_PPN
r	rw	rw
0x0	0x0	0x0

63:48	RESERVED
47:44	IOHGATP mode (GATP_M) - TLB entry IOHGATP mode value used.
43:0	IOHGATP PPN (GATP_PPN) - TLB entry IOHGATP PPN value used.

Reset value: 0x0

Table 994. 0x1020-0x1024 - CTX_DIAG3 - CTX TLB Diagnostic interface, word 3

63 42	41	40	39	38	37	36	35 20	19 0
RESERVED	GADE	SADE	DTF	BYP	SUM	ENS	GSCID	PSCID
0x0	0	0	0	0	0	0	0x0	0x0
r	rw	rw	rw	rw	rw	rw	rw	rw

63:42	RESERVED
41	GADE (GADE) - TLB entry GADE value used.
40	SADE (SADE) - TLB entry SADE value used.

Table 994. 0x1020-0x1024 - CTX_DIAG3 - CTX TLB Diagnostic interface, word 3

39	Disable-translation.-fault (DTF) - TLB entry DTF value used.
38	Bypass (BYP) - TLB entry bypass value used. If set, when the address translation is completed and the translated address is to the striped memory area then the access will use the secondary striped interface.
37	Permit Supervisor User Memory access (SUM) - TLB entry SUM value used.
36	Enable-Supervisory-access (ENS) - TLB entry ENS value used.
35:20	Guest soft -context identifier (GSCID) - TLB entry GSCID value used.
19:0	Process soft -context identifier (PSCID) - TLB entry PSCID value used.

Reset value: 0x0

Table 995. 0x1028-0x102C - CTX_DIAG4 - CTX TLB Diagnostic interface, word 4

63 48	47 44	43 0
RESERVED	MSI_M	MSI_PPN
r	rw	rw
0x0	0x0	0x0

63:48	RESERVED
47:44	MSI MODE (MSI_M) - TLB entry MSI mode value used.
43:0	MSI PPN (MSI_PPN) - TLB entry MSI PPN value used.

Reset value: 0x0

Table 996. 0x1030-0x1034 - CTX_DIAG5 - CTX TLB Diagnostic interface, word 5

63 52	51 0
RESERVED	MSI_MSK
0x0	0x0
rw	rw

63:52	RESERVED
51:0	MSI mask (MSI_MSK) - TLB entry MSI mask value used.

Reset value: 0x0

Table 997. 0x1038-0x103C - CTX_DIAG6 - CTX TLB Diagnostic interface, word 6

63 52	51 0
RESERVED	MSI_PAT
0x0	0x0
rw	rw

63:52	RESERVED
51:0	MSI pattern (MSI_PAT) - TLB entry MSI pattern value used.

Reset value: 0x0

65.5.34 SV-stage TLB index selector

Table 998. 0x1040 - SV_INDEX - SV-stage TLB index selector

31 16	15 0
NUM_SV	IND
*	0x0
r	rw

31:16 Number SV-stage TLBs (NUM_SV) - Number of available SV-stage TLB entries. Shadows the *sv_tlbnum* VHDL parameter.

15:0 SV-stage index selector (IND) - Selects which SV-stage TLB entry that is exposed in the *SV-stage TLB Diagnostic interface* register. If a value is written that is larger than the total amount of available SV-stage TLBs, then the value instead will be set to 0.

Reset value: 0x0

65.5.35 SV-stage TLB Diagnostic interface

With the SV-stage TLB Diagnostic interface one SV-stage TLB entry, selected by the *SV-stage TLB index selector* register, can be accessed at a time. The diagnostic area is split up into three 64b registers.

Table 999. 0x1048-0x104C - SV_DIAG0 - SV-stage TLB Diagnostic interface, word 0

63	62 46	45 2	1	0
LCK	RESERVED	PPN	GSV	V
0	0	0x0	0	0
rw	r	rw	rw	rw

63 Entry locked (LCK) - TLB entry locked bit. If set the TLB entry will never be overwritten by the IOMMU during the process of a address translation. If the *fctl.invl* bit is set, then the TLB entry can be cleared through the command-queue with the *IOTINVAL* command.

62:46 RESERVED

45:2 Physical page number (PPN) -Physical page number used as a tag for the TLB lookup.

1 Guest soft -context identifier valid (GSV) - Guest soft -context identifier valid bit, used as tag for the TLB lookup.

0 Entry Valid (V) - TLB entry valid bit

Reset value: 0x0

Table 1000. 0x1050-0x1054 - SV_DIAG1 - SV-stage TLB Diagnostic interface, word 1

63 37	36	35 16	15 0
RESERVED	SADE	PSCID	GSCID
0x0	0	0x0	0x0
r	rw	rw	rw

63:37 RESERVED

36 SADE (SADE) -SADE value, used as a tag for the TLB lookup.

34:16 Process soft -context identifier (PSCID) - Process soft -context identifier , used as tag for the TLB lookup.

15:0 Guest soft -context identifier (GSCID) - Guest soft -context identifier , used as tag for the TLB lookup.

Reset value: 0x0

Table 1001. 0x1058-0x105C - SV_DIAG2 - SV-stage TLB Diagnostic interface, word 2

63 52	51	50	49	48	47	46	45 2	1 0
RESERVED	X	D	A	U	R	W	PPN	LVL
r	rw	rw	rw	rw	rw	rw	rw	rw
0x0	0	0	0	0	0	0	0x0	0x0

63:52 RESERVED

51 Execute permission bit (X) - TLB entry execute permission bit used

50 Dirty bit (D) - TLB entry dirty bit used

49 Accessed bit (A) - TLB entry accessed bit used.

48 User mode bit (U) - TLB entry user mode bit used.

47 Read permission bit (R) - TLB entry read permission bit used

46 Write permission bit (W) - TLB entry write permission bit used.

45:2 Physical page number (PPN) - TLB entry PPN used.

1:0 PTE Level (LVL) - TLB entry leaf size.

Sv32: “00”: 4KiB page, “01” 4MiB *megapage*

Sv39: “00” 4KiB page, “01” 2MiB *megapage*, “10” 1GiB *gigapage*

15:0

Reset value: 0x0

65.5.36 G-stage TLB index selector

Table 1002. 0x1080 - G_INDEX - G-stage TLB index selector

31 16	15 0
NUM_G	IND
*	0x0
r	rw

31:16 Number G-stage TLBs (NUM_G) - Number of available G-stage TLB entries. Shadows the *g_tlb-num* VHDL parameter.

15:0 G-stage index selector (IND) - Selects which G-stage TLB entry that is exposed in the *G-stage TLB Diagnostic interface* register. If a value is written that is larger than the total amount of available G-stage TLBs, then the value instead will be set to 0.

Reset value: 0x0

65.5.37 G-stage TLB Diagnostic interface

With the G-stage TLB Diagnostic interface one G-stage TLB entry, selected by the *G-stage TLB index selector* register, can be accessed at a time. The diagnostic area is split up into two 64b registers.

Table 1003. 0x1088-0x108C - G_DIAG0 - G-stage TLB Diagnostic interface, word 0

63	62 61	60 45	44 1	0
LCK	RESERVED	GSCID	GPA	V
0	0x0	0x0	0x0	0

Table 1003. 0x1088-0x108C - G_DIAG0 - G-stage TLB Diagnostic interface, word 0

rw	r	rw	rw	rw

63	Entry locked (LCK) - TLB entry locked bit. If set the TLB entry will never be overwritten by the IOMMU during the process of a address translation. If the *fctl.invl* bit is set, then the TLB entry can be cleared through the command-queue with the *IOTINVAL* command.
62:61	RESERVED
60:45	Guest soft -context identifier (GSCID) - Guest soft -context identifier value used as tag for the TLB lookup..
44:1	Guest physical address (GPA) - Guest physical address used as a tag for the TLB lookup. (Note: GPA is a PPN, i.e. it is index after 4KiB).
0	Entry Valid (V) - TLB entry valid bit

Reset value: 0x0

Table 1004. 0x1090-0x1094 - G_DIAG1 - G-stageTLB Diagnostic interface, word 1

63 52	51	50	49	48	47	46	45 2	1 0
RESERVED	X	D	A	U	R	W	PPN	LVL
r	rw	rw	rw	rw	rw	rw	rw	rw
0x0	0	0	0	0	0	0	0x0	0x0

63:52	RESERVED
51	Execute permission bit (X) - TLB entry execute permission bit used
50	Dirty bit (D) - TLB entry dirty bit used
49	Accessed bit (A) - TLB entry accessed bit used.
48	User mode bit (U) - TLB entry user mode bit used.
47	Read permission bit (R) - TLB entry read permission bit used
46	Write permission bit (W) - TLB entry write permission bit used.
45:2	Physical page number (PPN) - TLB entry PPN used.
1:0	PTE Level (LVL) - TLB entry leaf size. Sv32: “00”: 4KiB page, “01” 4MiB *megapage* Sv39: “00” 4KiB page, “01” 2MiB *megapage*, “10” 1GiB *gigapage*
15:0	

Reset value: 0x0

65.5.38 MSI TLB index selector

Table 1005. 0x10C0 - MSI_INDEX - MSI TLB index selector

31 16	15 0
NUM_MSI	IND
*	0x0
r	rw

31:16	Number MSI TLBs (NUM_MSI) - Number of available MSI TLB entries. Shadows the *msi_tlbnum* VHDL parameter.
15:0	MSI index selector (IND) - Selects which MSI TLB entry that is exposed in the *MSI TLB Diagnostic interface* register. If a value is written that is larger than the total amount of available MSI TLBs, then the value instead will be set to 0.

Reset value: 0x0

65.5.39 MSI TLB Diagnostic interface

With the MSI TLB Diagnostic interface one MSI TLB entry, selected by the *MSI TLB index selector* register, can be accessed at a time. The diagnostic area is split up into three 64b registers.

Table 1006. 0x10C8-0x10CC - MSI_DIAG0 - MSI TLB Diagnostic interface, word 0

63	62 61	60 45	44 1	0
LCK	RESERVED	GSCID	GPA	V
0	0x0	0x0	0x0	0
rw	r	rw	rw	rw

63 Entry locked (LCK) - TLB entry locked bit. If set the TLB entry will never be overwritten by the IOMMU during the process of a address translation. If the *fctl.invl* bit is set, then the TLB entry can be cleared through the command-queue with the *IOTINVAL* command.

62:61 RESERVED

60:45 Guest soft -context identifier (GSCID) - Guest soft -context identifier value used as tag for the TLB lookup..

44:1 Guest physical address (GPA) - Guest physical address used as a tag for the TLB lookup. (Note: GPA is a PPN, i.e. it is index after 4KiB).

0 Entry Valid (V) - TLB entry valid bit

Reset value: 0x0

Table 1007. 0x10D0-0x10D4 - MSI_DIAG1 - MSI TLB Diagnostic interface, word 1

63 60	59 49	48	47 1	0
RESERVED	NID	C	ADDR	MRIF
0x0	0x0	0	0x0	0
r	rw	rw	rw	rw

63:60 RESERVED

59:49 Notice Identifier (NID) - MSI TLB entry Notice identifier used. (Only used by MRIF MSI PTE entries)

48 Custom-use (C) - MSI TLB entry custom-use bit used. (Currently no custom mode is defined for the MSI handling of the core, a *msi pte misconfigured* fault will be thrown if this bit is set.)

47:1 MSI PPN / MRIF ADDR (ADDR) - If the MSI TLB entry is a MRIF (*MRIF='1'*), then the *ADDR* field holds the MRIF address. Otherwise the MSI TLB entry is in basic translation mode and the field instead contains the MSI PPN.

0 Is MRIF (MRIF) - If the *MRIF* bit is set the MSI TLB entry is treated as a MRIF PTE (M=1 in the MSI PTE). Otherwise the MSI TLB entry is treated as MSI basic translation mode (M=3 in the MSI PTE).

Reset value: 0x0

Table 1008. 0x10D8-0x10DC - MSI_DIAG2 - MSI TLB Diagnostic interface, word 2

63 44	43 0
RESERVED	NPPN
0x0	0x0
r	rw

63:44 RESERVED

43:0 Notice Physical Page Number (NPPN) - MSI TLB entry NPPN used. (Only used by MRIF MSI PTE entries)

Reset value: 0x0

65.5.40 IOPMP registers

See RISC-V IOPMP specification for register description. Only differences from specification is that registers *MDSTALL, ERR_REQADDR, MDLCK, SRCMD_EN* and *ENTRY_ADDR* are considered to be 64b registers, see chapter 65.5 for implications of this change.

Detailed register description TBD.

65.6 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x0D3. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

65.7 Implementation

65.7.1 Reset

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

65.7.2 Endianness

GRIOMMU2 supports big and little endian systems, the endianness is set by the *cpumi.endian* signal. As per the RISC-V IOMMU specification, the register interface of the IOMMU should be in little endian format, even in the case of a big endian system. However, there is a VHDL generic called *be_reg_order* that can be used to override this behavior for 32b accesses to 64b registers to provide better support for LEON systems.

65.8 Configuration options

Table 1009 shows the configuration options of the core (VHDL generics).

*Table 1009.*Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
ndmamst	Number of DMA devices connected.	1 - 63	-
sbusw*	AHB bus width for memory bus (both normal and bypass interface)	64, 128	-
paddrw*	Address width used for striped memory bus (both normal and bypass interface).	32-39	-
fifo_tech*	Memory technology used for FIFO buffers for the striped interface	0 - NTECH	inferred
fifo_abits*	Address bits of the FIFO buffers for the striped interface.	0- 16#FFFF#	2
fifo_ft*	Fault tolerance enable for the FIFO buffers for the striped interface.	0, 1	0
cpumidx	AHB master index.	0 - NAHBMST-1	-
strmidx*	Stripe AHB master index	0 - NAHBMST-1	-
strmidx_bp*	Stripe AHB master index, bypass interface.	0 - NAHBMST-1	-
hindex	AHB slave index (register interface).	0 - NAHBSLV-1	-
bridge_addr	I/O Bridge register BAR base address	0 - 16#FFF#	16#000#
bridge_mask	I/O Bridge register BAR address mask	0 - 16#FFF#	16#FFF#
iommu_included	Determines if the IOMMU core should be included in the design.	0, 1	1
iommu_addr	IOMMU register BAR base address	0 - 16#FFF#	16#000#
iommu_mask	IOMMU register BAR address mask	0 - 16#FFF#	16#FFF#

*Table 1009.*Configuration options (VHDL generics)

Generic	Function	Allowed range	Default
hirq	AHB interrupt line used by the IOMMU core.	0 - NAHBIRQ-1	0
dbg	Enable debug interface capabilities	0, 1	1
pclvl	Number of PDT page walk levels supported.	0 - 3	1
msi_flat	Enables translation of MSI capabilities.	0, 1	1
hpm	Number of performance monitoring entries enabled.	0 - 32	8
hpm_ctr_width	Width in bits of the HPM counters	0 - 64	64
num_vec_bits	Determines the number of interrupt cause sources. For WSI, *num_vec_bits* is the number of bits that are writable in the *icvec* register, as well as the number of bits that make up the outgoing irq interrupt line. For MSI, *num_vec_bits* determines the number of active *msi_cfg_tbl* entries in the format: *#msi_cfg_tbl=2^num_vec_bits*.	0 - 4	0
Sv32	Set the IOMMU to use the Sv32 translation scheme instead of the default Sv39.	0, 1	0
ctx_tlbnum	Number of CTX TLB entries (used by combined DDT and PDT page walk).	0 - 31	8
sv_tlbnum	Number of SV TLB entries (used by first- and two-stage address translation page walk).	0 - 31	8
g_tlbnum	Number of G TLB entries (used by second stage address translation page walk).	0 - 31	8
msi_tlbnum	Number of MSI TLB entries (used by MSI-PTE lookup).	0 - 31	2
b_hpm	Bridge performance monitoring counters enabled.	0, 1	1
be_reg_order	Enables a special mode where 32b accesses to 64b registers will be endian dependent. This goes against the RISCV IOMMU specification but increases the compatibility between big endian systems with the core.	0, 1	0
use_ahb_priv	Use the AHB HPROT[1] to determine privilege mode for an access. Otherwise the default value stored in *default_priv* is used instead.	0, 1	0
default_priv	Default privilege mode value. *default_priv*='1' means privileged accesses as default.	0, 1	0
use_ahb_exe	Use the AHB HPROT[0] to determine execution intent for an access. Otherwise the default value stored in *default_exe* is used instead.	0, 1	0
default_exe	Default execution intent value. *default_exe*='1' means execution intent as default.	0, 1	0
iopmp_included	Determines if the IOPMP core should be included in the design.	0, 1	0
iopmp_addr	IOPMP register BAR base address	0 - 16#FFF#	16#000#
iopmp_mask	IOPMP register BAR address mask	0 - 16#FFF#	16#FFF#
iopmp_num_mds	Number of memory domains active in the IOPMP.	0 - 63	8
iopmp_k_n	K-value used by the Rapid-K model in the IOPMP in the format: *K_VALUE=2^iopmp_k_n*	0 - 8	2
iopmp_p_chunks	Number of chunks which should contain priority PMP entries.	0 - 256	2
iopmp_chunk_s	Size of chunk in number of PMP entries.	1 - 256	8
iopmp_cache_s	Number of entries in the IOPMP cache.	1 - 256	8
** Only available in the advanced version of the IP*			

65.9 Signal descriptions

Table 1010 shows the interface signals of the core (VHDL ports).

*Table 1010.*Signal descriptions (VHDL ports)

Signal name	Field	Type	Function	Active
IOCLK		Input	Clock used by the I/O bridge, IOMMU, IOPMP, and register interface.	-
STRCLK^		Input	Clock used by the striped memory controller.	-
RSTN		Input	Reset	Low
PID		Input	Vector of 19 bit RISC-V Process IDs with length *ndmamst-1*. Used by PDT page walk.	-
PID_V		Input	Vector of process ID valid bits of length *ndmamst-1*. Used by PDT page walk.	-
DID		Input	Vector of 23 bit RISC-V Device IDs of length *ndmamst-1*. Used by DDT page walk.	-
DMAMI	*	Output	AHB master input signal provided to the I/O DMA devices	-
DMAMO	*	Input	AHB master output signal provided to the I/O DMA devices.	-
CPUMI	*	Input	AHB master input signal. Used for the conventional bus.	-
CPUMO	*	Output	AHB master output signal. Used for the conventional bus.	-
STRMI0^	*	Input	Striped AHB master input signal.	-
STRMO0^	*	Output	Striped AHB master output signal.	-
STRMI1^	*	Input	Striped AHB master input signal, bypass interface.	-
STRMO1^	*	Output	Striped AHB master output signal, bypass interface.	-
STRCFG^		Input	Stripe configuration.	-
AHBSI	*	Input	Register interface AHB slave input signal.	-
AHBSO	*	Output	Register interface AHB slave output signal.	-
RST_MODE	*	Input	Signal sampled at reset which determines the reset value of the *ddtp.iommu_mode* register. ‘0’ = OFF, ‘1’ = BARE	-
IRQ		Output	Wired signal interrupt output bit vector of length *num_vec_bits*.	-

* see GRLIB IP Library User’s Manual

^ *Only available in the advanced version of the IP*

65.10 Library dependencies

Table 1011 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1011.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	GRIOMMU2_PKG	Components, Types, Constants	Component declaration for GRIOMMU2 and interface
TECHMAP	GENCOMP	Component, Constants	Component declaration for *syncfifo_2p* used by the *advanced* version of the core as well as constant for tech generic.

66 GRLSEDC- Lattice FPGA soft error detection interface

66.1 Overview

The GRLSEDC IP is a wrapper block and gives a user the ability to perform SEDSEC (Soft-Error Detection & Soft-Error Correction) on Lattice FPGAs that support it. The supported FPGAs are: LIFCL-17, LIFCL-33, LIFCL-40, LFD2NX-17, LFD2NX-40, LFCPNX-50, LFCPNX-100 and LFMXO525. GRLSEDC is configured via APB registers.

A block diagram of the IP is shown in Figure 159 and the internal block diagram is shown in Figure 160.

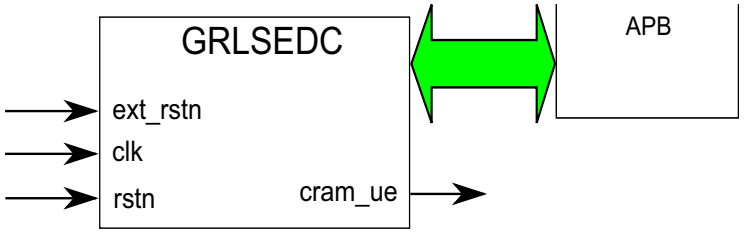


Figure 159. Top-level view of GRLSEDC IP.

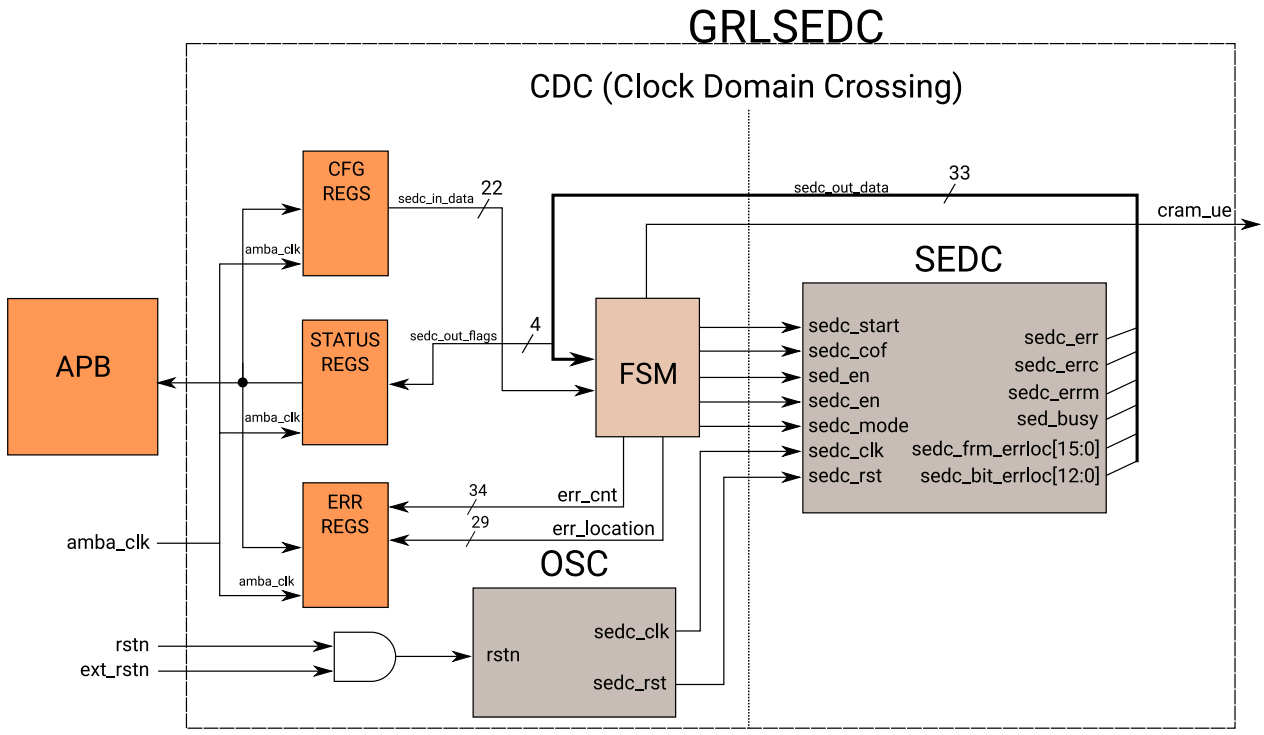


Figure 160. Internal view of GRLSEDC IP.

66.2 Configuration

The clock frequency of the scrubber is configured in a file called *lattice_ips/lifcl/osc/osc_cram_sedc_50o.cfg* inside *boards/<lattice board>* (50MHz by default). The clock frequency is calculated by the following formula:

```
clk_freq = 450/sedclk_div
```

Where *sedclk_div* is an integer in the range 2 to 256. After calculating the requested clock frequency, the *osc_cram_sedc_50o.cfg* file must be updated accordingly (*SEDSEC_CLK_FREQ* must be changed):

```
{
    "HF_OSC_EN": "DISABLED",
    "LF_OUTPUT_EN": "DISABLED",
    "SEDCLK_EN": 1,
    "SEDSEC_CLK_FREQ": 50.0,
    "LMMI_CLK_EN": "DISABLED"
}
```

The other configuration options can be changed on the implemented design by writing to the configuration register of the core (see Table 1013).

66.3 Operation

66.3.1 FSM

The GRLSEDC IP is controlled by an FSM consisting of the following states:

idle: Reset state and return state upon completion of an SEDC run. The *en* signal is pulled low in order to reset all sticky error signals. Moves to *start_run* if ran in continuous mode, or if the number of clock cycles indicated by *SCRUBPER* have elapsed.

start_run: Assign *start* and *en* signals depending on which mode is currently running, then move to *running* when the SEDC unit starts

running: The SEDC unit is currently running. Move to different stages depending on what error is found. If no errors are detected, and the run is completed, return to *idle*

ce: A correctable error has been detected in the current frame. Return to *running* when done

ue: An uncorrectable error has been detected in the current frame. If the continue-on-failure (*cof*) signal is high return to *running*, otherwise stay in this state until a power-cycle is performed

crce: A CRC error has occurred, possibly caused by the CRC checksum being incorrectly programmed in the bitstream. If the continue-on-failure (*cof*) signal is high, return to *running*, otherwise stay in this state until a power-cycle is performed

The FSM is shown in Figure 161.

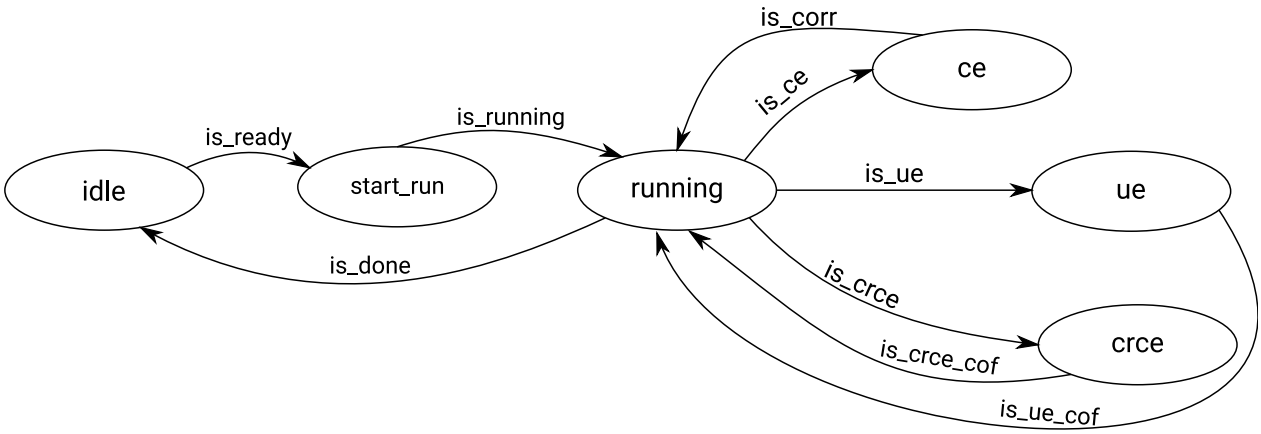


Figure 161. FSM of GRLSEDC IP

66.3.2 Error injection

To simulate an error injection, certain steps need to be followed. The only way to inject an error is through the Lattice Radiant software, specifically the Soft Error Injection Editor:

1. Start by enabling *BACKGROUND_RECONFIG* in the **Global** settings under **Device Constraint Editor** when generating the bitstream

2. Run **SEI Editor** and create errors. These errors can be placed within three different blocks of the configuration memory: PFU, EBR and DSP

3. Program the faulty frame bitstream by selecting **SEI Fast Program** inside **Radiant Programmer**

4. The error can now be detected and, optionally, corrected by GRLSEDC. **NOTE:** SED checking must be stopped while programming the device with the faulty frame bitstream. Easiest solution is to assert *RST* by writing to the configuration register. Once the background reprogramming finishes *RST* can be deasserted safely

66.3.3 Limitations

1. Frame error location in the **SEI Editor** is only accurate since **Lattice Radiant 2023.1**. This version still contains mismatches for the bit error location. **Lattice Radiant 2023.2** displays the correct frame and bit error location

2. **IMPORTANT:** Dynamic RAM blocks are not masked by the SEDC module, meaning that these blocks cannot be included in a design to detect and correct errors. Therefore, the design must be constrained to remove these blocks:

```
define_attribute {v:gaisler.mmu_icache*} {syn_ramstyle} {registers}
define_attribute {v:gaisler.mmu_dcache*} {syn_ramstyle} {registers}
```

In this example, the blocks *mmu_icache* and *mmu_dcache* were implemented in dynamic RAM cells by default. The two constraints above force them to be implemented in block RAM (EBR) cells instead. These constraints should be added to an *FDC* constraint file.

3. If an error occurs during configuration, the device needs to be power-cycled to function correctly again

4. Only one error can be injected at a time

5. Enabling the scrubber while connected to the board over JTAG causes the debug unit to disconnect and the design will need to be reprogrammed/power-cycled to fix the error

66.4 Registers

The core is configured via registers mapped into the AHB memory address space. Both 32-bit and 64-bit accesses are supported.

*Table 1012.*GRLSEDC: APB registers

AHB address offset	Register
0x00	Configuration
0x04	Correctable error counters
0x08	Uncorrectable error counters
0x0C	Error location
0x10	Status

66.4.1 Configuration Register

*Table 1013.*0x00 - LSEDCC - GRLSEDC Configuration register

31 21	20 5	4	3	2	1	0
RESERVED	SCRUBPER	RST	CEN	COF	MODE	EN
0	0	0	1	1	1	0
r	rw	rw	rw	rw	rw	rw

31: 21 RESERVED

20: 5 Scrubber period (SCRUBPER) - Clock cycles to wait between SEDC scrubbing events.

4 Reset (RST) - Reset the SEDC and return wrapper to *idle* state.

3 Correction enable (CEN) - Enable correction of correctable errors.

2 Continue-on-failure (COF) - Continue scrubbing cycle even if an uncorrectable error is detected.

1 Mode (MODE) - Scrubbing mode. 0 = one-shot 1 = continuous.

0 Enable (EN) - Enable scrubbing.

66.4.2 Correctable Error Counters register

*Table 1014.*0x04 - LSEDCCEC - GRLSEDC Correctable Error Counters register

31 16	15 0
CED	CEC
0	0
rw	rw

31: 16 Correctable error detected (CED) - Counter for detected correctable errors.

15: 0 Correctable error corrected (CEC) - Counter for corrected correctable errors.

66.4.3 Uncorrectable Error Counters register

*Table 1015.*0x08 - LSEDCUEC - GRLSEDC Uncorrectable Error Counters register

31 2	1	0
RESERVED	UE	CRCE
0	0	0
r	rw	rw

31: 2 RESERVED

1 Uncorrectable error (UE) - Uncorrectable error detected.

0 CRC error (CRCE) - CRC error detected.

66.4.4 Error Location register

*Table 1016.*0x0C - LSEDCEL - GRLSEDC Error Location register

31 29	28 16	15 0
RESERVED	BITEL	FRMEL
0	0	0
r	r	rw

31: 29 RESERVED

28: 16 Bit error location (BITEL) - Bit position in the frame of the detected error.

15: 0 Frame error location (FRMEL) - Frame location of the detected error.

66.4.5 Status register

*Table 1017.*0x10 - LSEDCS - GRLSEDC Status register

31	6	5	4	3	2	1	0
RESERVED		START	ERR	ERRM	ERRC	ERRCRC	BUSY
0		0	0	0	0	0	0
r		r	r	r	r	r	r

31: 6	RESERVED
5	Start flag (START) - Indicates if an SEDC scrub cycle has been started.
4	Correctable error flag (ERR) - Indicates if a correctable error has occurred.
3	Uncorrectable error flag (ERRM) - Indicates if an uncorrectable error has occurred.
2	Error in current frame flag (ERRC) - Indicates if an error was observed in the frame currently being scrubbed.
1	CRC error flag (ERRCRC) - Indicates if a CRC error has occurred.
0	Busy flag (BUSY) - Indicates if the SEDC is busy (running) or not.

66.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x0DE. For description of vendor and device identifier see GRLIB IP Library User's Manual.

66.6 Implementation

66.6.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). When reset is asserted the FSM returns to *idle* state and all error counters are reset to zero.

66.6.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual).

66.7 Configuration options

Table 1018 shows the configuration options of the core (VHDL generics).

*Table 1018.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index.	0 - NAPBSLV-1	0
paddr	ADDR field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#

66.8 Signal descriptions

Table 1019 shows the interface signals of the core (VHDL ports).

*Table 1019.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
EXT_RSTN	N/A	Input	External reset	Low
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
CRAM_UE	N/A	Output	Uncorrectable error detected	High

*) see GRLIB IP Library User's Manual.

66.9 Library dependencies

Table 1020 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1020.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Amba, stdlib, devices	AMBA signal definitions
TECHMAP	GENCOMP	All	Memory components
GAISLER	GRLSEDC	All	Component declaration

66.10 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.stdlib.all;
use grlib.devices.all;

library gaisler;
use gaisler.grlsedc_pkg.all;

library techmap;
use techmap.gencomp.all;


entity grlsedc_ex is

  port (
    clk      : in  std_ulogic;
    rstn     : in  std_ulogic;
    ext_rstn : in  std_ulogic;
    cram_ue  : out std_ulogic);

end entity grlsedc_ex;

architecture rtl of grlsedc_ex is

 -- APB signals
 signal apbi : apb_slv_in_type;
```

```
 signal apbo : apb_slv_out_vector := (others => apb_none);

begin  -- architecture rtl

  -- AMBA components are instantiated here
  ...

  grlsedc0 : grlsedc
    generic map(
      pindex => 1,
      paddr  => 16#100#,
      pmask  => 16#FFF#)
    port map(
      clk      => clk,
      rstn     => rstn,
      ext_rstn => ext_rstn,
      apbi     => apbi,
      apbo     => apbo(1),
      cram_ue  => cram_ue);
```

67 GRPCI2 - 32-bit PCI(Initiator/Target) / AHB(Master/Slave) bridge

67.1 Overview

The GRPCI2 core is a bridge between the PCI bus and the AMBA AHB bus. The core is capable of connecting to the PCI bus via both a target and a initiator/master interface. The connection to the AMBA bus is an AHB master interface for the PCI target functionality and an AHB slave interface for the PCI initiator functionality. The core also contains a DMA controller. For the DMA functionality, the core uses the PCI initiator to connect to the PCI bus and an AHB master to connect to the AMBA bus. Configuration registers in the core are accessible via a AMBA APB slave interface.

The PCI and AMBA interfaces belong to two different clock domains. Synchronization is performed inside the core through FIFOs with configurable depth.

The PCI interface is compatible with the 2.3 PCI Local Bus Specification.

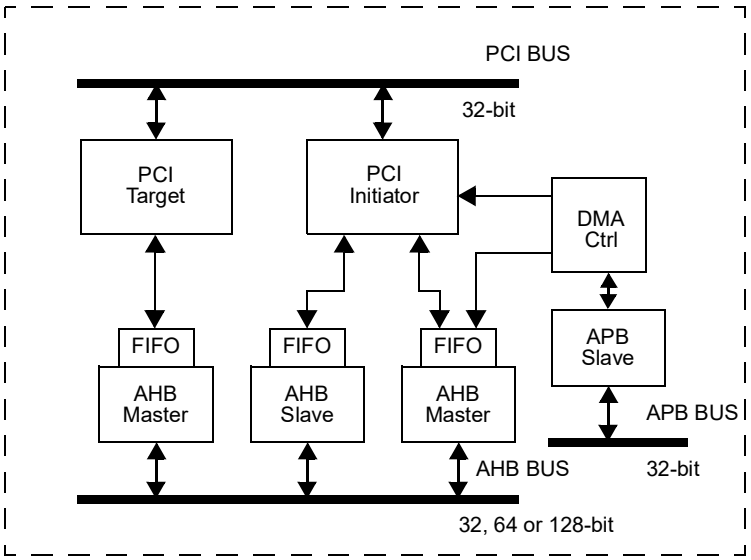


Figure 162. Block diagram

67.2 Configuration

The core has configuration registers located both in PCI Configuration Space (Compliant with the 2.3 PCI Local Bus Specification) and via an AMBA APB slave interface (for core function control and DMA control). This section defines which configuration options that are implemented in the PCI configuration space together with a list of capabilities implemented in the core. For a more detailed description of the core registers and DMA controller registers, see section Registers.

67.2.1 Configuration & Capabilities

Which of the core capabilities that are implemented is configured through VHDL generics at core instantiation. The implemented configuration can be determined by reading the Status & Capability register accessible via the APB slave interface.

- The PCI vendor and device ID is set with the VHDL generic *vendorid* and *deciceid*.
- The PCI class code and revision ID is set with the VHDL generic *classcode* and *revisionid*.
- 32-bit PCI initiator interface is implemented when the VHDL generic *master* is enabled.
- 32-bit PCI target interface is implemented when the VHDL generic *target* is enabled.

- DMA controller is implemented when the VHDL generic *dma* is enabled.
- The depth and number of FIFOs is configured with the VHDL generic *fifo_depth* and *fifo_count*.
- PCI BARs. The default size and number of BARs implemented is configured with the VHDL generic *bar0* to *bar5*.
- User defined register in Extended PCI Configuration Space can be enabled with the VHDL generic *ext_cap_pointer*.
- Device interrupt generation is enabled with the VHDL generic *deviceirq*.
- PCI interrupt sampling and forwarding is enabled with the VHDL generic *hostirq*.
- Support for two PCI functions is enabled with the VHDL generic multifunc.

67.2.2 PCI Configuration Space

The core implements the following registers in the PCI Configuration Space Header. For more detailed information regarding each field in these registers please refer to the PCI Local Bus Specification.

*Table 1021.*GRPCI2: Implemented registers in the PCI Configuration Space Header

PCI address offset	Register
0x00	Device ID, Vendor ID
0x04	Status, Command
0x08	Class Code, Revision ID
0x0C	BIST, Header Type, Latency Timer, Cache Line Size
0x10 - 0x24	Base Address Registers
0x34	Capabilities Pointer
0x3C	Max_Lat, Min_Gnt, Interrupt Pin, Interrupt Line

67.2.2.1 Device ID and Vendor ID register

*Table 1022.*0x00-0x10 - Device ID and Vendor ID register

31 16	15 0
Device ID	Vendor ID
*	*
r	r

31 : 16 Device ID, Set by the *deviceid* VHDL generic.

15 : 0 Vendor ID, Set by the *vendorid* VHDL generic.

67.2.2.2 Status and Command Register

*Table 1023.*G0x04 - STSCMD - Status and Command register

31	30	29	28	27	26 25	24	23	22	21	20	19	18 11	10	9	8	7	6	5	4	3	2	1	0
DPE	SSE	RMA	RTA	STA	DEVSEL timing	MDPE	FBBC	RES	66MHz	CL	IS	RESERVED	ID	Not Imp	SE	RES	PER	Not Imp	MWI	Not Imp	BM	MS	Not Imp
0	0	0	0	0	01	0	0	0	*	1	0	0	0	0	0	0	0	0	0	0	0	0	0
wc	wc	wc	wc	wc	r	r	r	r	r	r		r	rw	r	rw	r	rw	r	rw	r	rw	rw	r

31 Detected Parity Error

30 Signaled System Error

29 Received Master Abort

28 Received Target Abort

27 Signaled Target Abort

26: 25 DEVSEL timing, Returns “01“ indicating medium

24 Master Data Parity Error

23 Fast Back-to-Back Capable, Returns zero. (Read only)

22 RESERVED

21 66 MHz Capable (Read only)
NOTE: In this core this bit has been defined as the status of the M66EN signal rather than the capability of the core. For a 33 MHz design, this signal should be connected to ground and this status bit will have the correct value of ‘0’. For a 66 MHz design, this signal is pulled-up by the backplane and this status bit will have the correct value of ‘1’. For a 66 MHz capable design inserted in a 33 MHz system, this bit will then unfortunate only indicate a 33 MHz capable device.

20 Capabilities List, Returns one (Read only)

19 Interrupt Status (Read only)

18: 11 RESERVED

10 Interrupt Disable

9 NOT IMPLEMENTED, Returns zero.

8 SERR# Enable

7 RESERVED

6 Parity Error Response

5 NOT IMPLEMENTED, Returns zero.

4 Memory Write and Invalidate Enable

3 NOT IMPLEMENTED, Returns zero.

2 Bus Master

1 Memory Space

0 NOT IMPLEMENTED, Returns zero.

67.2.2.3 Class Code and Revision ID Register

*Table 1024.*0x08 - CCRID - Class Code and Revision ID register

31 — 8	7 — 0
Class Code	Revision ID
*	*
r	r

31 : 8 Class Code, Set by the *classcode* VHDL generic.

7 : 0 Revision ID, Set by the *revisionid* VHDL generic.

67.2.2.4 BIST, Header Type, Latency Timer, and Cache Line Size Register

*Table 1025.*0x0C- CCFG - BIST, Header Type, Latency Timer, and Cache Line Size register

31 — 24	23 — 16	15 — 8	7 — 0
BIST	Header Type	Latency Timer	Cache Line Size
0	0	0	0
r	r	rw	rw

31 : 24 NOT IMPLEMENTED, Returns zeros

23 : 16 Header Type, Returns 00

15 : 8 Latency Timer, All bits are writable.

7 : 0 NOT IMPLEMENTED, Returns zero.

67.2.2.5 Base Address Registers

*Table 1026.*0xC0-0x24 - BASEADDR - Base Address Registers

31 — 4	3	2 — 1	0
Base Address	PF	Type	MS
0	*	0	0
rw	r	r	r

31 : 4 Base Address. The size of the BAR is determine by how many of the bits (starting from bit 31) are implemented. Bits not implemented returns zero.

3 Prefetchable, Returns zero indicating non-prefetchable.

2 : 1 Type, Returns zero.

0 Memory Space Indicator

67.2.2.6 Capabilities Pointer Register

*Table 1027.*0x34 - CAPP - Capabilities Pointer Register

31 — 8	7 — 0
RESERVED	Capabilities Pointer
0	*
r	r

31 : 8 RESERVED

7 : 0 Capabilities Pointer. Indicates the first item in the list of capabilities of the Extended PCI Configuration Space. This offset is set with the VHDL generic *cap_pointer*.

67.2.2.7 Max_Lat, Min_Gnt, Interrupt Pin and Interrupt Line Register

*Table 1028.*0x3C - CCFG2 - Max_Lat, Min_Gnt, Interrupt Pin and Interrupt Line register

31 24	23 16	15 8	7 0
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line
0	0	*	0
r	r	r	rw

31 : 24 NOT IMPLEMENTED, Returns zero

23 : 16 NOT IMPLEMENTED, Returns zero

15 : 8 Interrupt Pin, Indicates INTA# when VHDL generic *deviceirq* is 1, otherwise zero is returned (Read only)

7 : 0 Interrupt Line

67.2.3 Extended PCI Configuration Space

This section describes the first item in the list of capabilities implemented in the Extended PCI Configuration Space. This capability is core specific and contains the PCI to AMBA address mapping and the option to change endianness of the PCI bus.

When user defined capability list items are implemented, the next pointer defines the offset of this list item. The AMBA address mapping for these registers can be accessed in the core specific item (first list item). The registers implemented in this AMBA address range must be compliant to the capability list items defined in the 2.3 PCI Local Bus Specification.

*Table 1029.*GRPCI2: Internal capabilities of the Extended PCI Configuration Space

PCI address offset (with the Capabilities pointer as base)	**Register**
0x00	Length, Next Pointer, ID
0x04 - 0x18	PCI BAR to AHB address mapping
0x1C	Extended PCI Configuration Space to AHB address mapping
0x20	AHB IO base address and PCI bus config (endianness switch)
0x24 - 0x38	PCI BAR size and prefetch
0x3C	AHB master burst limit

67.2.3.1 Length, Next Pointer and ID

Table 1030. Length, Next pointer and ID (address offset 0x00)

31 24	23 16	15 8	7 0
RESERVED	Length	Next Pointer	Capability ID

31 : 24 RESERVED.

23 : 16 Length, Returns 0x40. (Read only)

15 : 8 Pointer to the next item in the list of capabilities.This offset is set with the VHDL generic *ext_cap_pointer*. (Read only)

7 : 0 Capability ID, Returns 0x09 indicating Vendor Specific. (Read only)

67.2.3.2 PCI BAR to AHB Address Mapping Register

Table 1031. PCI BAR to AHB address mapping register (address offset 0x04 - 0x18)

31 0
PCI BAR to AHB address mapping

31 : 0 32-bit mapping register for each PCI BAR. Translate an access to a PCI BAR to a AHB base address. The size of the BAR determine how many bits (starting form bit 31) are implemented. Bits non implemented returns zero.

67.2.3.3 Extended PCI Configuration Space to AHB Address Mapping Register

Table 1032. Extended PCI Configuration Space to AHB address mapping register (address offset 0x1C)

31 8	7 0
Extended PCI Configuration Space to AHB address mapping	RESERVED

31 : 8 Translates an access to the Extended PCI Configuration Space (excluding the address range for the internal register located in this configuration space) to a AHB address.

7 : 0 RESERVED

67.2.3.4 AHB IO Base Address and PCI Bus Config

Table 1033. AHB IO base address and PCI bus config (endianness register) (address offset 0x20)

31 20	19 1	0
AHB IO base address	RESERVED	DISEN	Endian

31 : 8 Base address of the AHB IO area. (Read only, not replicated for each PCI function)

19 : 2 RESERVED

1 Target access discard time out enable. When set to ‘1’, the target will discard a pending access if no retry of the access is detected during 2**15 PCI clock cycles. (Not replicated for each PCI function)

0 PCI bus endianness switch. 1: defines the PCI bus to be little-endian, 0: defines the PCI bus to be big-endian. Reset value is set be the conv_endian VHDL generic. (Not replicated for each PCI function)

67.2.3.5 PCI BAR Size and Prefetch Register

Table 1034. PCI BAR size and prefetch register (address offset 0x24 - 0x38)

31 : 4	3	2 : 1	0
PCI BAR size mask	Pre	RESERVED	Type

31 : 4	A size mask register for each PCI BAR. When bit[n] is set to '1' bit[n] in the PCI BAR register is implemented and can return a non-zero value. All bits from the lowest bit set to '1' up to bit 31 need to be set to '1'. When bit 31 is '0', this PCI BAR is disabled. The number of implemented bits in this field depends in the VHDL generic barminsize. The minimal size of the BAR is not allowed to be smaller than the internal FIFO.
3	Prefetch bit in PCI BAR register. When the EXTCAP.PF bit is valid (STATCAP.ECAP = '1') and set to '1', this configuration option change the access pattern towards the AMBA bus. When this bit is set to '1', burst accesses towards the target BAR will prefetch data on the AMBA bus. When this bit is set to '0', only single accesses will be generated towards the AMBA bus and no prefetching will be done. When the EXTCAP.PF bit is not valid (STATCAP.ECAP = '0') or set to '0', this configuration option only change the prefetchable bit in the BAR register and has no affect on the AMBA access pattern.
2 : 1	RESERVED
0	BAR type. 0 = Memory BAR, 1 = IO BAR

67.2.3.6 AHB Master Prefetch Burst Limit

Table 1035. AHB master burst limit (address offset 0x3C)

31	30 : 16	15 : 0
SRF	RESERVED	Burst length

31	Store Read FIFO. When set to 1, the prefetched FIFO will be stored until the next PCI access when the PCI target terminates the access with disconnect without data.
30 : 16	RESERVED
15 : 0	Maximum number of beats - 1 in the burst. (Maximum value is 0xFFFF => 0x10000 beats => 65kB address)

67.2.4 Multi-Function

The core supports up to two PCI functions starting from function 0. Each function has its own PCI configuration space located at offset 0x0 for function 0 and offset 0x100 for function 1. Some registers in the Extended PCI configuration space is shared between all functions. All functions also share the same Vendor ID.

67.3 Operation

67.3.1 Access support

The core supports both single and burst accesses on the AMBA AHB bus and on the PCI bus. For more information on which PCI commands that are supported, see the PCI target section and for burst limitations see the Burst section.

67.3.2 FIFOs

The core has separate FIFOs for each data path: PCI target read, PCI target write, PCI master read, PCI master write, DMA AHB-to-PCI, and DMA PCI-to-AHB. The number and depth of the FIFOs for each data path is configurable by VHDL generics.

67.3.3 Byte enables and byte twisting (endianness)

The core has the capability of converting endianness between the two busses. This means that all byte lanes can be swapped by the core as shown in figure below.

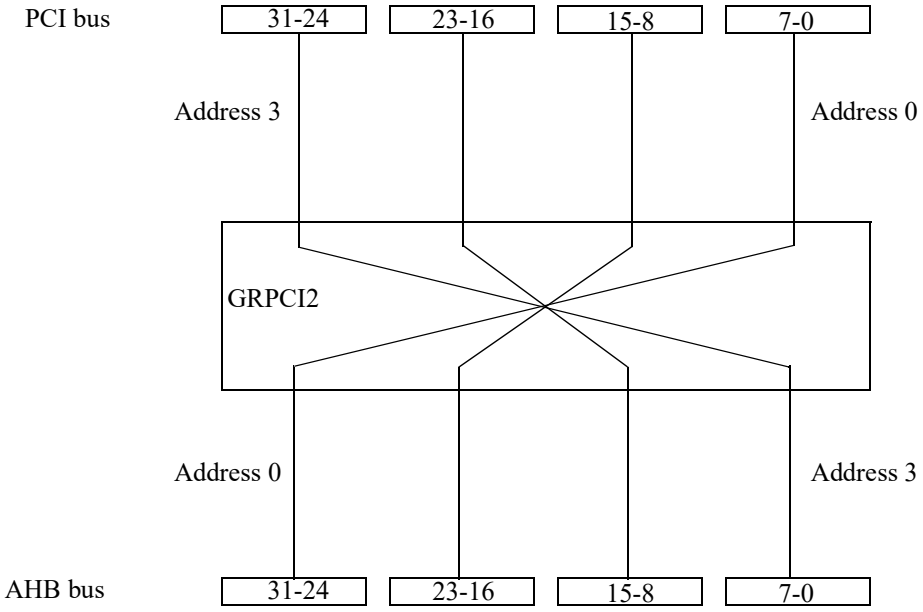


Figure 163. GRPCI2 byte twisting

Table 1036 defines the supported AHB address/size and PCI byte enable combinations.

Table 1036. AHB address/size <=> PCI byte enable combinations.

AHB HSIZE	AHB ADDRESS[1:0]	Little-endian CBE[3:0]	Big-endian CBE[3:0]
00 (8-bit)	00	1110	0111
00 (8-bit)	01	1101	1011
00 (8-bit)	10	1011	1101
00 (8-bit)	11	0111	1110
01 (16-bit)	00	1100	0011
01 (16-bit)	10	0011	1100
10 (32-bit)	00	0000	0000

As the AHB bus in GRLIB is defined as big-endian, the core is able to define the PCI bus as little-endian (as defined by the PCI Local Bus Specification) with endianness conversion or define the PCI bus as big-endian without endianness conversion.

The endianness of the PCI bus is configured via the core specific Extended PCI Configuration Space. The default value is set by a VHDL generic *conv_endian*.

67.3.4 PCI configuration cycles

Accesses to PCI Configuration Space are not altered by the endianness settings. The PCI Configuration Space is always defined as little-endian (as specified in the PCI Local Bus Specification). This

means that the PCI target does not change the byte order even if the endianness conversion is enabled and the PCI master always converts PCI Configuration Space accesses to little-endian.

Data stored in a register in the PCI Configuration Space as 0x12345678 (bit[31:0]) is transferred to the AHB bus as 0x78563412 (bit[31:0]). This means that non-8-bit accesses to the PCI Configuration Space must be converted in software to get the correct byte order.

67.3.5 Memory and I/O accesses

Memory and I/O accesses are always affected by the endianness conversion setting. The core should define the PCI bus as little-endian in the following scenarios: When the core is the PCI host and little-endian peripherals issues DMA transfers to host memory. When the core is a peripheral device and issues DMA transfers to a little-endian PCI host.

67.3.6 Bursts

PCI bus: The PCI target terminates a burst when no FIFO is available (the AMBA AHB master is not able to fill or empty the FIFO fast enough) or when the burst reached the length specified by the "AHB master burst limit" register. This register defines a boundary which a burst can not cross i.e. when set to 0x400 beats (address boundary at 4kB) the core only prefetch data up to this boundary and then terminates the burst with a disconnect.

The PCI master stops the burst when the latency timer times out (see the PCI Local Bus Specification for information on the latency timer) or for reads when the burst reaches the limit defined by "PCI master prefetch burst limit" register (if AHB master performing the access is unmasked). If the master is masked in this register, the limit is set to 1kB. The PCI master do not prefetch data across this address boundary.

AHB bus: As long as FIFOs are available for writes and data in a FIFO is available for read, the AHB slave do not limit the burst length. The burst length for the AHB master is limited by the FIFO depth. The AHB master only burst up to the FIFO boundary. Only linear-incremental burst mode is supported.

DMA: DMA accesses are not affected by the "AHB master prefetch burst limit" register or the "PCI master prefetch burst limit" register.

All FIFOs are filled starting at the same word offset as the bus access (i.e. with a FIFO of depth 8 words and the start address of a burst is 0x4, the first data word is stored in the second FIFO entry and only 7 words can be stored in this FIFO).

67.3.7 Host operation

The core provides a system host input signal that must be asserted (active low) for PCI system host operations. The status of this signal is available in the Status & Capability register accessible via the APB slave interface. The device is only allowed to generate PCI configuration cycles when this signal is asserted (device is the system host).

For designs intended to be host or peripherals only the PCI system host signal can be tied low or high internally in the design. For multi-purpose designs it should be connected to a pin. The PCI Industrial Computer Manufacturers Group (PCIMG) cPCI specification uses pin C2 on connector P2 for this purpose. The pin should have a pull-up resistor since peripheral slots leave it unconnected.

An asserted PCI system host signal makes the PCI target respond to configuration cycles when no IDSEL signal is asserted (none of AD[31:11] are asserted). This is done for the PCI master to be able to configure its own PCI target.

67.4 PCI Initiator interface

The PCI master interface is accessible via the AMBA AHB slave interface. The AHB slave interface occupies 1MB to 2GB of the AHB memory address space and 128kB to 256kB of AHB I/O address

space. An access to the AHB memory address area is translated to a PCI memory cycle. An access to the first 64kB of the AHB IO area is translated to a PCI I/O cycle. The next 64kB are translated to PCI configuration cycles. When the PCI trace buffer is implemented, it is accessible via the last 128kB of the AHB I/O area.

67.4.1 Memory cycles

A single read access to the AHB memory area is translated into a PCI memory read access, while a burst read translates into a PCI memory read multiple access. A write to this memory area is translated into a PCI write access.

The address translation is determined by AHB master to PCI address mapping registers accessible via the APB slave interface. Each AHB master on the AMBA AHB bus has its own mapping register. These registers contain the MSBs of the PCI address.

When the PCI master is busy performing a transaction on the PCI bus and not able to accept new requests, the AHB slave interface will respond with an AMBA RETRY response. This occurs on reads when the PCI master is fetching the requested data to fill the read FIFO or on writes when no write FIFO is available. This means that all masters on the AMBA bus accessing the AHB slave interface must be round-robin arbitrated without prioritization to avoid deadlock situations.

67.4.2 I/O cycles

Accesses to the lowest 64kB of the AHB I/O address area are translated into PCI I/O cycles. The address translation is determined by the “AHB to PCI mapping register for PCI I/O”. This register sets the 16 MSB of the PCI address. The “AHB to PCI mapping register for PCI I/O” is accessible via the APB slave interface. When the “IB” (PCI IO burst) bit in the Control register (accessible via the APB slave interface) is cleared, the PCI master does not perform burst I/O accesses.

67.4.3 Configuration cycles

Accesses to the second 64kB address block (address offset range 64kB to 128kB) of the AHB I/O address area is translated into PCI configuration cycles. The AHB address is translated into PCI configuration address different for type 0 and type 1 PCI configuration cycles. When the “bus number” field in the control register (accessible via the APB slave interface) is zero, type 0 PCI configuration cycles is issued. When the “bus number“ field is non-zero, type 1 PCI configuration cycles are issued to the PCI bus determine by this field. The AHB I/O address mapping to PCI configuration address for type 0 and type 1 PCI configuration cycles is defined in table 1037 and table 1038.

Only the system host is allowed to generate PCI configuration cycles. The core provides a system host input signal that must be asserted (active low) for PCI system host operations. The status of this signal is available in the Status & Capability register accessible via the APB slave interface.When the “CB” (PCI Configuration burst) bit in the Control register (accessible via the APB slave interface) is cleared, the PCI master does not perform burst configuration accesses.

67.4.3.1 Mapping of AHB I/O address to PCI Configuration Cycle, type 0

Table 1037. GRPCI2 Mapping of AHB I/O address to PCI configuration cycle, type 0

31 16	15 11	10 8	7 2	1 0
AHB ADDRESS MSB	IDSEL	FUNC	REGISTER	BYTE

31: 16	AHB address MSBs: Not used for PCI configuration cycle address mapping.
15: 11	IDSEL: This field is decoded to drive PCI AD[IDSEL+10]. Each of the signals AD[31:11] are suppose to be connected (by the PCI back plane) to one corresponding IDSEL line.
10: 8	FUNC: Selects function on a multi-function device.
7: 2	REGISTER: Used to index a PCI DWORD in configuration space.
1: 0	BYTE: Used to set the CBE correctly for non PCI DWORD accesses.

67.4.3.2 Mapping of AHB I/O address to PCI Configuration Cycle, type 1

Table 1038. GRPCI2 Mapping of AHB I/O address to PCI configuration cycle, type 1

31 16	15 11	10 8	7 2	1 0
AHB ADDRESS MSB	DEVICE	FUNC	REGISTER	BYTE

31: 16	AHB address MSBs: Not used for PCI configuration cycle address mapping.
15: 11	DEVICE: Selects which device on the bus to access.
10: 8	FUNC: Selects function on a multi-function device.
7: 2	REGISTER: Used to index a PCI DWORD in configuration space.
1: 0	BYTE: Used to set the CBE correctly for non PCI DWORD accesses.

67.4.4 Error handling

When a read access issued by the PCI master is terminated with target-abort or master-abort, the AHB slave generates an AMBA ERROR response when the “ER” bit in the control register is set. When the “EI“ bit in the control register is set, an AMBA interrupt is generated for the error. The interrupt status field in the control register indicates the cause of the error.

67.4.5 Bus parking

The PCI initiator supports bus parking and will drive the PCI bus to a defined state when granted without requesting the bus. In systems with one single initiator the grant input to the PCI interface can be constantly asserted.

67.5 PCI Target interface

The PCI Target occupies memory areas in the PCI address space corresponding to the BAR registers in the PCI Configuration Space. Each BAR register (BAR0 to BAR5) defines the address allocation in the PCI address space. The size of each BAR is set by the “BAR size and prefetch” registers accessible via the core specific Extended PCI Configuration Space. The size of a BAR can be determined by checking the number of implemented bits in the BAR register. Non-implemented bits returns zero and are read only. The size of the BAR is not allowed to be smaller then the size of the internal FIFO.

67.5.1 Supported PCI commands

These are the PCI commands that are supported by the PCI target.

- **PCI Configuration Read/Write:** Burst and single access to the PCI Configuration Space. These accesses are not transferred to the AMBA AHB bus except for the access of the user defined capability list item in the Extended PCI Configuration Space.

- **Memory Read:** A read command to the PCI memory BAR is transferred to a single read access on the AMBA AHB bus.
- **Memory Read Multiple, Memory Read Line:** A read multiple command to the prefetchable* PCI memory BAR is transferred to a burst access on the AMBA AHB bus. This burst access prefetch data to fill the maximum amount of data that can be stored in the FIFO. When the BAR is not defied as prefetchable*, only signal accesses will be generated on the AMBA AHB bus and no prefetching is done.

  A BAR is defined as prefetchable* when the prefetchable bit in the BAR register is set to '1' and the register bit STATCAP.ECAP = '1' and EXTCAP.PF = '1'. In the case when STATCAP.ECAP = '0' or EXTCAP.PF = '0' all bars are defined as prefetchable independent of the configuration bit in the BAR register).
- **Memory Write, Memory Write and Invalidate:** These command are handled similarly and are transferred to the AMBA AHB bus as a single or burst access depending on the length of the PCI access (a single or burst access).
- **IO Read:** A read command to the PCI IO BAR is transferred to a single read access on the AMBA AHB bus.
- **IO Write:** A write command to the PCI IO BAR is transferred to the AMBA AHB bus as a single access.

67.5.2 Implemented PCI responses

The PCI target can terminate a PCI access with the following responses.

- **Retry:** This response indicates the PCI target is busy by either fetching data for the AMBA AHB bus on a PCI read or emptying the write FIFO for a PCI write. A new PCI read access will always be terminated with a retry at least one time before the PCI target is ready to deliver data.
- **Disconnect with data:** Terminate the transaction and transfer data in the current data phase. This occurs when the PCI master request more data and the next FIFO is not yet available or for a PCI burst access with the Memory Read command.
- **Disconnect without data:** Terminate the transaction without transferring data in the current data phase. This occurs if the CBE change within a PCI burst write.
- **Target Abort:** Indicates that the current access caused an internal error and the target is unable to finish the access. This occurs when the core receives a AMBA AHB error during a read operation.

67.5.3 Supported byte-enables (CBE)

The PCI-target only supports aligned 8-, 16-, and 32-bit accesses. The supported combinations of CBE are 0000, 1110, 1101, 1011, 0111, 1100, 0011. All other combinations of CBE are interpret as a 32-bit access (CBE = 0000) except for writes with CBE set to 1111, which is treated as a no-operation (no write will be performed).

67.5.4 PCI to AHB translation

Each PCI BAR has translation register (mapping register) to translate the PCI access to a AMBA AHB address area. These mapping registers are accessible via the core specific Extended PCI Configuration Space. The number of implemented bits in these registers correspond to the size of (and number of implemented bits in) the BARs registers.

67.5.5 PCI system host signal

When the PCI system host signal is asserted the PCI target responds to configuration cycles when no IDSEL signal is asserted (none of AD[31:11] are asserted). This is done for the PCI master, in a system host position, to be able to configure its own PCI target.

67.5.6 Error handling

The PCI target terminates the access with target-abort when the PCI target requests data from the AHB bus which results in an error response on the AHB bus. Because the writes to the PCI target is posted, no error is reported on write AHB errors.

When a PCI master is terminated with a retry response it is mandatory for that master to retry this access until the access is completed or terminated with target-abort. If the master never retries the access, the PCI target interface would be locked on this access and never accept any new access. To recover from this situation, the PCI target has a option to discard an access if it is not retried within 2**15 clock cycles. This discard time out can be enabled via the “AHB IO base address and PCI bus config” located in the core specific Extended PCI Configuration Space.

67.6 DMA Controller

The DMA engine is descriptor base and uses two levels of descriptors.

67.6.1 DMA channel

The first level is a linked list of DMA channel descriptors. Each descriptor has a pointer to its data descriptor list and a pointer to the next DMA channel. The last DMA channel descriptor should always points to the first DMA channel for the list to be a closed loop. The descriptor needs to be aligned to 4 words (0x10) in memory and have the following structure.

*Table 1039.*GRPCI2: DMA channel descriptor structure

Descriptor address offset	Descriptor word
0x00	DMA channel control
0x04	Next DMA channel (32-bit address to next DMA channel descriptor).
0x08	Next data descriptor in this DMA channel (32-bit address to next data descriptor).
0x0C	RESERVED

67.6.1.1 DMA Channel Control

Table 1040. GRPCI2 DMA channel control

31	25	24	22	21	20	19	16	15	0
EN	RESERVED	CID		Type		RESERVED		Data descriptor count	

31	Channel descriptor enable (for version < 2, this bit should always be set to ‘1’).
30: 25	RESERVED
24: 22	Channel ID. Each DMA channel needs a ID to determine the source of a DMA interrupt.
21: 20	Descriptor type. 01 = DMA channel descriptor.
19: 16	RESERVED
15: 0	Maximum number of data descriptors to be executed before moving to the next DMA channel. 0 indicates that all data descriptors should be executed before moving to the next DMA channel.

The number of enabled DMA channels must be stored in the “Number of DMA channels“ field in the DMA control register accessible via the APB slave interface.

67.6.2 Data descriptor

The second descriptor level is a linked list of data transfers. The last descriptor in this list needs to be a disabled descriptor. To add a new data transfer, this disabled descriptor is updated to reflect the data transfer and to point to a new disabled descriptor. The control word in the descriptor should be updated last to enable the valid descriptor. To make sure the DMA engine reads this new descriptor, the enable bit in the DMA control register should be updated. The descriptor needs to be aligned to 4 words (0x10) in memory and have the following structure.

*Table 1041.*GRPCI2: DMA data descriptor structure

Descriptor address offset	Descriptor word
0x00	DMA data control
0x04	32-bit PCI start address
0x08	32-bit AHB start address
0x0C	Next data descriptor in this DMA channel (32-bit address to next data descriptor).

67.6.2.1 DMA Data Control

Table 1042. GRPCI2 DMA data control

31	30	29	28	22	21	20	19	18	16	15	0
EN	IE	DR	BE	RESERVED		Type	ER	RESERVED		LEN	

31	Data descriptor enable.
30	Interrupt generation enable.
29	Transfer direction. 0: PCI to AMBA, 1: AMBA to PCI.
28	PCI bus endianness switch. 1: defines the PCI bus to be little-endian for this transfer, 0: defines the PCI bus to be big-endian for this transfer.
27: 22	RESERVED (Must be set to zero)
21: 20	Descriptor type. 00 = DMA data descriptor.
19	Error status
18: 16	RESERVED
15: 0	Transfer length. The number of word of the transfer is (this field)+1.

67.6.3 Data transfer

The DMA engine starts by reading the descriptor for the first DMA channel. If the DMA channel is enabled the first data descriptor in this channel is read and executed. When the transfer is done the data descriptor is disabled and status is written to the control word. If no error occurred during the transfer, the error bit is not set and the transfer length field is unchanged. If the transfer was terminated because of an error, the error bit is set in the control word and the length field indicates where in the transfer the error occurred. If no error has occurred, the next data descriptor is read and executed. When a disabled data descriptor is read or the maximum number of data descriptors has been executed, the DMA channel descriptor is updated to point to the next data descriptor and the DMA engine moves on to the next DMA channel.

When a disabled channel descriptor is read, the DMA controller will move on to the next DMA channel without reading in any data descriptors form the disabled channel (Only applies to version 2 of the core).

When the DMA is disabled (via the APB interface), the channel descriptor is updated to point to the next data descriptor (Only applies to version 2 of the core).

The DMA engine will stop when an error is detected or when no enabled data descriptors is found. The error type is indicated by bit 7 to bit 11 in the DMA control register. The error type bits must be cleared (by writing '1') before the DMA can be reenabled.

67.6.4 Interrupt

The DMA controller has an interrupt enable bit in the DMA control register (accessible via the APB slave interface) which enables interrupt generation.

Each data descriptor has an interrupt enable bit which determine if the core should generate a interrupt when the descriptor has been executed.

The VHDL generic *irqmode* determines if the DMA engine assert the same interrupt as the PCI core or the DMA uses the irq signal following the PCI core interrupt, see the IRQ mode field in the Status and Capability register for irq routing information.

67.7 PCI trace buffer

67.7.1 Trace data

The data from the trace buffer is accessible in the last 128 kB block of the AHB I/O address area. Each 32-bit word in the first 64kB of this block represents a sample of the AD PCI signal. The second 64kB of the block is the corresponding PCI control signal. Each 32-bit word is defined in table 1043.

Note that it is not supported to access the trace buffer data at the same time the AMBA AHB slave interface is used also for PCI master accesses. Doing this could return corrupted data for the PCI master access.

67.7.1.1 PCI Control Signal Trace (32-bit word)

Table 1043. GRPCI2 PCI control signal trace (32-bit word)

31 20	19 16	15	14	13	12	11	10	9	8	7	6	5	4	3	0
RESERVED	CBE[3:0]	FRAME	IRDY	TRDY	STOP	DEVSEL	PAR	PERR	SERR	IDSEL	REQ	GNT	LOCK	RST	RES

31: 20	RESERVED
19: 3	The state of the PCI control signals.
2: 0	RESERVED

67.7.2 Triggering function

The core can be programmed to trigger on any combination of the PCI AD and PCI Control signals by setting up the desired pattern and mask in the PCI trace buffer registers accessible via the APB slave interface. Each bit the PCI AD signal and any PCI control signal can be masked (mask bit equal to zero) to always match the triggering condition.

The "Trig count" field in the "PCI trace buffer: counter & mode" register defines how many times the trigger condition should occur before the trace buffer disarms and eventually stops sampling. The number of samples stored after the triggering condition occurs defines by the "Delayed stop" + 2.

To start sampling, the trace buffer needs to be armed by writing one to the start bit in the "PCI trace buffer: Control" register. The state of the trace buffer can be determine by reading the Armed and Enable/Running bit in the control register. When the Armed bit is set, the triggering condition has not occurred. The Enable/Running bit indicates that the trace buffer still is storing new samples. When the delayed stop field is set to a non zero value, the Enabled bit is not cleared until all samples are stored

in the buffer). The trace buffer can also be disarmed by writing the “stop” bit in the “PCI trace buffer: control” register.

When the trace buffer has been disarmed, the “trig index” in the “PCI trace buffer: control” register is updated with index of trace entry which match the triggering condition. The address offset of this entry is the value of the “trig index“ field times 4.

67.7.3 Trace Buffer APB interface

A separate APB register can optionally be enabled for access of the PCI trace buffer. The register layout is the same as the core APB interface but only registers related to the PCI trace buffer is accessable. The trace buffer data is located at offset 0x20000 for PCI AD and offset 0x30000 for PCI control signals.

67.8 Interrupts

The core is capable of sampling the PCI INTA-D signals and forwarding the interrupt to the APB bus. The PCI INTA-D signals can be connected to one APB irq signal or to 4 different irq signals. This is configured by the VHDL generic *irqmode*. The “host INT mask” field in the control register is used only for sampling the valid PCI INT signal.

The core supports PCI interrupt generation. For single function configuration the dirq signal is sampled and forwarded to the PCI INTA signal. For a multi function (and multi interrupt) configured device, each bit of the dirq signal is connected to one of the PCI INTA..D signal (dirq[0] => INTA, dirq[1] => INTB, ...). The core has a mask bit (the “device INT mask“ field in the control register) for each bit in the dirq vector. The core also has a PCI interrupt force bit in the control register to be able to force the PCI INT asserted. For a multi interrupt configuration the PCI interrupt force bit is masked by the “device INT mask” to be able to assert all PCI INT signals separately.

When the system error PCI signal (SERR) is asserted the core sets the system error bit in the “core interrupt status” field in the Status & Capability register. If the system interrupts is enabled the core will also generate a interrupt on the APB bus.

67.9 Registers

The core is configured via registers mapped into the APB memory address space.

*Table 1044.*GRPCI2: APB registers

APB address offset	Register
0x00	Control
0x04	Status & Capability (Read only)
0x08	PCI master prefetch burst limit
0x0C	AHB to PCI mapping for PCI IO
0x10	DMA Control & Status
0x14	DMA descriptor base
0x18	DMA channel active (read only)
0x1C	RESERVED
0x20 - 0x34	PCI BAR to AHB address mapping (Read only)
0x38	Extended Capability register
0x3C	RESERVED
0x40 - 0x7C	AHB master to PCI memory address mapping
0x80	PCI trace buffer: control & status
0x84	PCI trace buffer: counter & mode
0x88	PCI trace buffer: AD pattern
0x8C	PCI trace buffer: AD mask
0x90	PCI trace buffer: Ctrl signal pattern
0x94	PCI trace buffer: Ctrl signal mask
0x98	PCI trace buffer: AD state
0x9C	PCI trace buffer: Ctrl signal state

67.9.1 Control Register

Table 1045. 0x00 - CTRL - Control register

31	30	29	28	27	26	25	24	23 16	15 12	11	10	9	8	7 4	3 0
RE	MR	TR	R	SI	PE	ER	EI	Bus Number	RESERVED	DFA	IB	CB	DIF	Device INT mask	Host INT mask
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rw	rw	rw	r	rw	rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw

31 PCI reset. When set, the PCI reset signal is asserted. Needs to be cleared to deassert PCI reset.

30 PCI master reset. Set to reset the cores PCI master. This bit is self clearing.

29 PCI target reset. Set to reset the cores PCI target. This bit is self clearing.

28 RESERVED

27 When set, Interrupt is enabled for System error (SERR)

26 When set, AHB error response is enabled for Parity error

25 When set, AHB error response is enabled for Master and Target abort.

24 When set, Interrupt is enabled for Master and Target abort and Parity error.

23: 16 When not zero, type 1 configuration cycles is generated.This field is also used as the Bus Number in type 1 configuration cycles.

15: 12 RESERVED

11 Disable internal AHB-slave / DMA fair arbitration (DFA). When this bit is set, the arbitration is done when the current transfer has complete.

10 When set, burst accesses may be generated by the PCI master for PCI IO cycles

9 When set, burst accesses may be generated by the PCI master for PCI configuration cycles.

8 Device interrupt force. When set, a PCI interrupt is forced.

7: 4 Device interrupt mask. When bit[n] is set dirq[n] is unmasked

3: 0 Host interrupt mask
bit[3] = 1: unmask INTD.
bit[2] = 1: unmask INTC.
bit[1] = 1: unmask INTB.
bit[0] = 1: unmask INTA.

67.9.2 Status and Capability Register

Table 1046. 0x04 - STATCAP - Status and Capability register

31	30	29	28	27	26	25 24	23	22	21	20	19	18 12	11 8	7 5	4 2	1 0
Host	MST	TAR	DMA	DI	HI	IRQ mode	Trace	ECAP	FH	CFGDO	CFGER	Core interrupt status	Host interrupt status	RES	FDEPTH	FNUM
*	*	*	*	*	*	*	*	0	0	0	0	0	*	0	*	*
r	r	r	r	r	r	r	r	r	r	wc	wc	wc	r	r	r	r

31 When zero, the core is inserted in the System slot and is allowed to act as System Host.

30 Master implemented

29 Target implemented

28 DMA implemented

27 Device drives PCI INTA

26 Device samples PCI INTA..D (for host operations)

25: 24 APB IRQ mode
00: PCI INTA..D, Error interrupt and DMA interrupt on the same IRQ signal
01: PCI INTA..D and Error interrupt on the same IRQ signal. DMA interrupt on IRQ+1
10: PCI INTA..D on IRQ..IRQ+3. Error interrupt and DMA interrupt on IRQ.
11: PCI INTA..D on IRQ..IRQ+3. Error interrupt on IRQ. DMA interrupt on IRQ+4

23 PCI trace buffer implemented

Table 1046. 0x04 - STATCAP - Status and Capability register

22	Indicates of extended capability register (offset 0x38) is available. The extended capability register is only valid when this bit is '1'.
21	Fake device in system slot (Host). This bit should always be written with '0'. Only for debugging.
20	PCI configuration access done, PCI configuration error status valid.
19	Error during PCI configuration access
18: 12	Interrupt status: bit[6]: PCI target access discarded due to time out (access not retried for 2**15 PCI clock cycles) bit[5]: System error bit[4]: DMA interrupt bit[3]: DMA error bit[2]: Master abort. bit[1]: Target abort. bit[0]: Parity error.
11: 8	Host interrupt status bit[3] = 0: indicates that INTD is asserted. bit[2] = 0: indicates that INTC is asserted. bit[1] = 0: indicates that INTB is asserted. bit[0] = 0: indicates that INTA is asserted.
7: 5	RESERVED
4: 2	Words in each FIFO = 2**(FIFO depth)
1: 0	Number of FIFOs 2 = 2 FIFOs (ping-pong buffers) 3 = 3 FIFOs (ping-pong buffers) 0 = 4 FIFOs (ping-pong buffers)

67.9.3 Master Prefetch Burst Limit

Table 1047. 0x08 - BCIM - PCI master prefetch burst limit

31 24	23 16	15 8	7 0
AHB master unmask		RESERVED	Burst length
0		0	0xFF
rw		r	rw

31 : 16 When bit[n] is set, the prefetch burst of AHB master n is limited by the “Burst length” field.

15 : 8 RESERVED

7 : 0 Maximum number of beats - 1 in the burst. (Maximin value is 0xFF => 0x100 beats => 1kB address)

67.9.4 AHB to PCI Mapping for PCI IO

Table 1048. 0x0C - AHB2PCI - AHB to PCI mapping for PCI IO

31 16	15 0
AHB to PCI IO	RESERVED
0	0
rw	r

31 : 16 Used as the MSBs of the base address for a PCI IO access.

15 : 0 RESERVED

67.9.5 DMA Control and Status Register

Table 1049. 0x10 - DMACTRL - GRPCI2 DMA control and status register

31	30 - 20	19 12	11	10	9	8	7	6 4	3	2	1	0
SAFE	RES	CHIRQ	MA	TA	PE	AE	DE	Number of DMA channels	ACTIVE	DIS	IE	EN
1	0	0	0	0	0	0	0	0	0	0	0	0
rw	r	wc	wc	wc	wc	wc	wc	rw	r	rw	rw	rw

31 Safety guard for update of control fields. Needs to be set to ‘1’ for the control fields to be updated.

30 : 20 RESERVED

19 : 12 Channel IRQ status. Set to ‘1’ when a descriptor is configured to signal interrupt. bit[0] corresponds to the channel with ID 0, bit[1] corresponds to the channel with ID 1, ... Clear by writing ‘1’.

11 Master abort during PCI access. Clear by writing ‘1’

10 Target abort during PCI access. Clear by writing ‘1’

9 Parity error during PCI access. Clear by writing ‘1’

8 Error during AHB data access. Clear by writing ‘1’

7 Error during descriptor access. Clear by writing ‘1’.

6 : 4 Number of DMA channels (Guarded by bit[31], safety guard)

3 DMA is active (read only)

2 DMA disable/stop. Writing ‘1’ to this bit disables the DMA.

1 Interrupt enable (Guarded by bit[31], safety guard).

0 DMA enable/start. Writing ‘1’ to this bit enables the DMA.

67.9.6 DMA Descriptor Base Address (/ Active Descriptor) Register

Table 1050. 0x14 - DMABASE - DMA descriptor base address (/ Active Descriptor) register

31	0
DMA descriptor base address	
0	
rw	

31 : 0 Base address of the DMA descriptor table. When running, this register points to the active descriptor.

67.9.7 DMA Channel Active Register

Table 1051. 0x18 - DMACHAN - DMA channel active register

31	0
DMA descriptor base address	
0	
r	

31 : 0 Base address of the active DMA channel.

67.9.8 PCI BAR to AHB Address Mapping Register

Table 1052. 0x20-0x34 - PCI2AHB - PCI BAR to AHB address mapping register

31	0
PCI BAR to AHB address mapping	
0	
r	

31 : 0 32-bit mapping register for each PCI BAR. Translate an access to a PCI BAR to a AHB base address. This is a read-only view of the bar to AHB mapping registers in PCI configuration space (section 67.2.3.2).

67.9.9 Extended Capability Register

Table 1053. 0x38 - EXTCAP - Extended Capability register

31 30 29 28 27 26 25 24 23 22 21 20 19 18	12 11	8 7	5 4	2 1	0
RESERVED					PF
0					*
r					r

31: 1 RESERVED

0 BAR prefetchable bit affects AMBA access pattern.

67.9.10 AHB Master to PCI Memory Address Mapping Register

Table 1054. 0x40-0x7C - AHBM2PCI - AHB master to PCI memory address mapping register

31	0
AHB master to PCI memory address mapping	
0	
rw	

31 : 0 32-bit mapping register for each AHB master. Translate an access from a specific AHB master to a PCI base address. The size of the AHB slave address area determine how many bits (starting from bit 31) are implemented. Bits not implemented returns zero. The mapping register for AHB master 0 is located at offset 0x40, AHB master 1 at offset 0x44, and so on up to AHB master 15 at offset 0x7C. Mapping registers are only implemented for existing AHB masters.

67.9.11 PCI Trace Control and Status Register

Table 1055. 0x80 - TCTRC - PCI trace Control and Status register

31 16	15	14	13 12	11 4	3 2	1	0
TRIG INDEX	AR	EN	RES	DEPTH	RES	SO	SA
NR	0	0	0	*	0		
r	r	r	r	r	r	w	w

31: 16 Index of the first entry of the trace.

15 Set when trace buffer is armed (started but the trig condition has not occurred).

14 Set when trace buffer is running

13: 12 RESERVED

11: 4 Number of buffer entries = 2**DEPTH

3: 2 RESERVED

1 Stop tracing. (Write only)

0 Start tracing. (Write only)

67.9.12 PCI Trace Counter and Mode Register

Table 1056. 0x84 - TMODE - PCI trace counter and mode register

31 28	27 24	23 16	15 0
RES	Trace mode	Trig count	Delayed stop
0	0	0	0
r	rw	rw	rw

31: 28 RESERVED

27: 24 Tracing mode
00: Continuos sampling
01: RESERVED
10: RESERVED
11: RESERVED

23: 16 The number of times the trig condition should occur before the trace is disarmed.

15: 0 The number of entries stored after the trace buffer has been disarmed. (Should not be lager than number of buffer entries - 2).

67.9.13 PCI Trace AD Pattern Register

Table 1057. 0x88 - TADP - PCI trace AD pattern register

31 0
PCI AD pattern
NR
rq

31: 0 AD pattern to trig on

67.9.14 PCI Trace AD Mask Register

Table 1058. 0x8C - TADM - PCI trace AD mask register

31 0
PCI AD mask
NR
rw

31: 0 Mask for the AD pattern. When mask bit[n] = 0 pattern bit[n] will always be a match.

67.9.15 PCI Trace Ctrl Signal Pattern Register

Table 1059. 0x90 - TCP - PCI trace Ctrl signal pattern register

31 20	19 16	15	14	13	12	11	10	9	8	7	6	5	4	3 0
RESERVED	CBE[3:0]	FRAME	IRDY	TRDY	STOP	DEVSEL	PAR	PERR	SERR	IDSEL	REQ	GNT	LOCK	RST	RES
0	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	0
r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r

31: 20 RESERVED

19: 3 PCI Ctrl signal pattern to trig on

2: 0 RESERVED

67.9.16 PCI Trace Ctrl Signal Mask Register

Table 1060. 0x94 - TCM - PCI trace Ctrl signal mask register

31 20	19 16	15	14	13	12	11	10	9	8	7	6	5	4	3 0
RESERVED	CBE[3:0]	FRAME	IRDY	TRDY	STOP	DEVSEL	PAR	PERR	SERR	IDSEL	REQ	GNT	LOCK	RST	RES
0	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	0
r		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r

31: 20 RESERVED

19: 3 Mask for the Ctrl signal pattern. When mask bit[n] = 0 pattern bit[n] will always be a match.

2: 0 RESERVED

67.9.17 PCI Trace PCI AD State Register

Table 1061. 0x98 - TADS - PCI trace PCI AD state register

31 0
Sampled PCI AD signal
NR
r

31: 0 The state of the PCI AD signal.

67.9.18 PCI Trace PCI Ctrl Signal State Register

Table 1062. 0x9C - TCS - PCI trace PCI Ctrl signal state register

31 20	19 16	15	14	13	12	11	10	9	8	7	6	5	4	3 0
RESERVED	CBE[3:0]	FRAME	IRDY	TRDY	STOP	DEVSEL	PAR	PERR	SERR	IDSEL	REQ	GNT	LOCK	RST	RES
0	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR	NR
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

31: 20 RESERVED

19: 3 The state of the PCI Ctrl signals.

2: 0 RESERVED

67.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x07C. The DMA engine has device identifier 0x07D. The separate APB interface for the PCI Trace-Buffer has device identifier 0x07E. For description of vendor and device identifier see GRLIB IP Library User's Manual

67.11 Implementation

67.11.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). By default, the core makes use of synchronous reset and resets a subset of its internal registers in the system clock domain.

The deassertion of the PCI reset is synchronized to the PCI clock and delayed 3 clock cycles.

The core can be configured to drive the AHB reset on the PCI reset signal. This option is used when the backplane does not have logic to drive the PCI reset.

The PCI reset signal can optionally be forwarded to the AHB reset via the ptarst signal. This functionality can be used then the AMBA clock domain needs to be reset when the PCI reset is asserted.

The core require that the PCI and AMBA clock domain is reset at the same time, i.e. the PCI reset and the AMBA reset needs to be asserted at the same time.

67.11.2 Technology mapping

The core has a technology mapping VHDL generic, *memtech*, which controls how the memory cell used will be implemented. See the GRLIB Users's Manual for available settings.

67.11.3 RAM usage

The FIFOs in the core is implemented with the *syncram_2pft* (with separate clocks for each port) component from the technology mapping library (TECHMAP). Each data path implements its FIFOs in a separate 32-bit wide syncram_2pft component. The depth of each of these RAMs is the FIFO depth * number of FIFOs.

67.11.4 Pull-ups

Please refer to the PCI Local Bus Specification on which of the PCI signals needs to have pull-ups for correct operations.

67.11.5 PHY

All logic and registers directly controlled by the PCI bus signals has be placed in a separate entity. This makes it easier to control the setup-, hold- and clock-to-out timing for the PCI bus signals. This logic can also be implemented as a netlist which can be manually placed before running place-and-route for the entire design. A netlist is provided for Axcelerator and RTAX targets.

67.11.6 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

67.12 Configuration options

Table 1063 shows the configuration options of the core (VHDL generics).

*Table 1063.*Configuration options

Generic name	Function	Allowed range	Default
memtech	The memory technology used for the internal FIFOs.	0 - NTECH	0
tbmemtech	The memory technology used for trace buffers	0 - NTECH	0
oepol	Polarity of the pad output enable signal. 0 = active low, 1 = active high.	0 - 1	0
hmindex	AHB master index.	0 - NAHBMST-1	0
hdmindex	DMA AHB master index.	0 - NAHBMST-1	0
hsindex	AHB slave index.	0 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR (for PCI memory access).	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR.	0 - 16#FFF#	16#000#
ioaddr	ADDR field of the AHB IO BAR (for PCI configuration and PCI IO access).	0 - 16#FFF#	16#000#
pindex	APB slave index	0 - APBMAX-1	0
paddr	APB interface base address	0 - 16#FFF#	0
pmask	APB interface address mask	0 - 16#FFF#	16#FFF#
irq	Interrupt line used by the core.	0 - NAHBIRQ-1	0
irqmode	IRQ routing option: 00: PCI INTA..D, Error interrupt and DMA interrupt on the same IRQ signal 01: PCI INTA..D and Error interrupt on the same IRQ signal. DMA interrupt on IRQ+1 10: PCI INTA..D on IRQ..IRQ+3. Error interrupt and DMA interrupt on IRQ. 11: PCI INTA..D on IRQ..IRQ+3. Error interrupt on IRQ. DMA interrupt on IRQ+4	0 - 3	0
master	Enable the PCI master	0 - 1	1
target	Enable the PCI target	0 - 1	1
dma	Enable the PCI dma	0 - 1	1
tracebuffer	Enable and number of entries of the PCI trace buffer, Allowed values is 0, 32, 64, 128, ..., 16384.	0 - 16384	0
confspace	Enable the PCI Configuration Space when PCI target is disabled	0 - 1	1
vendorid	PCI vendor ID	0 - 16#FFFF#	0
deviceid	PCI device ID	0 - 16#FFFF#	0
classcode	PCI class code	0 - 16#FFFFFF#	0
revisionid	PCI revision ID	0 - 16#FF#	0
cap_pointer	Enabled and sets the offset of the first item in the Extended PCI Configuration Space	0 - 16#C0#	0
ext_cap_pointer	Offset of the first user defined item in the capability list	0 - 16#FC#	0
iobase	AHB base address of the AHB I/O area	0 - 16#FFF#	16#FFF#
extcfg	Default value of the user defined Extended PCI Configuration Space to AHB address mapping.	0 - 16#FFFFFFF#	0
bar0	Sets the default size of BAR0 in address bits.	0 - 31	0
bar1	Sets the default size of BAR1 in address bits.	0 - 31	0
bar2	Sets the default size of BAR2 in address bits.	0 - 31	0
bar3	Sets the default size of BAR3 in address bits.	0 - 31	0

*Table 1063.*Configuration options

Generic name	Function	Allowed range	Default
bar4	Sets the default size of BAR4 in address bits.	0 - 31	0
bar5	Sets the default size of BAR5 in address bits.	0 - 31	0
bar0_map	Set the default PCI BAR to AHB address mapping for BAR0	0 - 16#FFFFFF#	0
bar1_map	Set the default PCI BAR to AHB address mapping for BAR1	0 - 16#FFFFFF#	0
bar2_map	Set the default PCI BAR to AHB address mapping for BAR2	0 - 16#FFFFFF#	0
bar3_map	Set the default PCI BAR to AHB address mapping for BAR3	0 - 16#FFFFFF#	0
bar4_map	Set the default PCI BAR to AHB address mapping for BAR4	0 - 16#FFFFFF#	0
bar5_map	Set the default PCI BAR to AHB address mapping for BAR5	0 - 16#FFFFFF#	0
bartype	Bit[5:0] set the reset value of the prefetch bit for the BAR. bit[n] corresponds to BARn . Bit[13:8] set the reset value of the BAR type bit for the BAR. bit[n + 8] corresponds to BARn.	0 - 16#FFFF#	16#003F#
barminsize	Sets the minimal supported BAR size in address bits. The minimal BAR size is not allowed to be smaller then the internal FIFO (or barminsize >= 2 + fifo_depth).	5 - 31	12
fifo_depth	Depth of each of the FIFOs in the data path. Depth = 2**fifo_depth.The minimal BAR size is not allowed to be smaller then the internal FIFO (or barminsize >= 2 + fifo_depth).	3 - 7	3
fifo_count	Number of FIFOs in the data path	2 - 4	2
conv_endian	Default value of the endianness conversion setting	0 - 1	1
deviceirq	Enable the device to drive the PCI INTA signal	0 - 1	1
deviceirqmask	Default value of the irq mask for the dirq input	0 - 16#F#	16#0#
hostirq	Enable the core to sample the PCI INTA-D signals to drive a AHB irq.	0 - 1	1
hostirqmask	Default value for the PCI INTA-D signals.	0 - 16#F#	16#0#
nsync	Number of synchronization registers between the two clock domains.	0 - 2	2
hostrst	Mode of the reset signal. 0: PCI reset is input only 1: The AHB reset is driven on the PCI reset when PCII.HOST is asserted 2: The AHB reset is driven on the PCI reset.	0 - 2	0
bypass	When 1, logic is implemented to bypass the pad on signals driven by the core.	0 - 1	1
ft	Enable fault-tolerance against SEU errors	0 - 1	0

*Table 1063.*Configuration options

Generic name	Function	Allowed range	Default
scantest	Enable support for scan test	0 - 1	0
debug	Enables debug output signals	0 - 1	0
tbapben	Enables a separate APB interface for access of the Trace-Buffer.	0 - 1	0
tbpindex	Trace-Buffer APB slave index	0 - APBMAX-1	0
tbpaddr	Trace-Buffer APB interface base address	0 - 16#FFF#	0
tbmask	Trace-Buffer APB interface address mask	0 - 16#FFF#	16#FFF#
netlist	Enables a netlist implementation of the logic controlled by the PCI bus signals (GRPCI2_PHY).	0 - 1	0
masters	Controls which AHB masters belongs to PCI function0	0 - 16#FFFF#	16#FFFF#
multifunc	Enables Multi-Function support	0 - 1	0
multiint	Enables support to drive all PCI interrupt signalsINTA...D	0 - 1	0
mf1_deviceid	PCI device ID (PCI function1)	0 - 16#FFFF#	0
mf1_classcode	PCI class code (PCI function1)	0 - 16#FFFFFF#	0
mf1_revisionid	PCI revision ID (PCI function1)	0 - 16#FF#	0
mf1_bar0	Sets the default size of BAR0 in address bits. (PCI function1)	0 - 31	0
mf1_bar1	Sets the default size of BAR1 in address bits. (PCI function1)	0 - 31	0
mf1_bar2	Sets the default size of BAR2 in address bits. (PCI function1)	0 - 31	0
mf1_bar3	Sets the default size of BAR3 in address bits. (PCI function1)	0 - 31	0
mf1_bar4	Sets the default size of BAR4 in address bits. (PCI function1)	0 - 31	0
mf1_bar5	Sets the default size of BAR5 in address bits. (PCI function1)	0 - 31	0
mf1_bartype	Bit[5:0] set the reset value of the prefetch bit for the BAR. bit[n] corresponds to BARn . Bit[13:8] set the reset value of the BAR type bit for the BAR. bit[n + 8] corresponds to BARn.	0 - 16#FFFF#	16#003F#
mf1_bar0_map	Set the default PCI BAR to AHB address mapping for BAR0 (PCI function1)	0 - 16#FFFFFF#	0
mf1_bar1_map	Set the default PCI BAR to AHB address mapping for BAR1 (PCI function1)	0 - 16#FFFFFF#	0
mf1_bar2_map	Set the default PCI BAR to AHB address mapping for BAR2 (PCI function1)	0 - 16#FFFFFF#	0
mf1_bar3_map	Set the default PCI BAR to AHB address mapping for BAR3 (PCI function1)	0 - 16#FFFFFF#	0
mf1_bar4_map	Set the default PCI BAR to AHB address mapping for BAR4 (PCI function1)	0 - 16#FFFFFF#	0
mf1_bar5_map	Set the default PCI BAR to AHB address mapping for BAR5 (PCI function1)	0 - 16#FFFFFF#	0
mf1_cap_pointer	Enabled and sets the offset of the first item in the Extended PCI Configuration Space (PCI function1)	0 - 16#C0#	0
mf1_ext_-cap_pointer	Offset of the first user defined item in the capability list (PCI function1)	0 - 16#FC#	0

*Table 1063.*Configuration options

Generic name	Function	Allowed range	Default
mf1_extcfg	Default value of the user defined Extended PCI Configuration Space to AHB address mapping. (PCI function1)	0 - 16#FFFFFFF#	0
mf1_masters	Controls which AHB masters belongs to PCI function1	0 - 16#FFFF#	16#0000#

67.13 Signal descriptions

Table 1064 shows the interface signals of the core (VHDL ports).

*Table 1064.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
PCICLK	N/A	Input	PCI Clock	-
AHBSI	*1	Input	AHB slave input signals	-
AHBSO	*1	Output	AHB slave output signals	-
AHBMI	*1	Input	AHB master input signals	-
AHBMO	*1	Output	AHB master output signals	-
AHBDMI	*1	Input	DMA AHB master input signals	-
AHBDMO	*1	Output	DMA AHB master output signals	-
APBI	*1	Input	APB slave input signals	-
APBO	*1	Output	APB slave output signals	-
PCII	*2	Input	PCI input signals	-
PCIO	*2	Output	PCI output signals	-
DIRQ		Input	Interrupt signals	High
TBAPBI	*1,*3	Input	Trace-Buffer APB slave input signals	-
TBAPBO	*1, *3	Output	Trace-Buffer APB slave output signals	-
PTARST	N/A, *3	Output	PCI reset to AMBA reset output signal	Low
DEBUG	N/A, *3	Output	Debug signals	-

*1) see GRLIB IP Library User's Manual.
*2) see PCI Local Bus Specification
*3) Can be left unconnected, if not used.

The PCII.HOST signal selects of the core should operate as a system host or peripheral device.

67.14 Library dependencies

Table 1065 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1065.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	PCI	Component	Component declaration

67.15 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
```

```
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.stdlib.all;
use grlib.tech.all;
library gaisler;
use gaisler.pci.all;

.
.
signal apbi : apb_slv_in_type;
signal apbo : apb_slv_out_type;
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector;
signal ahbmi : ahb_mst_in_type;
signal ahbmo : ahb_mst_out_vector;

signal pcii : pci_in_type;
signal pcio : pci_out_type;

begin

pci0 : grpci2
generic map (
 memtech => memtech,
 oepol => OEPOL,
 hmindex => CFG_NCPU+CFG_AHB_UART+CFG_AHB_JTAG,
 hdmindex => CFG_NCPU+CFG_AHB_UART+CFG_AHB_JTAG+1,
 hsindex => 4,
 haddr => 16#c00#,
 hmask => 16#f00#,
 ioaddr => 16#000#,
 pindex => 4,
 paddr => 4,
 irq => 0,
 irqmode => 0,
 master => CFG_GRPCI2_MASTER,
 target => CFG_GRPCI2_TARGET,
 dma => CFG_GRPCI2_DMA,
 tracebuffer => CFG_GRPCI2_TRACE,
 vendorid => CFG_GRPCI2_VID,
 deviceid => CFG_GRPCI2_DID,
 classcode => CFG_GRPCI2_CLASS,
 revisionid => CFG_GRPCI2_RID,
 cap_pointer => CFG_GRPCI2_CAP,
 ext_cap_pointer => CFG_GRPCI2_NCAP,
 iobase => CFG_AHBIO,
 extcfg => CFG_GRPCI2_EXTCFG,
 bar0 => CFG_GRPCI2_BAR0,
 bar1 => CFG_GRPCI2_BAR1,
 bar2 => CFG_GRPCI2_BAR2,
 bar3 => CFG_GRPCI2_BAR3,
 bar4 => CFG_GRPCI2_BAR4,
 bar5 => CFG_GRPCI2_BAR5,
 fifo_depth => log2(CFG_GRPCI2_FDEPTH),
 fifo_count => CFG_GRPCI2_FCOUNT,
 conv_endian => CFG_GRPCI2_ENDIAN,
 deviceirq => CFG_GRPCI2_DEVINT,
 deviceirqmask => CFG_GRPCI2_DEVINTMSK,
 hostirq => CFG_GRPCI2_HOSTINT,
 hostirqmask => CFG_GRPCI2_HOSTINTMSK,
 nsync => 2,
 hostrst => 2,
 bypass => CFG_GRPCI2_BYPASS)
port map (
 rstn,
 clkm,
 pciclk,
 gnd(3 downto 0),
 pcii,
 pcio,
```

```
 apbi,
 apbo(4),
 ahbsi,
 ahbso(4),
 ahbmi,
 ahbmo(CFG_NCPU+CFG_AHB_UART+CFG_AHB_JTAG),
 ahbdmi,
 ahbmo(CFG_NCPU+CFG_AHB_UART+CFG_AHB_JTAG+1);

pcipads0 : pcipads generic map (padtech => padtech, host => 1, oepol => OEPOL,
                                noreset => 0, drivereset => 1)  -- PCI pads
    port map ( pci_rst, pci_gnt, pci_idsel, pci_lock, pci_ad, pci_cbe,
      pci_frame, pci_irdy, pci_trdy, pci_devsel, pci_stop, pci_perr,
      pci_par, pci_req, pci_serr, pci_host, pci_66, pcii, pcio );
;
```

68 GRPPSTIMER - PPS TIMER

68.1 Overview

The PPS Timer (GRPPSTIMER) provides basic time keeping functions such an Elapsed Time (ET) counter according to the Consultative Committee for Space Data Systems (CCSDS) Unsegmented Code specification, [CCSDS]. It comprises a Frequency Synthesizer (FS) by which a binary frequency is generated to drive the ET counter. The GRPPSTIMER also provides support for setting the increment rate of the ET counter as well as of the FS counter.

The timer can be synchronised by providing a Pulse Per Second (PPS) input signal. With this signal, the core can adjust its internal frequency synthesizer, leading to corrections in the ET.

The GRPPSTIMER provides datation services that sample the ET counter value on external events. It also provides generation of periodic pulses with cycle periods both greater and less than 1 second. All services in the GRPPSTIMER core are accessible via an AMBA APB slave interface.

This core can operate in two modes: autonomous and synchronous. In autonomous mode, the PPS input signal is not utilized, so no time corrections are applied. Functionalities like datation and pulse generation can still be used. In synchronized mode, a PPS input signal is used, and time corrections are made based on this signal. The GRPPSTIMER can act as a master and/or a slave in a time distribution system.

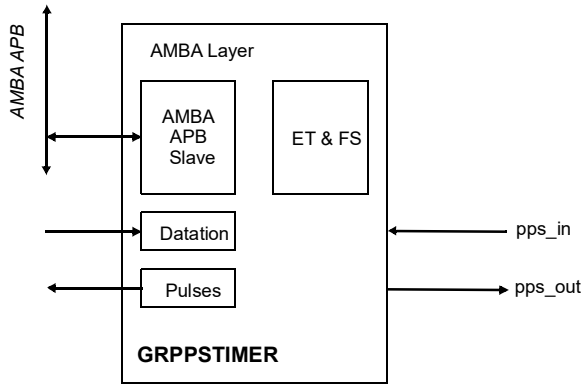


Figure 164. Block diagram

68.1.1 Foreseen usage of the core

On-board time maintenance and distribution is to be handled through a PPS Timer (GRPPSTIMER) and can be extended by one or more slave GRPPSTIMERs. There are two ways to connect a slave to a master:

- Using the APB interface, the slave GRPPSTIMER can be synchronised with another GRPPSTIMER by with the help of a CPU. This CPU can read the Elapsed Time (ET) from one GRPPSTIMER and share that time with the CPU of another GRPPSTIMER, which will function as a slave. The ET can be read and send by accessing the APB interface of the GRPPSTIMERS.

- The PPS output signal of the master GRPPSTIMER can be connected to the PPS input of the slave GRPPSTIMER. By linking the PPS signals, the slave GRPPSTIMER can align its timing precisely with the master.

The slave GRPPSTIMER can further distribute the time to the payload. The GRPPSTIMER slaves will thus be slaved to the master GRPPSTIMER, but also act as masters for other modules. This isolates the master GRPPSTIMER from the payloads.

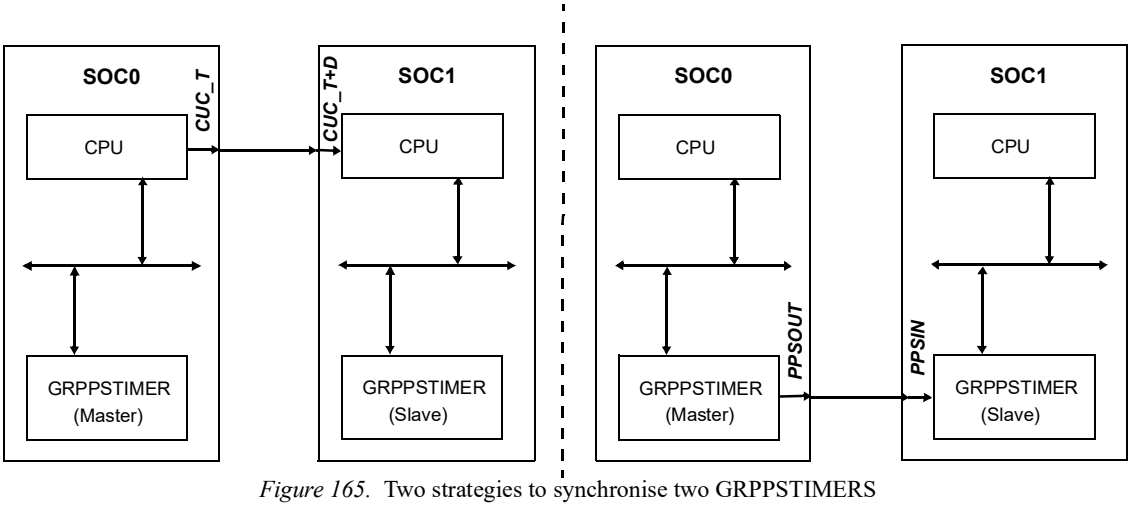


Figure 165. Two strategies to synchronise two GRPPSTIMERS

68.1.2 Description of a general system using the core

The standard method for maintaining precise on-board time involves using a central time reference that tracks elapsed time from a chosen epoch. This time information is then regularly shared with on-board applications through messages and synchronization pulses. Each GRPPSTIMER instance keeps its own version of this central elapsed time, allowing on-board applications to accurately timestamp their data. This continuous link between time references on-board, as well as between the spacecraft and ground, ensures that the timing of any event on-board can be accurately determined in any specific space-time frame.

68.2 Data formats

All Elapsed Time (ET) information handled by GRPPSTIMER is compliant with the CCSDS Unsegmented Code defined in [CCSDS] and repeated hereafter.

68.2.1 Reference documents

[CCSDS] Time Code Formats, CCSDS 301.0-B-4, www.ccsds.org

68.2.2 CCSDS Unsegmented Code: Preamble Field (P-Field)

The time code preamble field (P-field) is constant and has value 0x2F. As shown in the table below, this core supports the Agency-defined epoch (Level 2) and has 4 octets for the coarse time and 3 octets for the fine time.

*Table 1066.*CCSDS Unsegmented Code P-Field definition

Bit	Value		Interpretation
0	0		Extension flag
1 - 3	“001”	1958 January 1 epoch (Level 1) [1]	Time code identification
	“010”	Agency-defined epoch (Level 2) [1]	
4 - 5	(number of octets of coarse time) - 1		Detail bits for information on the code
6 - 7	(number of octets of fine time)		

1 *For the Standard Spacecraft Time Source Packet defined in the ESA Packet Telemetry Standard, bits 1 to 3 must are set to 010_b.*

68.2.3 CCSDS Unsegmented Code: Time Field (T-Field)

For the unsegmented binary time codes described herein, the T-Field consists of a selected number of contiguous time elements, each element being one octet in length. An element represents the state of 8 consecutive bits of a binary counter, cascaded with the adjacent counters, which rolls over at a modulo of 256.

*Table 1067.*CCSDS Unsegmented Code T-Field definition

CCSDS Unsegmented Code												
Preamble Field	Time Field											
	Coarse time								Fine time			
-	2^{31}	2^{24}	2^{23}	2^{16}	2^{15}	2^{8}	2^{7}	2^{0}	2^{-1}	2^{-8}	2^{-9}	2^{-15}	2^{-16}	2^{-24}

The basic time unit is the second. The T-Field consists of 32 bits of coarse time (seconds) and 24 bits of fine time (sub seconds). The coarse time code elements are a count of the number of seconds elapsed from the epoch. The 32 bits of coarse time results in a maximum ambiguity period of approximately 136 years. Arbitrary epochs may be accommodated as a Level 2 code. The 24 bits of fine code elements result in a resolution of 2^{-24} second (about 60 nanoseconds). This code is not UTC-based and leap second corrections do not apply according to CCSDS.

68.2.4 Pulse generator waveform

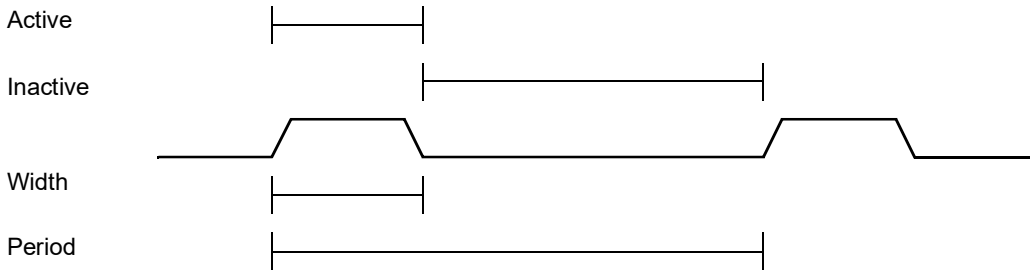


Figure 166. Pulse generation waveform

68.3 Operation

The CCSDS Time Manager (GRPPSTIMER) synthesizable core can be configured for various purposes.

68.3.1 Elapsed Time (ET)

The local Elapsed Time (ET) counter is based on a default 32 bit coarse time field and a 24 bit fine time field, complying to the CCSDS Unsegmented Code (CUC) T-Field. The width of the two time fields is fixed. The counter implementing the ET is incremented on the system clock only when enabled by the frequency synthesizer described below. The ET is incremented with a pre-calculated increment value, which matches the synthesised frequency. The local ET is output in the CUC format, P-Field and T-Field, to be used by an application embedding the GRPPSTIMER. The P-Field is static with the Time Code Identifier is set to 010b.

68.3.2 Frequency Synthesizer (FS)

The binary frequency required to determine the ET counter increment is derived from the system clock using a frequency synthesizer. The frequency synthesizer is incremented with a pre-calculated increment value, which matches the available system clock frequency. The FS simply generates a tick

every time it wraps around, which makes the ET to step forward with the pre-calculated increment value. The output of the frequency synthesizer is used for enabling the increment of the local ET as described above.

68.3.3 Datation

The GRPPSTIMER comprises eight datation registers for the purpose of datation of user events relative the ET counter. The datation is triggered by eight external edge sensitive inputs (programmable rising or falling edge).

Every datation services is automatically disabled after an occurrence and is not re-enabled until the corresponding fine time register is read. Therefore, it is important to read the coarse time first, because reading the fine time will unlock the datation register. The format of all eight datation registers is compliant to the CUC T-Field.

The ET counter can be accessed directly via the AMBA APB interface. This can be used for direct datation from software.

68.3.4 PPS Input Selection

The GRPPSTIMER provides 4 PPS input signals. The desired PPS input signal can be selected by writing to Select PPS (SP) field in the Global Control Register (GCR).

68.3.5 PPS Output

The PPS output can be enabled or disabled in the Global Control Register (GCR). The output signal is always active high. If the signal is disabled while active high, it will turn off immediately without completing the current cycle. The output signal width is set in the Fine Correction Timeout and PPS Output Width Register (FCTMRPOW), in the PPS Output Width (POW) field. The width is defined as 2^{POW} clock cycles.

68.3.6 Pulses

The GRPPSTIMER provides sixteen external outputs used for clock pulse distribution. The timing of each pulse output is individually derived from the Elapsed Time counter. The corresponding base 2 or 10 can be set by the Pulse Definition Register (PDRX).

Base 2:

For base 2, it is possible to program for each pulse output individually the following parameters:

- periodicity pulse
- width of pulse
- polarity of pulse
- enable/disable pulse generation

The pulse has two parts, the active and the inactive part. The active part always starts the pulse, followed by the inactive part. The polarity or logical level of the active part is programmable. The inactive part takes the logical inversion of the active pulse, and is the default output from the generator when the pulse is not issued or the overall generation is disabled. The leading edge of the active pulse part is aligned with the 1 second transition of the Elapsed Time counter.

The periodicity of the pulse corresponds to one of the ET bits that can be selected in the range 27 to 2-8 seconds, providing a range from 128 seconds to 3,91 ms, i.e. 0,0078 to 256 Hz frequency. See register definition for details.

The width of the active part of the pulse corresponds to one of the ET bits that can be selected in the range 2^{6} to 2^{-9} seconds, providing a range from 64 seconds to 1,95 ms. See register definition for details.

It is possible to generate a pulse that has a duty cycle of 50%. It is also possible to generate a pulse for which the active part is as short as 2^{-9} seconds, and its period is as high as 2^{7} seconds. The effective duty cycle can be as low as $2^{-9}/ 2^{7}$ for the longest period, up to 50% for the shortest period of 2^{-8} seconds = 256 Hz. The duty cycle choice becomes more restricted as the frequency increases. Note that it is only possible to reduce the duty cycle in one direction: 50%/50%, 25%/75% ... 1%/99%. The active part of the pulse can thus never be more than 50% of the cycle. It should be noted that the active pulse width must be at most 50% of the pulse period. This is a requirement on the software usage.

Base 10:

Base 10 can be approximated by the following parameters:

- pulse count
- polarity of pulse
- enable/disable pulse generation (reset status is disabled)

For base 10, only signals with a 1 second period can be generated. The smallest width can be achieved by setting the pulse count 1, which results in a pulse width of 2^{-18}. The maximum pulse with is $1\text{-}2^{-18}$. For instance, a signal width of 100 ms can be approximated by setting the pulse count register to 26214 ($\text{round}(0.1 / 2^{-18}) = 26214$).

The pulse outputs are guaranteed to be spike free. If the re-synchronisation of the GRPPSTIMER in slave mode occurs within 0,5 ms of the expected synchronisation instance, the ongoing pulse output width will be accurate to within 0,5 ms. Else, the pulse output will remain unchanged corresponding to up to four times the expected output width.

If a pulse output is disabled by means of writing to the corresponding register (PDRx) (i.e. writing a zero to the Pulse Enable bit (PE)), the pulse output will be immediately driven to the inversion of the Pulse Level bit (PL), which corresponds to the level of the inactive part of the pulse. It is thus possible to modify immediately the pulse output by disabling it using the PE bit and then changing the PL bit, since the output will always drive the inversion of the PL bit while disabled. The pulse outputs are always synchronized with the PPS output signal.

68.3.7 Glitch filter

The GRPPSTIMER includes a glitch filter which passes the PPS signal when the PPS input signal stays high for the configured amount of filter width. The filter width can be configured with good margin, for instance 96 clock cycles for a 128 clock cycle PPS Pulse. The filter width can be configured in the Input PPS Control Register (IPCR). The glitch filter can raise an interrupt when it detects an input glitch.

68.3.8 PPS Time Synchronisation

The core supports Time Synchronisation by providing a PPS reference signal. With Drift and Fine Corrections the current Elapsed Time remains accurate. Drift and Fine Correction can be enabled by individual bits within the Global Control Register (GCR). Drift Correction calculates the average duration of the PPS signal, with the required number of PPS cycles defined in the Frequency Accumulator Control Register (FACR). It measures the absolute difference between the desired frequency and the PPS reference signal. When the average PPS input duration exceeds the desired frequency, indicating a positive drift in the board clock frequency, this difference is multiplied by the Compensation Value Register (CVR) and subtracted from the Frequency Synthesizer Increment (FSINC). If the average duration is lower than the desired frequency, the difference is added to the FSINC.

When Fine Correction is enabled, the fine correction will take the difference in clock cycles between the PPS reference signal, after passing through the glitch filter, and the internal reference PPS signal (PPS output). This difference is multiplied by the same compensation constant and is then either added to or subtracted from the frequency accumulator.

An interrupt is raised when the delta in the fine correction reaches the timeout threshold set in the Fine Correction Timeout Register (FCTR). This interrupt is always triggered, regardless of the fine correction enable bit status.

68.3.9 AMBA APB slave interface

The APB interface of the GRPPSTIMER, supports 32 bit read and write accesses.

68.3.10 Interrupts

The GRPPSTIMER provides an interrupt when the programmed PPS Timeout (PTR) has been reached, the core will send the MISSPPS interrupt.

The RCVDPPS interrupt is raised when a PPS input signal has been received and has been passed by the glitch filter when glitch filter is enabled. When the glitch filter is disabled the interrupt is immediately raised on arrival of a PPS input pulse.

The FINECORRHIGH provides an interrupt when the fine correction reaches a certain threshold. GLITCHDETECT is triggered when the glitch filter detects a glitch. In addition, the GRPPSTIMER provides individual interrupt lines for the incoming datation inputs and the occurrence of the individual pulse outputs. The interrupt lines are asserted for at least two system clock cycles and can be connected to an external interrupt controller. The interrupts indicate that a new datation value can be read.

The interrupts defined in table 1068 are generated. When the number of datations or pulses are reduced, the interrupt number is reduced as well.

*Table 1068.*Interrupts

Interrupt offset	Interrupt name	Description
1:st	MISSPPS	Missing PPS
2:nd	RCVDPPS	PPS input signal received
3:rd	FINECORRHIGH	Fine correction too high
4:rd	GLITCHDETECT	Glitch detected in glitch filter
5:th	DRL0	Datation Register 0 Latched
6:th	DRL1	Datation Register 1 Latched
7:th	DRL2	Datation Register 2 Latched
8:th	DRL3	Datation Register 3 Latched
9:th	DRL4	Datation Register 4 Latched
10:th	DRL5	Datation Register 5 Latched
11:th	DRL6	Datation Register 6 Latched
12:th	PULSE0	Pulse 0 interrupt
13:th	PULSE1	Pulse 1 interrupt
14:th	PULSE2	Pulse 2 interrupt
15:th	PULSE3	Pulse 3 interrupt
16:th	PULSE4	Pulse 4 interrupt
17:th	PULSE5	Pulse 5 interrupt
18:th	PULSE6	Pulse 6 interrupt
19:th	PULSE7	Pulse 7 interrupt
20:th	PULSE8	Pulse 8 interrupt
21:th	PULSE9	Pulse 9 interrupt
22:th	PULSE10	Pulse 10 interrupt
23:th	PULSE11	Pulse 11 interrupt
24:th	PULSE12	Pulse 12 interrupt
25:th	PULSE13	Pulse 13 interrupt
26:th	PULSE14	Pulse 14 interrupt
27:th	PULSE15	Pulse 15 interrupt

68.4 Registers

The core is programmed through registers mapped into APB address space. Only 32-bit single-accesses to the registers are supported.

*Table 1069.*GRPPSTIMER registers

APB address offset	Register
0x00	Core Frequency Register (CFR)
0x04	Frequency Synthesizer Increment Register (FSIR)
0x08	Frequency Accumulator Control Register (FACR)
0x0C	Global Control Register (GCR)
0x10	Compensation Value Register (CVR)
0x14	Elapsed Time Coarse Register (ETCR)
0x18	Elapsed Time Fine Register (ETFR)
0x1C	Total PPS Count Register (TPCR)
0x20	Accumulator Average Register (AAR)
0x24	PPS Phase Difference Register (PPDR)
0x28	Input PPS Control Register (IPCR)
0x2C	Datation Control Register (DCR)
0x30	Datation Status Register (DSR)
0x34	Preamble Field Register (PFR)
0x38	Set Elapsed Time Coarse Register (SETCR)
0x3c	Set Elapsed Time Fine Register (SETFR)
0x40-0x7F	Pulse Definition Registers 0-15 (PDR0-15) 0x40: Pulse Definition Register 0 0x44: Pulse Definition Register 1 0x48: ....
0x80-0xBF	Datation Time Registers 0-7 (DCR/DFR-7) 0x80: Datation Coarse Register 0 0x84: Datation Fine Register 0 0x88: Datation Coarse Register 1 0x8C: ....
0xC0	Pending Interrupt Masked Status Register (PIMSR)
0xC4	Pending Interrupt Masked Register (PIMR)
0xC8	Pending Interrupt Status Register (PIRSR)
0xCC	Pending Interrupt Register (PIR)
0xD0	Interrupt Mask Register (IMR)
0xD4	Pending Interrupt Clear Register (PICR)
0xE0	Fine Correction Timeout and PPS Output Width Register (FCT-POWR)

68.4.1 Core Frequency Register

Table 1070. 0x00 - CFR - Core Frequency Register

31	0
FREQ	
0	
rw	

31: 0 Core Frequency (FREQ)

68.4.2 Frequency Synthesizer Increment Register

Table 1071. 0x04 - FSIR - Frequency Synthesizer Increment Register

31	0
FSINC	
0	
rw	

31: 0 Frequency Synthesizer Increment, bits: (31 downto 0) [Note 2]

68.4.3 Frequency Accumulator Control Register

Table 1072. 0x08 - FACR - Frequency Accumulator Control Register

31 24	23 20	17	16	15 8	7 0
RESERVED	FSINC	LEV	BGF	NOPPSAVGC	ETINC
0	0	1	0	0	0
r	rw	rw	rw	rw	rw

31: 24 RESERVED

23: 20 Frequency Synthesizer Increment (FSINC) - bits (35 downto 32)

17 PPS signal level (LEV) - 0 = active low, 1 = active high

16 Bypass Glitch Filter (BGF) - bypassing the glitch filter for internal PPS Reference signals

15: 8 Number Of PPS For Average Count (NOPPSAVGC) - The number of PPS input signals used for calculating the average pulse width.

7:0 Elapsed Time Increment (ETINC) - Timer counter increment on frequency synthesizer wrap around [Note 2]

68.4.4 Global Control Register

Table 1073. 0x0C - GCR - Global Control Register

31 16	15 9	8	7	6	5 4	3	2	1	0
OFFSET	RESERVED	EPO	RTFP	SYNC	SP	EFC	EDC	RST	EN
0	0	1	1	0	0	0	0	0	0
rw	r	rw	rw	rw	rw	rw	rw	rw	rw

31: 16 Offset (OFFSET) - The time required to determine whether the PPS signal is valid. This field is added to the fine time and represents the lower 16 bits.

15: 9 RESERVED

8 Enable PPS Output (EPO)

7 Reset Time on First PPS (RTFP) - Time Reset Strategy upon receiving the first PPS signal:
0: Do not reset the time
1: Reset time to 0

Table 1073. 0x0C - GCR - Global Control Register

6	Enable Sync Time PPS (SYNC) - The Elapsed Time (ET) will be updated by the time set in the Set Elapsed Time Register (SETR): 0 = On the next clock cycle. 1 = On the next falling or rising edge of the PPS input signal.
5:4	Select PPS Input (SP)
3	Enable Fine Correction (EFC)
2	Enable Drift Correction (EDC)
1	Reset core (RST)
0	Enable core (EN)

68.4.5 Compensation Value Register

Table 1074. 0x10 - CVR - Compensation Value Register

31	0
CVR	
0	
rw	

31: 0 Compensation Value Register (CVR) [Note 2]

68.4.6 Elapsed Time Coarse Register

Table 1075. 0x14 - ETCR - Elapsed Time Coarse Register

31	0
T-FIELD, COARSE	
0	
r	

31: 0 T-FIELD, Coarse part [Note 0]

68.4.7 Elapsed Time Fine Register

Table 1076. 0x18 - ETFR - Elapsed Time Fine Register

31 24	23 0
APC	T-FIELD, FINE
0	0
r	r

31: 24 Average PPS count (APC) - Current PPS Count for the average accumulator

23: 0 T-FIELD, Fine part [Note 0]

68.4.8 Total PPS Count Register

Table 1077. 0x1C - TPCR - Total PPS Count Register

31	0
TPC	
0	
r	

31: 0 Total PPS Count (TPC) - The total number of PPS input signals received.

68.4.9 Accumulator Average Register

Table 1078. 0x20 - AAR - Accumulator Average Register

31	0
AA	
0	
rw	

31: 0 Accumulator Average (AA) - Represents the average period of the most recent PPS signals in clock cycles.

68.4.10 PPS Phase Difference Register

Table 1079. 0x24 - PPDR - PPS Phase Difference Register

31	0
PPD	
0	
r	

31: 0 PPS Phase Difference Register (PPD) - The phase difference in clock cycles, by which the PPS out is delayed by the PPS in.

68.4.11 Input PPS Control Register

Table 1080. 0x28 - IPCR - Input PPS Control Register

31 24	23 0
GFW	PT
0x4	0xC350
rw	rw

31: 24 Glitch Filter Width (GFW) - Number of clock cycles

23: 0 PPS Timeout (PT) - Number of clock cycles

68.4.12 Datation Control Register

Table 1081. 0x2C - DCR - Datation Control Register

31 16	15 8	7 0
RESERVED	DEN	DED
0	0	0xF
r	rw	rw

31: 8 RESERVED

15: :8 Datation Enable (DEN)

7:0 Datation Edge (DED) - Configures the edge for the corresponding datation input. 0 = active low, 1 = active high.

68.4.13 Datation Status Register

Table 1082. 0x30 - DSR - Datation Status Register

31	0
RESERVED	DRL
0	0
r	r

31: 8 RESERVED

7:0 Datation Register Latched (DRL) - When set to 1, it indicates whether the corresponding Datation register has been latched. This bit is sticky which means that it remains asserted until the corresponding Datation register is read at which point the bit is cleared. The corresponding registers should be read in the DCRx – DFRx order.

68.4.14 Preamble Field Register

Table 1083. 0x34 - PFR - Preamble Field Register

31 8	7 0
RESERVED	P-FIELD
0	0x2F
r	r

Table 1083. 0x34 - PFR - Preamble Field Register

31: 8 RESERVED

7: 0 Preamble Field (P-Field) - See Table 1066 for the interpretation of this value.

68.4.15 Set Elapsed Time Coarse Register

*Table 1084.*0x38 - SETCR - Set Elapsed Time Coarse Register

31	0
T-FIELD, COARSE	
0	
rw	

31: 0 T-Field, Coarse part

68.4.16 Set Elapsed Time Fine Register

*Table 1085.*0x3C - SETFR - Set Elapsed Time Fine Register

31 25	24	23 0
RESERVED	ARM	T-FIELD, FINE
0	0	0
r	R	rw

31: 25 RESERVED

24: Armed (ARM) - Indicates that the time has been set, this flag will be cleared when the time is updated.

23: 0 T-FIELD, Fine part

68.4.17 Pulse Definition Register 0 to 7

*Table 1086.*0x40 - 0x7F- PDR0 to PDR7 - Pulse Definition Register 0 to 7

31 30 29	12	11 8	7 4	3	2	1	0
RES	PCNT	PP	PW	RES	BS	PL	PE
0	0	0	0	0	0	1	0
r	rw	rw	rw	r	rw	rw	rw

31: 30 RESERVED

29: 12 Pulse Count (PCNT) - Each count represents a time interval of 2^{-18} seconds. Only used in base 10 mode.

11: 8 Pulse Period (PP) - Only used in base 2 mode:

'0000'	=	2^{7} seconds
'0001'	=	2^{6} seconds
'0010'	=	2^{5} seconds
...		
'1110'	=	2^{-7} seconds
'1111'	=	2^{-8} seconds
Period	=	$2^{(7-\mathbf{PP})}$
Frequency	=	$2^{-(7-\mathbf{PP})}$

7: 4 Pulse Width (PW) - Only used in base 2 mode:

Write/Read:	'0000'	=	2^{6} seconds
	'0001'	=	2^{5} seconds
	'0010'	=	2^{4} seconds
	...		
	'1110'	=	2^{-8} seconds
	'1111'	=	2^{-9} seconds
	Width	=	$2^{(6-\mathbf{PW})}$

3: RESERVED

2 Base Select (BS) - '0' = base 2, '1'= base 10

1: Pulse Level (PL) - Defines logical level of active part of pulse output. 0 = Active Low, 1 = Active High

0: Pulse Enable (PE) - '0' = disabled, '1' = enabled

Write: Don't care.

Read: All zero.

68.4.18 Datation Coarse Register 0-7

*Table 1087.*0x80 to 0xBC - DTCR0-7 - Datation Coarse Register 0-7

31 0
T-FIELD, COARSE
0
r

31: 0 T-Field, coarse part [Note 1]

68.4.19 Datation Fine Register 0-7

*Table 1088.*0x84 to 0xBF - DFR0-7 - Datation Fine Register 0-7

31 24	23 0
RESERVED	T-FIELD, FINE
0	0
r	r

31: 24 RESERVED

23: 0 T-Field, fine part [Note 1]

68.4.20 Fine Correction Timeout and PPS Output Width Register

*Table 1089.*0xE0 - FCTPOWR - Fine Correction Timeout and PPS Output Width Register

31 28	27 24	23 0
RESERVED	POW	FCT
0	0x8	0xC350
r	rw	rw

31: 24 RESERVED

27: 24 PPS Out Width (POW) - Width of the PPS output signal in 2^n clock cycles. The value 2^n must be equal or larger than Elapsed Time Increment (ETINC).

23: 0 Fine Correction Timeout (FCT)

[Note 0]When ETCR is read, the ETFR register is latched and are not released until ETFR has been read. The registers should be read in the ETCR – ETFR order.

[Note 1]The coarse and fine time part of the register pair is latched on an external event and is released on reading the corresponding fine time register. No new event is accepted until the corresponding fine time register has been read.

[Note 2]See the provided Excel sheet to calculate the correct settings

68.4.21 Interrupt registers

The interrupt registers give complete freedom to the software, by providing means to mask interrupts, clear interrupts, force interrupts and read interrupt status.

When an interrupt occurs the corresponding bit in the Pending Interrupt Register is set. The normal sequence to initialize and handle a module interrupt is:

- Set up the software interrupt-handler to accept an interrupt from the module.
- Read the Pending Interrupt Register to clear any spurious interrupts.
- Initialize the Interrupt Mask Register, unmasking each bit that should generate the module interrupt.
- When an interrupt occurs, read the Pending Interrupt Status Register in the software interrupt-handler to determine the causes of the interrupt.
- Handle the interrupt, taking into account all causes of the interrupt.
- Clear the handled interrupt using Pending Interrupt Clear Register.

Masking interrupts: After reset, all interrupt bits are masked, since the Interrupt Mask Register is zero. To enable generation of a module interrupt for an interrupt bit, set the corresponding bit in the Interrupt Mask Register.

Clearing interrupts: All bits of the Pending Interrupt Register are cleared when it is read or when the Pending Interrupt Masked Register is read. Reading the Pending Interrupt Masked Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register. Selected bits can be cleared by writing ones to the bits that shall be cleared to the Pending Interrupt Clear Register.

Forcing interrupts: When the Pending Interrupt Register is written, the resulting value is the original contents of the register logically OR-ed with the write data. This means that writing the register can force (set) an interrupt bit, but never clear it.

Reading interrupt status: Reading the Pending Interrupt Status Register yields the same data as a read of the Pending Interrupt Register, but without clearing the contents.

Reading interrupt status of unmasked bits: Reading the Pending Interrupt Masked Status Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register, but without clearing the contents.

The interrupt registers comprise the following:

- Pending Interrupt Masked Status Register [PIMSR] R
- Pending Interrupt Masked Register [PIMR] R
- Pending Interrupt Status Register [PISR] R
- Pending Interrupt Register [PIR] R/W
- Interrupt Mask Register [IMR] R/W
- Pending Interrupt Clear Register [PICR] W

68.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x0DD. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

68.6 Configuration options

Table 1090 shows the configuration options of the core (VHDL generics).

*Table 1090.*Configuration options

Generic	Function	Allowed range	Default
tech	Select technology	0-NTECH	0
sync_stage	Syncreg Stages	Integer	2
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar	0 - 16#FFF#	0
pmask	Mask field of the APB bar	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the core	0 - NAHBIRQ-1	0
datation	Number of datations	0 - 8	8
pulses	Number of pulses	0-16	12
arch	Multiplier structure	0 - 3	0

68.7 Signal descriptions

Table 1091 shows the interface signals of the core (VHDL ports).

*Table 1091.*Signal descriptions

Signal name	Field	Type	Description	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
PPS_IN	N/A	Input	PPS reference input signal	-
FS_INC	N/A	Output	Frequency synthesizer value	-
DATATION_IN	N/A	Input	Datation input signals	-
PPS_OUT	N/A	Output	PPS output signal	-
PULSES_OUT	N/A	Output	Pulse generator output	-

* see GRLIB IP Library User's Manual

68.8 Library dependencies

Table 1092 shows libraries used when instantiating the core (VHDL libraries).

*Table 1092.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions

68.9 Instantiation

This example shows how the core can be instantiated.

```
library  IEEE;
use      IEEE.Std_Logic_1164.all;
library  GRLIB;
use      GRLIB.AMBA.all;

...

component grppstimer is
 generic
 (
  tech : integer := 0;
  sync_stage : integer := 0;
  pindex : integer := 0;
  paddr : integer := 0;
  pmask : integer := 16#fff#;
  pirq : integer := 0;
  datation : integer := DATATIONS;
  pulses : integer := PULSES;
  arch : integer := 0
);
port
(
  rstn : in std_ulogic;
  clk : in std_ulogic;
  apbi : in apb_slv_in_type;
  apbo : out apb_slv_out_type;
  pps_in : in std_logic_vector(3 downto 0);
  datation_in : in std_logic_vector(DATATIONS - 1 downto 0);
  pps_out : out std_ulogic;
```

```
    pulses_out : out std_logic_vector(PULSES - 1 downto 0));
  end component grppstimer;
```

69 GRPULSE - General Purpose Input Output

69.1 Overview

The General Purpose Input Output interface is assumed to operate in an AMBA bus system where the APB bus is present. The AMBA APB bus is used for control and status handling.

The General Purpose Input Output interface provides a configurable number of channels. Each channel is individually programmed as input or output. Additionally, a configurable number of the channels are also programmable as pulse command outputs. The default reset configuration for each channel is as input. The default reset value each channel is logical zero.

The pulse command outputs have a common counter for establishing the pulse command length. The pulse command length defines the logical one (active) part of the pulse. It is possible to select which of the channels shall generate a pulse command. The pulse command outputs are generated simultaneously in phase with each other, and with the same length (or duration). It is not possible to generate pulse commands out of phase with each other.

Each channel can generate a separate internal interrupt. Each interrupt is individually programmed as enabled or disabled, as active high or active low level sensitive, or as rising edge or falling edge sensitive.

69.1.1 Function

The core implements the following functions:

- Input
- Output
- Output pulse commands
- Input interrupts
- Status and monitoring

69.1.2 Interfaces

The core provides the following external and internal interfaces:

- Discrete input and output interface
- AMBA APB slave interface, with sideband signals as per [GRLIB] including:
- interrupt bus
- configuration information
- diagnostic information

69.2 Registers

The core is programmed through registers mapped into APB address space.

Table 1093. GRPULSE registers

APB address offset	Register
0x00	Input Register
0x04	Output Register
0x08	Direction Register
0x0C	Interrupt Mask Register
0x10	Interrupt Polarity Register
0x14	Interrupt Edge Register
0x18	Pulse Register
0x1C	Pulse Counter Register

69.2.1 Input Register

Table 1094. 0x00 - GPIOIN - Input Register

31 24	23 0
RESERVED	IN
0	*
r	r

23-0: IN Input Data

Note that only bits nchannel-1 to 0 are implemented.

69.2.2 Output Register

Table 1095. 0x04 - GPIOOUT - Output Register

31 24	23 0
RESERVED	OUT
0	0
r	r

23: 0 OUT Output Data

All bits are cleared to 0 at reset.

Note that only bits nchannel-1 to 0 are implemented.

69.2.3 Direction Register

*Table 1096.*0x08 - GPIODIR - Direction Register

31 24	23 0
RESERVED	DIR
0	0
r	rw

23: 0 DIR Direction:
0b=input,
1b=output

All bits are cleared to 0 at reset.

Note that only bits nchannel-1 to 0 are implemented.

69.2.4 Pulse Register

*Table 1097.*0x18 - GPIOPULSE - Pulse Register

31 24	23 0
RESERVED	PULSE
0	0
r	rw

23-0: PULSE Pulse enable:
0b=output,
1b=pulse command output

All bits are cleared to 0 at reset.

Only channels configured as outputs are possible to enable as command pulse outputs.

Note that only bits npulse-1 to 0 are implemented.

69.2.5 Pulse Counter Register

*Table 1098.*0x1C - GPIONTR - Pulse Counter Register

31 24	23 0
RESERVED	CNTR
0	0
r	rw

23-0: CNTR Pulse counter value

The pulse counter is decremented each clock period, and does not wrap after reaching zero.

Command pulse channels, with the corresponding output data and pulse enable bits set, are (asserted) while the pulse counter is greater than zero.

Setting CNTR to 0 does not give a pulse.

Setting CNTR to 1 does give a pulse with of 1 Clk period.

Setting CNTR to 255 does give a pulse with of 255 Clk periods.

Note that only bits cntrwidth-1 to 0 need be implemented.

69.2.6 Interrupt Mask Register

*Table 1099.*0x0C - GPIOMASK - Interrupt Mask Register

31 24	23 16	15 0
RESERVED	MASK	RESERVED
0	0	0
r	rw	r

23-16: MASK Interrupt enable, 0b=disable, 1b=enable

Note that only bits that are enabled by the imask VHDL generic and that are in the range nchannel-1 to 0 are implemented.

69.2.7 Interrupt Polarity Register

*Table 1100.*0x10 - GPIOPOL - Interrupt Polarity Register

31 24	23 16	15 0
RESERVED	POL	RESERVED
0	0	0
r	rw	r

23-16: POL Interrupt polarity, 0b=active low or falling edge, 1b=active high or rising edge

Note that only bits that are enabled by the imask VHDL generic and that are in the range nchannel-1 to 0 are implemented.

69.2.8 Interrupt Edge Register

*Table 1101.*0x14 - GPIOEDGE - Interrupt Edge Register

31 24	23 16	15 0
RESERVED	EDGE	RESERVED
0	0	0
r	rw	r

23-16: EDGE Interrupt edge or level, 0b=level, 1b=edge

Note that only bits that are enabled by the imask VHDL generic and that are in the range nchannel-1 to 0 are implemented.

69.3 Operation

69.3.1 Interrupt

Two interrupts are implemented by the interface:

Index:Name:Description:

0 PULSEPulse command completed

31:0 IRQ Filtered input interrupt

The PULSE interrupt is configured by means of the *pirq* VHDL generic.

The IRQ interrupts are configured by means of the *imask* and *ioffset* VHDL generics, where *imask* enables individually the input interrupts, and *ioffset* adds an offset to the resulting index on the interrupt bus.

69.3.2 Reset

After a reset the values of the output signals are as follows:

Signal:Value after reset:

GPIOO.Dout[31:0]de-asserted

GPIOO.OEn[31:0]de-asserted

69.3.3 Asynchronous interfaces

The following input signals are synchronized to Clk:

- GPIOI.Din[31:0]

69.4 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x037. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

69.5 Implementation

69.5.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. See also the description for the *syncrst* VHDL generic for how the core implements reset.

69.6 Configuration options

Table 1102 shows the configuration options of the core (VHDL generics).

*Table 1102.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRPULSE.	0 - NAHBIRQ-1	1
nchannel	Number of input/outputs	1 - 32	24
npulse	Number of pulses	1 - 32	8
imask	Interrupt mask	0 - 16#FFFFFFFF#	16#FF00#
ioffset	Interrupt offset	0-32	8
invertpulse	Invert pulse output when set	1 - 32	0
cntrwidth	Pulse counter width	4 to 32	20
syncrst	Use synchronous reset 0: Core makes use of synchronous reset only 1: Output registers and registers controlling output enable are implemented with asynchronous reset.	0, 1	0
oepol	Output enable polarity	0, 1	1

69.7 Signal descriptions

Table 1103 shows the interface signals of the core (VHDL ports).

*Table 1103.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
GPIOI	*	Input		-
GPIOO	*	Output		-

* see GRLIB IP Library User's Manual

69.8 Signal definitions and reset values

The signals and their reset values are described in table 1104.

*Table 1104.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
gpio[]	Input/Output	General purpose input output	-	Tri-state

69.9 Timing

The timing waveforms and timing parameters are shown in figure 167 and are defined in table 1105.

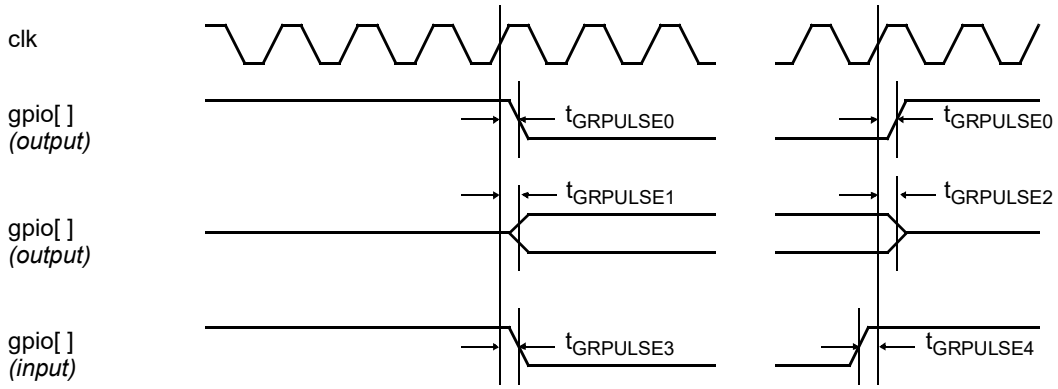


Figure 167. Timing waveforms

Table 1105. Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{GRPULSE0}$	clock to output delay	rising *clk* edge	-	TBD	ns
$t_{GRPULSE1}$	clock to non-tri-state delay	rising *clk* edge	TBD	-	ns
$t_{GRPULSE2}$	clock to tri-state delay	rising *clk* edge	-	TBD	ns
$t_{GRPULSE3}$	input to clock hold	rising *clk* edge	TBD	-	ns
$t_{GRPULSE4}$	input to clock setup	rising *clk* edge	TBD	-	ns

69.10 Library dependencies

Table 1106 shows the libraries used when instantiating the core (VHDL libraries).

Table 1106. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Signals, Components	Signal and component declarations

69.11 Instantiation

This example shows how the core can be instantiated.

```
TBD
```

70 GRPWM - Pulse Width Modulation Generator

70.1 Overview

GRPWM is a pulse width modulation (PWM) generator that supports several outputs, with different frequencies. The core is configured through a set of APB registers, described in section 70.3. The core supports both asymmetric and symmetric PWM generation. Each of the PWM outputs can be configured to be either a single PWM signal or a pair of PWM signals (where the two signals are each others' inverse), with configurable amount of dead band time in between them. The core also supports programming of the output polarity, setting the outputs to fixed values, and configurable interrupt schemes. Hardware support to simplify the generation of a PWM signal that emulate an arbitrary repetitive waveform is also included.

70.2 Operation

70.2.1 System clock scaling

In order to support a wide range of system clock and PWM frequencies the core includes programmable clock scalers. Each scaler is clocked by the system clock and decrement on each clock cycle. When a scaler underflows it is reloaded with the value of its reload register and a tick is generated. This tick can then be used to increment (or decrement) one or more PWM counters. The reload value(s) of the scaler(s) can be read and written through the APB register called *Scaler reload register*, described in section 70.3. The number of system clock scalers is configurable through the VHDL generic *nscalers* and the width of the scaler(s) is determined by the VHDL generic *sbits*.

70.2.2 Asymmetric and symmetric PWM generation

An asymmetric PWM is a pulse signal that is inactive at the beginning of its period and after a certain amount of time goes active, and then stays active for the rest of the period. A symmetric PWM is a pulse signal that is inactive for a certain amount of time at the beginning of the period and a certain amount of time at the end of the period, and stays active in between. The two inactive time periods are normally, but not necessarily, equally long.

For the core to generate a PWM, independent of whether asymmetric or symmetric method is used, software need to do the following (also see section 70.3 for more detailed description of register interface):

- Enable the core by writing the *en* bit in *Core control register*.
- Configure the scaler (see section 70.2.1) and set the PWM period in the *PWM period register*.
- Write the *PWM compare register* with the value at which the PWM's counter should match and switch the outputs.
- If dead band time should be generated, write the value at which the current PWM's dead band time counter should match to the *PWM dead band compare register*. Also set the *dben* bit in the *PWM control register* to 1. See section 70.2.4 for information on dead band time.
- Set the meth bit in PWM control register to either asymmetric och symmetric.
- Set the polarity of the PWM output be setting the *pol* bit in the *PWM control register*.
- If the PWM output should be paired with its inverse then set the *pair* bit in the *PWM control register* to 1, otherwise set it to 0. Note that each PWM always has two ouputs, but if the *pair* bit is set to 0 then the second output is constantly inactive or 0 when the *PWM control register* bit pz is set to 1.
- Program the interrupt, see section 70.2.5.
- Enable the PWM generation by writing the *en* bit in *PWM control register* to 1.

- If software wants the PWM output(s) to assume fix value(s) it can write the *fix* bits in the PWM *control register* appropriately.

Specific configuration required for symmetric PWM if dual compare mode should be used:

- If the core should update the PWM's compare register twice every PWM period, then set the *dcomp* bit in the *PWM control register* to a 1.
- If the *dcomp* bit in the *PWM control register* is set, and it is desired that the two inactive time periods are not of equal length, software needs to continuously update the *PWM compare register* with new values. Since the core updates its internal register at the start of and middle of the PWM period, software need to update the *PWM compare register* sometime during the first half of the period.

Note that the core's internal period register is updated from the *PWM period register* at the start of every period, both for asymmetric and symmetric PWM generation.

70.2.3 Waveform PWM generation

That, which in this document is referred to as a *waveform PWM* is not a PWM generated in a different way than the asymmetric or symmetric methods described above. In fact a waveform PWM is either generated asymmetrically or symmetrically. The difference is that when the compare registers are loaded with new values they are read from an internal RAM instead of the *PWM compare register*. The advantage with this is that if software wants to, for example, generate a PWM signals that emulates a sine wave, it can load a number of compare values into the RAM before starting the PWM generation. Once started, the core will read the RAM, increasing the address at every compare match, and generate the same pattern over and over without the need for software intervention. Note that any pattern that is loaded into the RAM is generated, the core is not limited to a sine wave. This feature is supported if the *wpwm* bit in *Capability register 2* is set to 1. The core only support one waveform PWM and it is always the PWM with the highest index. The index is determined by the VHDL generic *npwm*. If for example *npwm* = 4, then it is only PWM four that can be put in waveform mode. For details on how to configure the waveform mode and read/write the RAM please see the description of the *Waveform configuration register*, *Waveform RAM, word X* registers, and *PWM control register* in section 70.3.

70.2.4 Dead band time

It is often desired to have a delay between when one of the PWM signals of a PWM pair goes inactive and when the other signal goes active. This delay is called dead band time. By default the core does not generate any dead band time, but can be configured to do so by setting the *dben* bit in the *PWM control register* to 0b1. When dead band time is enabled the core will start a counter each time a PWM pair switch its outputs. The output going inactive is not delayed while the output going active is delayed until the counter matches the value in the *PWM dead band compare register*. To support a wide range of applications the amount of dead band time inserted is programmable. The number of bits used in the *PWM dead band compare register* is configurable through the VHDL generic *dbbits*, and also a four bit system clock scaler can be enabled for each PWM's dead band counter by setting the *dbscaler* VHDL generic to 1.

70.2.5 Interrupts

Interrupts can programmed individually for each PWM to be generated at PWM compare match, at PWM period match, or not generated at all. This is programmed in each PWM's *PWM control register.* Each PWM also has a 6-bit interrupt counter that can be used to scale down the frequency at which the interrupts occur. When an interrupt is generated the bit in the *Interrupt pending register* for the PWM in question is set. The bits in the *Interrupt pending register* stay set until software clears them by writing 1 to them. Through the *sepirq* and *npwm* VHDL generics the core supports several different interrupt numbers, this is described in section *70.6*.

When an interrupt is generated, or when the interrupt scaler counter is increased, an output tick is generated on the core's *tick* output signal. The output tick bit has the same index as the PWM in question.

70.3 Registers

The core is programmed through registers mapped into APB address space.

Table 1107. GRPWM registers

APB address offset	Register
0x00	Core control register
0x04	Scaler reload register
0x08	Interrupt pending register
0x0C	Capability register 1
0x10	Capability register 2
0x14*	Waveform configuration registers
0x18 - 0x1C	Reserved, always zero.
0x20**	PWM period register
0x24**	PWM compare register
0x28**	PWM dead band compare register
0x2C**	PWM control register
0x8000***	Waveform RAM, word 0
0x8004***	Waveform RAM, word 1
...	...
0xFFFC***	Waveform RAM, word 8191

* This register is only implemented if the wpwm bit (bit 0) in *Capability register 2* is set to 1.

** This register is implemented once for every PWM (value of *npwm* VHDL generic decides the number of registers), with an offset of 0x10 from the previous PWM's register. The functionality is the same for each PWM.

*** The implementation of this register depends on if the *wpwm* bit (bit 0) in *Capability register 2* is set to 1 and if the waveform RAM is large enough (the value of the field *wabits* in *Capability register 2* reports the number of address bits - 1 that is used for the waveform RAM).

70.3.1 Core Control Register

Table 1108. 0x00 - CTRL - Core control register

31 x+13	x+12 12	11	10 8	7 1	0
R	noup	R	scalersel	R	en
0	0	0	0	0	0
r	rw	r	rw	r	rw

31:x+13 Reserved, always zero. x is the value of bits 2:0 in *Capability register 1*

12+x:12 No update bits for each PWM. x is the value of bits 2:0 in *Capability register 1*. Bit 12 is for the first PWM, bit 13 for the second etc. If a bit is set to 0b1 then that PWM's internal period register, compare register, and dead band compare registers are not updated from the corresponding APB registers. These bits can be used by software if it wants to change more than one of the values and it is required that all values change in the same PWM period. It can also be used to synchronize the use of new values for different PWMs. Reset value 0b0..0.

11 Reserved, always zero.

10:8 System clock scaler select bits. These bits determine which of the implemented system clock scalers' reload value that can be read/written from the *Scaler reload register*.These bits are only present if the *nscalers* generic is greater than 1. Reset value is 0b000

7:1 Reserved, always zero.

0 Core enable bit. 0b0 = Core is disabled, no operations are performed and all outputs are disabled. 0b1 = Core is enabled, PWM outputs can be generated. Reset value is 0b0.

70.3.2 Scaler Reload Register

Table 1109. 0x04 - SCALER - Scaler reload register

31 sbits	sbits-1 0
R	reload
0	all 1
r	rw

31:sbits Reserved, always zero. If sbits = 32 then this field is not present. (sbits is the value of the *sbits* generic)

(sbits-1):0 The value of this field is used to reload the system clock scaler when it underflows. If the core is configured with more than one scaler (*nscalers* generic greater than 1) then the *scalersel* bits in the *Core control register* determine which of the scalers that is read/written. Reset value is 0b1..1 (all ones).

70.3.3 Interrupt Pending Register

Table 1110. 0x08 - IPEND - Interrupt pending register

31 npwm	npwm-1 0
R	irq pending
0	0
r	wc

31:npwm Reserved, always zero.

(npwm-1):0 Interrup pending bits for the PWM(s). When an interrupt event for a specific PWM occurs the core sets the corresponding bit in the interrupt pending register and generates an interrupt. Software can read this register to see which PWM that generated the interrupt.

70.3.4 Capability Register 1

*Table 1111.*0x0C - CAP1 - Capability register 1

31 29	28	27	26 25	24	23	22	21	20 16	15 13	12 8	7 3	2 0
R	def-pol	dcm ode	sepirq	R	sym pwm	asyp wm	dbsc aler	dbbits	nscalers	sbits	pbits	npwm
0	*	*	*	0	*	*	*	*	*	*	*	*
r	r	r	r	r	r	r	r	r	r	r	r	r

31:29 Reserved, always zero.

28 0 = Default polarity is active low (outputs are high after reset/power-up). 1 = Default polarity is active high (outputs are low after reset/power-up).

27 0 = Dual compare mode not implemented. 1 = Dual compare mode implemented.

26:25 Reports interrupt configuration. Value of *sepirq* VHDL generic. Read only.

24 Reserved, always zero.

23 0 = Symmetric PWM generation is not implemented. 1 = Symmetric PWM generation is implemented. Value of *sympwm* VHDL generic. Read only.

22 0 = Asymmetric PWM generation is not implemented. 1 = Asymmetric PWM generation is implemented. Value of *asympwm* VHDL generic. Read only.

21 0 = Dead band time scaler(s) is not implemented. 1 = Dead band time scaler(s) is implemented. Value of *dbscaler* VHDL generic. Read only.

20:16 Reports number of bits, -1, for the PWM's dead band time counters. Value of the *dbbits* VHDL generic - 1. Read only.

15:13 Reports number of implemented scalers, -1. Value of the *nscalers* VHDL generic - 1. Read only.

12:8 Reports number of bits for the scalers, -1. Value of the *sbits* VHDL generic - 1. Read only.

7:3 Reports number of bits for the PWM counters, -1. Value of the *pbits* VHDL generic - 1. Read only.

2:0 Reports number of implemented PWMs. Value of the *npwm* VHDL generic - 1. Read only.

70.3.5 Capability Register 2

*Table 1112.*0x10 - CAP2 - Capability register 2

31 11	10	9 6	5 1	0
R	wsync	wabits	wdbits	wpwm
0	*	*	*	*
r	r	r	r	r

31:11 Reserved, always zero

10 1 if Waveform PWM synch signal generation is implemented, 0 if not. Value of VHDL generic *wsync*. Read only

9:6 Reports the number of address bits - 1 used for the waveform RAM. Value is log2(*wdepth*) - 1, where *wdepth* is the VHDL generic *wdepth.* Read only.

5:1 Reports number of bits -1 for each word in the waveform RAM. Value of VHDL generic *wbits* - 1. Read only

0 1 if waveform PWM generation is implemented, 0 if not. Value of VHDL generic *wavepwm*. Read only

70.3.6 Waveform Configuration Register

*Table 1113.*0x14 - WCFG - Waveform configuration register

31	30	29	28	wabits +17	wabits +16	16	15	wabits +1	wabits	0
wsynccfg		wsen	R		wsynccomp		R		wstopaddr	
0		0	0		0		0		all 1	
rw		rw	r		rw		r		rw	

31:30 These bits are used to configure at which point in the PWM period matching the *wsynccomp* field (see description below) that the *wsync* ouput will be set high. 0b00 = The wsync ouput will be set at the start of the PWM period. 0b01 = The output will be set at the first compare match. 0b10 = If the *meth* bit in the *PWM Control Register* is set to one (symmetric) for the waveform PWM, then the output will be set at the middle of the PWM period. 0b11 = If the *meth* bit is set to one for the waveform PWM, then the output will be set at the second compare match.

29 Enables/disables the waveform sync signal. This bit is only present if the *wsync* bit in *Capability register 2* is set to 1. Reset value is 0b0

28:wabits+17 Reserved, always zero. wabits is the value of the *wabits* field in *Capability register 2*. Note that this field is not present if wabits is 12.

16+wabits:16 wabits is the value of the *wabits* field in *Capability register 2.* The number of words in the waveform RAM is the same as the maximum number of PWM periods that will occur before the waveform is restarted. The value of this field is used as an offset into the waveform PWM. A counter is increased every PWM period and when the counter matches this value the *wsync* output of the core will be set to 1 sometime during that period. These bits are only present if the *wsync* bit in *Capability register 2* is set to 1. Reset value is 0b0..0.

15:wabits+1 Reserved, always zero.

wabits:0 The value of this field is used by the core to wrap when accessing the waveform RAM. wabits is the value of the *wabits* field in *Capability register 2*. This field is reset to 0b1..1 (all ones) so that by default the core reads the whole RAM. If software wants to put a waveform in the RAM that does not fill the whole RAM it should set these bits to the address where the last waveform PWM compare value will be stored.

70.3.7 PWM Period Register

*Table 1114.*0x20 - PPERIOD - PWM period register

31	pbits	pbits-1	0
R		per	
0		0	
r		rw	

31:pbits Reserved, always zero. If pbits = 32 then this field is not present. (pbits is the value of the *pbits* generic)

(pbits-1):0 When the PWM counter reaches this value a PWM period has passed. Depending on the method used to generate the PWM the output could then be switched. When this register is written the actual PWM period value used inside the core is not updated immediately, instead a shadow register is used to hold the new value until a new PWM period starts. Reset value 0b0..0 (all zeroes).

70.3.8 PWM Compare Register

Table 1115. 0x24 - PCOMP - PWM compare register

31 pbits	pbits-1 0
R	comp
0	0
r	rw

31:pbits — Reserved, always zero. If pbits = 32 then this field is not present. (pbits is the value of the *pbits* generic)

(pbits-1):0 — When the PWM counter reaches this value the PWM output is switched. Depending on the method used to generate the PWM this register is used once or twice during each PWM period. When this register is written the actual PWM compare value used inside the core is not updated immediately, instead a shadow register is used to hold the new value until a new PWM period starts. Reset value 0b0..0 (all zeroes).

70.3.9 PWM Dead Band Compare Register

Table 1116. 0x28 - PDEAD - PWM dead band compare register

31 dbbits	dbbits-1 0
R	dbcomp
0	0
r	rw

31:dbbits — Reserved, always zero. If dbbits = 32 then this field is not present. (dbbits is the value of the *dbbits* generic)

(dbbits-1):0 — The dead band time has passed once the dead band counter reach the value of this field. When this register is written the actual compare value used inside the core is not updated immediately, instead a shadow register is used to hold the new value until a new PWM period starts. Reset value 0b0..0 (all zeroes).

70.3.10 PWM Control Register

Table 1117. 0x2C - PCTRL - PWM control register

31 27	26	25 22	21	20 15	14	13	12 10	9	8	7	6	5 3	2	1	0
R	flip	dbscaler	dben	irqscaler	irqt	irqen	scalersel	wen	dcen	pz	meth	fix	pair	pol	en
0	0	0	0	0	0	0	0	0	0	0	*	0	1	*	0
r	rw	rw	rw	rw	rw	rw	rw	rw*	rw*	r	rw	rw	rw	rw	rw

31:27 — Reserved, always zero.

26 — Output flip bit. When this bit is set to 0b1 the PWM outputs are flipped.

25:22 — Dead band scaler. These bits are used to scale the system clock when generating dead band time. This field is only present if the *dbscaler* generic is set to 1. When these bits are written the dead band scaler register inside the core is not updated immediately. Instead these bits are written to a reload register which updates the actual scaler when it underflows. This is done in order to prevent the dead band scaler register to change during the actual dead band time. Reset value is 0b0..0 (all zeroes).

21 — Dead band enable. 0b0 = Dead band time generation is disabled, no dead band time will be inserted when the PWM output switch from deactive to active. 0b1 = Dead band time will be inserted when the PWM output switch from deactive to active. Reset value is 0b0.

20:15 — Interrupt scaler. Determines how many compare/period matches that need to occur before an interrupt is generated. All zeroes means that an interrupt will occur every compare/period match, a one means that an interrupt will occur every second match etc. Note that when generating a symmetric PWM two compare matches occur during a PWM period but when generating an asymmetric PWM only one compare match occur during a period. Reset value is 0b0..0 (all zeroes).

14 — Interrupt type. 0b0 = Generate interrupt on PWM period match. 0b1 = Generate interrupt on PWM compare match. Reset value is 0b0.

13 — Interrupt enable/disable bit. 0b0 = Interrupt is disabled. 0b1 = Interrupt is enabled. Reset value is 0b0.

*Table 1117.*0x2C - PCTRL - PWM control register

12:10	Scaler select bits. These bits are used to select which of the system clock scalers that will be used when generating the current PWM. This field is only present when the *nscalers* generic is greater than 1. These bits can only be set if the PWM is disabled, i.e. *en* bit (see below) set to 0b0. Reset value is 0b000.
9	Waveform PWM enable. This bit can only be set if the current PWM is the PWM with the highest index (determined by the generic *npwm*) and if the *wavepwm* field in *Capability register 2* is set to 1. Also the PWM need to be disabled, i.e. *en* bit (see below) set to 0b0. When this bit is set the core will reload the internal PWM compare registers with values from the waveform RAM instead of values from the *PWM compare register*. Reset value is 0b0.
8	Dual compare mode enable. If this bit is set to 0b1 and the *meth* bit (see below) is set to 0b1 (symmetric) then the core will update its internal PWM compare register twice every PWM period, once when the counter is zero and once when a period match occur and the counter starts counting downwards again. In this way it is possible to have two different compare values, one when counter is counting upwards and one when counter is counting downwards. If this bit is 0b0 the compare register is only updated when the counter is zero. This bit has no effect if an asymmetric PWM is generated. Reset value is 0b0. This bit is only present if the *dcmode* bit in the *Capability register* is set.
7	When this pair_zero bit is set to 0b1 and the pair bit is set to 0b0 the complement output is always set to zero. When this bit is set to 0b0 and the pair bit is set to 0b0 the complement output is inactive (depending on the polarity). When the pair bit is set to 0b1, this bit has no function.
6	PWM generation method select bit. This bit selects if an asymmetric or symmetric PWM will be generated, where 0b0 = asymmetric and 0b1 = symmetric. . The asymmetric and symmetric methods are only available if the generics *asympwm* and *sympwm* respectively are set to 1. This bit can only be set if the PWM is disabled, i.e. *en* bit (see below) set to 0b0. The core prevents software from setting this bit to an invalid value. Reset value is 0b0 if asymmetric PWM is supported otherwise 0b1.
5:3	PWM fix value select bits. These bits can be used to set the PWM output to a fix value. If bit 3 is set to 0b1 then bit 4 decides what value the PWM output will have. If the *pair* bit (see below) is set to 0b1 while bit 3 is set to 0b1 as well then bit 5 determines what value the complement output will have. Reset value is 0b000.
2	PWM pair bit. If this bit is set to 0b1 a complement output for this PWM will be generated, creating a PWM pair instead of a single PWM. The complement output will be the first output's inverse, with the exception that dead band time might be added when the values switch from deactive to active. Reset value is 0b1.
1	PWM polarity select bit. 0b0 = PWM is active low, 0b1 = PWM is active high. This bit can only be set if the PWM is disabled, i.e. *en* bit (see below) set to 0b0. Reset value equals *defpol* bit in *Capability Register 1*.
0	PWM enable/disable bit. 0b0 = PWM is disabled. 0b1 = PWM is enabled. When this bit is set to 1 (from 0) and the *wen* bit (see bit 9 above) is set the core's internal address counter for the waveform RAM is reset. Reset value is 0b0.

70.3.11 Waveform RAM, Word X

*Table 1118.*0x8000 - 0xFFFC - Waveform RAM, word X

31 wbits+1	wbits 0
R	waveform data
0	NR
r	rw

31:wbits+1	Reserved, always zero. wbits is the value of the *wdbits* field in *Capability register 2.* Note that this field is not present if wbits = 31.
wbits:0	wbits is the value of the wdbits field in *Capability register 2*. Data in the waveform RAM at the address which the current register maps to can be read/written through these bits. This register can only be read/written if either the *wen* bit or *en* bit in the associated PWM's PWM control register are set to 0.

70.4 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x04A. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

70.5 Implementation

70.5.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. The registers driving PWM output are always implemented with asynchronous reset.

70.5.2 RAM usage

The core maps all usage of RAM on the *syncram* (or *syncramft* if *ft* generic is not set to 0) component from the technology mapping library (TECHMAP). RAM is only used if the core is configured with support for generation of a waveform PWM (*wavepwm* generic set to 1). The size of the instantiated RAM is determined by the *wbits* and *wdepth* generics. *wdepth* is the number of words that the RAM can hold, and *wbits* is the number of bits in each word. Fault tolerance - byte parity DMR or TMR - can be added to the RAM by setting the *ft* generic to 1 or 2. Note that the *ft* generic need to be set to 0 if the core is used together with the GPL version of GRLIB, since that version does not support any fault tolerance.

70.6 Configuration options

Table 1119 shows the configuration options of the core (VHDL generics).

Table 1119. Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR. Need to be set to 16#F00# or smaller if the generic *wavepwm* is set to 1.	0 - 16#FFF#	16#F00#
pirq	APB irq number.	0 - NAHBIRQ-1	0
memtech	Memory technology used for waveform buffer. This generic has no impact if *wavepwm* is 0.	0 - NTECH	inferred
npwm	Number of PWM outputs.	1 - 8	3
pbits	Number of bits used for each PWM.	1 - 32	16
sbits	Number of bits in the system clock scaler(s).	1 - 32	16
nscalers	Number of system clock scalers.	1 - 8	1
dbbits	Number of bits used for the dead band configuration for each PWM.	1 - 32	8
dbscaler	Decides if a scaler is implemented for the dead band configuration for each PWM. 1 = A four bit system clock scaler is implemented for each PWM. 0 = No scaling of the system clock when calculating the dead band time is implemented.	0 - 1	1
asympwm	Decides if assymetric PWM generation is implemented. This generic can not be set to 0 if the *sympwm* generic is set to 0 as well.	0 - 1	1
sympwm	Decides if symmetric PWM generation is implemented. This generic can not be set to 0 if the *asympwm* generic is set to 0 as well.	0 - 1	1

*Table 1119.*Configuration options

Generic name	Function	Allowed range	Default
dcmode	Enables dual compare mode. Core then supports updates of the PWM's compare registers twice during every (symmetric) PWM period. This generic has no effect if sympwm is set to 0.	0 - 1	0
wavepwm	Decides if the core implements support for generating a waveform PWM. If this generic is set to 1 then a RAM block of size *wdepth*wbits* bits will be instantiated.	0 - 1	1
wbits	The number of bits in each of the wdepth words in the internal RAM that holds the waveform. This generic has no impact if *wavepwm* is 0. This generic can not be larger than the *pbits* generic.	1 - 32	8
wdepth	The number of *wbits* wide words that need to fit in the internal buffer holding the waveform. If *wdepth* is not a power of two then the actual number of words that will fit in the buffer is the closest power of two above *wdepth*. This generic has no impact if *wavepwm* is 0.	1 - 8192	512
wsync	If this generic is set the core supports the generation of a synchronization signal. The synchronization signal can be configured to go active any time during the waveform PWM. This generic has no impact if *wavepwm* is 0.	0 - 1	1
sepirq	0 = One irq number (value of *pirq* generic) is used for all PWMs. 1 = Each PWM has it's own irq number, starting with the value of *pirq* and counting up to *pirq*+(*npwm*-1). 2 = The interrupt configuration depend on the *npwm* generic in the following way: If *npwm* < 3, each PWM has its own irq (*pirq* and possibly *pirq*+1). If *npwm* = 3 the PWMs share irq (*pirq*). If 3 < *npwm* < 6 the first three PWMs share irq (*pirq*) and the remaining PWM(s) have their own irq (*pirq*+1 and possibly *pirq*+2). If *npwm* >= 6 the first three PWMs share irq number *pirq,* the second three PWMs share irq number *pirq*+1, and (if implemented) the last two PWMs have their own irq (*pirq*+2 and *pirq*+3).	0 - 2	0
ft	This generic determines if fault tolerance should be added to the RAM that holds the waveform PWM. This generic has no impact if *wavepwm* is 0. 0 = no fault tolerance, 1 = Byte parity DMR, 2 = TMR. Note that this generic need to be set to 0 if the core is used together with the GPL verison of GRLIB, since that version does not include any fault tolerance.	0 - 2	0
defpol	This generic sets the default polarity of the PWM outputs. 0 = Active low polarity, outputs are high after reset/power-up. 1 = Active high polarity, outputs are low after reset/power-up.	0 - 1	1

70.7 Signal descriptions

Table 1120 shows the interface signals of the core (VHDL ports).

Table 1120. Signal descriptions

Signal name	Field	Type	Function	Active
rst	N/A	Input	Reset	Logical 0
clk	N/A	Input	Clock	-
apbi	*	Input	APB slave input signals	-
apbo	*	Output	APB slave output signals	-
o	pwm(x:0)**	Output	PWM signals	***
	wavesync****	Output	Waveform PWM synchronization signal	Logical 1
	tick(y:0)*****	Output	PWM synchronization tick outputs	Logical 1

* see GRLIB IP Library User's Manual

** The width depends on core configuration in the following way: x = <number of PWMs>*2-1 (<number of PWMs> = value of VHDL generic *npwm*)

*** Depends on core configuration.

**** Signal is only driven if the waveform PWM and and waveform sync functionality are implemented (VHDL generics *wavepwm* and *wsync* need to be set to 1).

***** The width depends on core configuration in the following way: y = <number of PWMs>-1 (<number of PWMs> = value of VHDL generic *npwm*)

70.8 Signal definitions and reset values

The signals and their reset values are described in table 1121.

Table 1121. Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
pwm(x:0)*	Output	PWM signals	**	**
wavesync***	Output	Optional synchronization signal	Logical 1	Logical 0
tick(y:0)****	Output	PWM synchronization tick outputs	Logical 1	Logical 0

* The width depends on core configuration in the following way: x = <number of PWMs>*2-1 (<number of PWMs> = value of VHDL generic *npwm*)

** Depends on core configuration.

*** Signal is only driven if the waveform PWM and and waveform sync functionality are implemented (VHDL generics *wavepwm* and *wsync* need to be set to 1).

**** The width depends on core configuration in the following way: y = <number of PWMs>-1 (<number of PWMs> = value of VHDL generic *npwm*)

70.9 Library dependencies

Table 1122 shows the libraries used when instantiating the core (VHDL libraries).

Table 1122. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	PWM	Signals, component	Component declaration
TECHMAP	GENCOMP	Constants, components	Components etc. for technology mapping.

70.10 Timing

The timing waveforms and timing parameters are shown in figure 168 and are defined in table 1123.

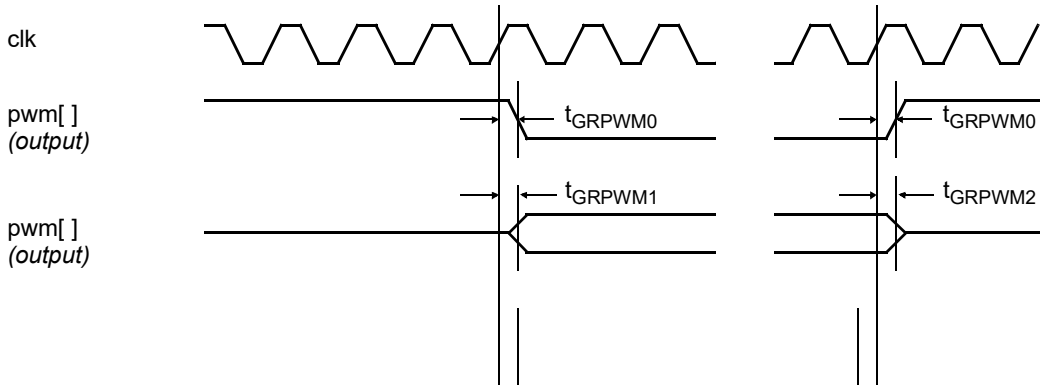


Figure 168. Timing waveforms

Table 1123. Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{GRPWM0}	clock to output delay	rising *clk* edge	TBD	TBD	ns
t_{GRPWM1}	clock to non-tri-state delay	rising *clk* edge	-	-	ns
t_{GRPWM2}	clock to tri-state delay	rising *clk* edge	-	-	ns

70.11 Instantiation

This example shows how the core can be instantiated. The instantiated core has all its generics, except *pindex*, *paddr*, and *pirq* at their default values. The impact of the generics can be seen in table 1119.

```
library ieee, grlib, gaisler;
use ieee.std_logic_1164.all;
use grlib.amba.all;
use gaisler.pwm.all;

entity grpwm_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;
    pwm : out std_logic_vector(5 downto 0)
    );
end;

architecture rtl of grpwm_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- GRPWM signals
  signal pwm : grpwm_out_type;

begin

  -- AMBA Components are instantiated here
  ...

  -- GRPWM core
  grpwm0 : grpwm
    generic map (pindex => 10, paddr => 10, pirq => 10)
    port map (rstn, clk, apbi, apbo(10), pwm);
```

```
  -- Pads for GRPWM core
  pwm_pad : outpadv generic map (tech => padtech, width => 6)
                  port map (pwmo, pwm.pwm);

end;
```

71 GRRT - MIL-STD-1553B / AS15531 Remote Terminal Back-End

71.1 Overview

This core provides the back-end logic for a MIL-STD-1553B Remote Terminal, taking care of the details of the bus protocol. The core provides a simple signaling interface, that can be converted by a front-end to bus master accesses, direct block-RAM, FIFO or register accesses, depending on requirements.

The RT supports single or dual-redundant buses, and all types of transfers allowed by the 1553B standard. The back-end includes 1553 codec, RT protocol handling state machines, and terminal fail-safe timers.

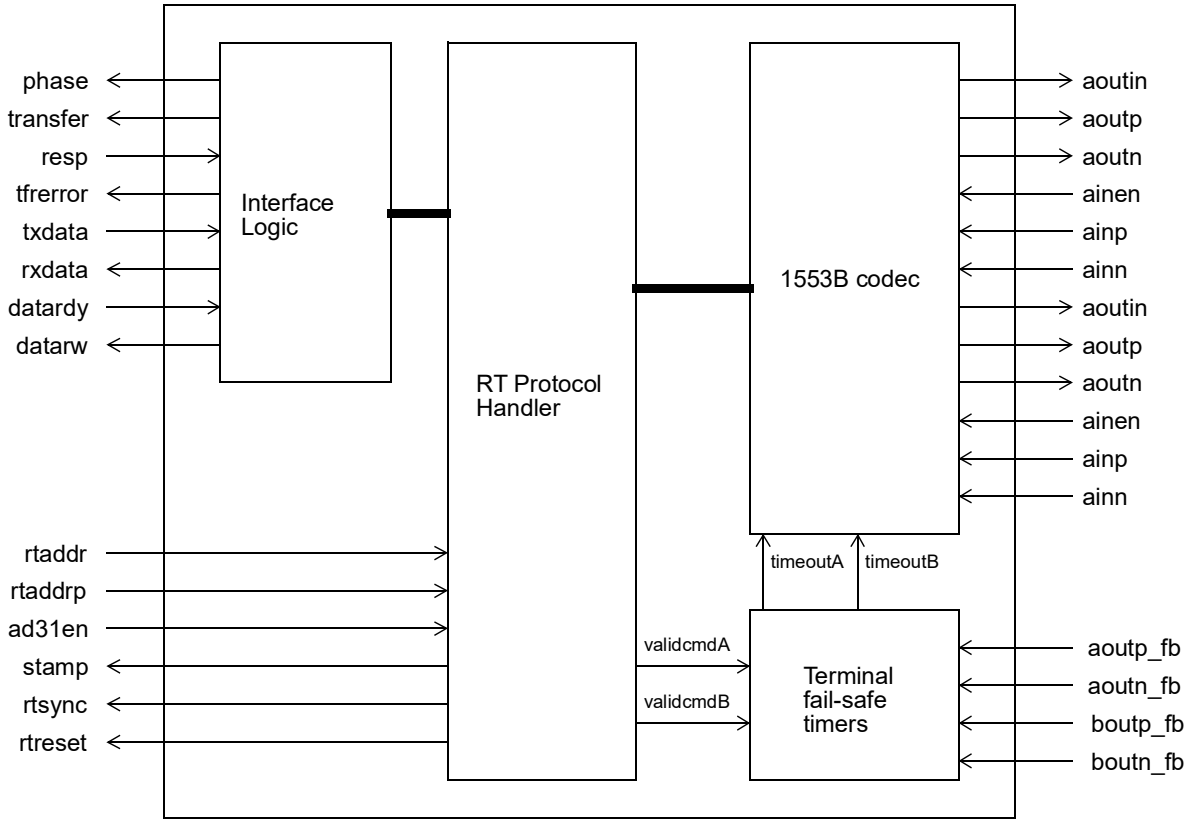


Figure 169. GRRT block diagram showing interfacing signals

71.2 Electrical interface

The core is connected to the MIL-STD-1553B bus wire through single or dual transceivers, isolation transformers and transformer or stub couplers as shown in figure 170. If single-redundancy is used, the unused bus receive P/N signals should be tied both-high or both-low. The transmit enable output is

inverted (called transmitter inhibit) as is the standard on most transceivers. See the standard and the respective component's data sheets for more information on the electrical connection.

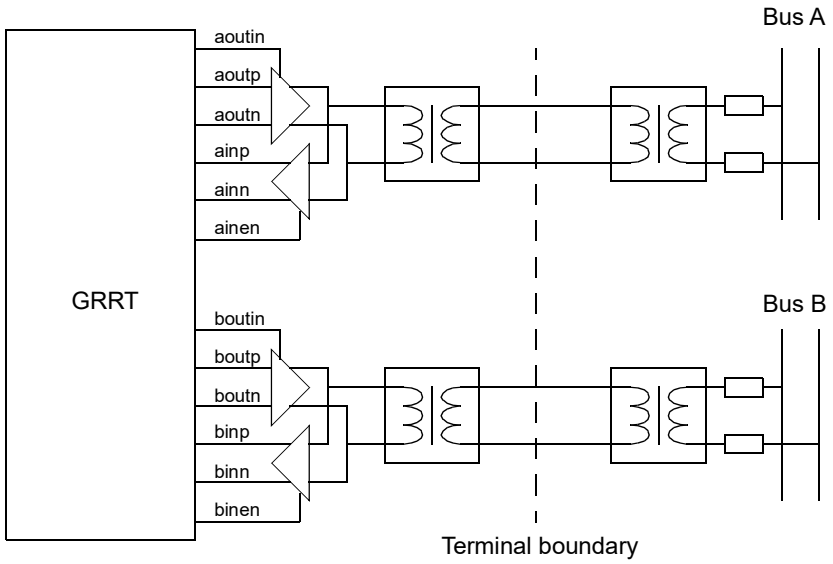


Figure 170. Interface between core and MIL-STD-1553B bus (dual-redundant, transformer coupled)

71.3 Operation

71.3.1 Address configuration

The core's RT address is set through RT address and parity input signals. If the parity is not odd (the exclusive-or of the RT address lines and parity line is zero), the back-end will not respond to commands from the bus.

There is also an input bootstrap signal ad31en, determining whether the address 31 should be treated as a normal RT address or as the broadcast address. If ad31en is high and the RT is configured to address 31, it will respond on this address. If ad31en is low, then the RT will use address 31 as broadcast address and never respond on this address regardless of address configuration.

*Table 1124.*Possble address configurations

ad31en	rtaddr	rtaddrp	Regular address	Broadcast address
0	0..30	Correct	=rtaddr	31
		Wrong	None	None
	31	Correct (0)	None	31
		Wrong (1)	None	None
1	0,2,...,30	Correct	=rtaddr	None
		Wrong	None	None
	31	Correct (0)	=rtaddr(=31)	None
		Wrong (1)	None	None

71.3.2 Transfer handling

Each transfer is processed in the following four stages:

- Idle (phase="00") - waiting for command on the bus, "transfer" signal outputs at undefined
- Legalize (phase="01") - Waiting for front-end to respond to legalization request by setting resp inputs to legal (resp="01") or illegal (resp="10").
- Transfer (phase="11") - Transferring data words.

- Commit (phase=”10”) - Transfer completed, successfully if tferror=0. Waiting for frontend to deassert resp input, then goes back to idle.

After the front-end has legalized or illegalized the command by setting resp to “01” or “10”, it should keep the resp signal set until the transfer has completed. In the idle and commit stages, the response should be set back to unknown (“00”). The core will delay going from idle to legalize state if the response signal is already set. This acts as a handshake and prevents earlier responses from getting used multiple times.

71.3.3 Data transfers

During the transfer phase, data words are transferred depending on direction (transfer bit 10).

For receive transfers, received data words will be available on the rxdata output when the datarw signal is high. For transmit transfers, outgoing data words are read from the txdata input when the datarw signal is high.

To provide flow control, the datardy input signals whether the user logic is ready for the datarw signal to be asserted.

The number of words to be transferred is given by bits 4:0 of the transfer output signal, all all-zero value indicates 32 data words. An exception to this is for mode commands with data, where the word count is always 1. In case of error, less words than expected may be transferred.

71.3.4 Error handling

Normally, the best way to handle errors in an RT is to ignore the failed command and let the BC perform the error recovery appropriate for the system. The design of the 1553 bus means the BC can always see an error occurred so there is no need to signal this separately.

To simplify the user logic design, transfers will always go through all the four phases and spend at least one cycle in each state, even if an error occurs during the transfer. If an error occurs during the transfer or the request was illegalized by the user logic, the tfrerror output signal is asserted and kept high until the core has left the commit phase.

71.3.5 Response-time requirements

The core will wait for the user logic in three cases:

1. In the legalize phase, waiting for command legalization.

2. In the transfer phase, waiting for the datardy signal to go high.

3, In the commit phase, waiting for the resp signal to go low.

In these cases, the user logic must respond within a limited time in order for the core to continue. Failing to meet these limits will result in the RT not responding to the command or not sending the full word count. It does not cause any permanent error so after the user logic has caught up, it will again work as before.

For the datardy signaling, the maximum tolerated delay is 20 us (400 cycles at 20 MHz), which is the time it takes to send a data word on the 1553 bus.

To analyze the maximum value tolerated for the legalization interface one must consider the bus switching requirement, where another command may arrive on the other bus at any time and override the current command. This could happen while an old transfer is still in the legalize phase. Since the GRRT always processes commands through all four phases, the old command must be legalized, then go to the commit phase and wait for resp to go low, then the new command needs to be legalized. All this must complete within 8.5 us (170 cycles at 20 MHz).

71.3.6 Mode commands

The following mode commands without external data are supported by the core and can not be illegalized:

- Synchronize - will assert the rtsync output one cycle
- Transmit status word - transmits the current value of the 1553 status word
- Transmit last command - transmits the current value of the 1553 status word followed by the last valid command word received by the RT.
- Transmitter shutdown - Shuts down the transmitter on the other bus than the one where the command
- Override transmitter shutdown - Cancels an earlier transmitter shutdown command
- Inhibit terminal flag - Masks the terminal flag bit
- Override inhibit terminal flag bit - Cancels an earlier inhibit terminal flag command
- Reset remote terminal - Clears the transmitter shutdown and inhibit terminal flag status, and asserts the rtreset output for one cycle. The front-end can use this to reset the core, if appropriate.

They will be handled internally by the core, keeping the phase at idle state during the command.

The following mode commands without data are not supported and will be treated as an illegal command (the core will respond with the message error status bit set):

- Dynamic bus control (only applicable to backup bus controllers)
- Initiate self test (no self test implemented)

71.3.7 Mode commands with external data

Three mode commands with external data are supported by the core:

- Synchronize with data word - One received data word is transferred
- Transmit vector word - One data word to transmit is transferred
- Transmit BIT word - One data word is transferred

These will be handled using the same four-phase scheme as regular data transfers, however to identify the mode command bits 9:5 of the transfer output will be set to “00000” or “11111” and bits 4:0 are set to the mode command number. As for regular transfers, it is also possible for the front-end to illegalize these two commands if they are not implemented.

Not supported (always illegalized) mode commands with data are:

- Selected Transmitter Shutdown (not applicable for dual-redundant RTs)
- Override Selected Transmitter Shutdown (not applicable for dual-redundant RTs)

71.3.8 Timestamping

The core has a stamp output which is asserted for one cycle when the command word is received. This is mainly useful together with the rtsync for accurately time-stamping the synchronize mode command for time synchronization purposes.

71.4 Implementation

71.4.1 Clocking

The core operates in two clock domains, the front-end clock domain and the codec clock domain, with internal synchronization between the domains. The codec clock must be 20 or 24 MHz, configured

via the codecfreq generic, but the front-end clock can be any frequency from 10 MHz and up. All the signals interfacing the front-end are synchronous to the front-end clock.

If the front-end clock is the same (20 or 24 MHz) clock as the codec, the generic sameclk can be set to 1 to remove the internal synchronization registers to save some area.

71.4.2 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core has two separate reset inputs for the two clock domain's registers. The resets can be configured as either synchronous or asynchronous.

71.5 Configuration options

Table 1125 shows the configuration options of the core (VHDL generics).

Table 1125. Configuration options

Generic	Function	Allowed range	Default
codecfreq	Codec clock frequency, 20 or 24 MHz	20 or 24	20
sameclk	Set to 1 if codec clock and user clock are tied to the same clock signal (removes synchronization registers)	0-1	1
syncrst	Chooses between synchronous ('1') or asynchronous ('0') reset.	0 - 1	1

71.6 Signal descriptions

Tables 1126 shows the interface signals of the core (VHDL ports).

Table 1126. Signal descriptions on AMBA side

Signal name	Type	Function	Active
CLK	Input	Clock, front-end clock domain	-
RST	Input	Reset, deasserted synchronously to CLK	Low
CLK1553	Input	Codec clock	
RST1553	Input	Reset, deasserted synchronously to CLK1553	Low
RTADDR	Input	RT address (bootstrap signal)	-
RTADDRP	Input	Odd parity for RTADDR (bootstrap signal)	-
RTSTAT	Input	RT status bits Bit 3 - Service request Bit 2 - Busy Bit 1 - Subsystem flag Bit 0 - Terminal flag	
AD31EN	Input	Address 31 configuration: 0 = Address 31 is broadcast address 1 = Address 31 is valid normal RT address Should be kept constant during operation	-
RTSYNC	Output	Asserted for synchronize mode command	High
RTRESET	Output	Asserted for reset remote terminal mode command	High
STAMP	Output	Asserted when receiving a valid command word	High

Table 1126. Signal descriptions on AMBA side

Signal name	Type	Function	Active
PHASE	Output	Transfer phase: 00 = Idle 01 = Legalize 11 = Transfer 10 = Commit	-
TRANSFER	Output	Transfer description, see section Bit 11 - Broadcast Bit 10 - TX/RX Bit 9:5 - Subaddress / mode code indicator Bit 4:0 - Word count/mode code	-
RESP	Input	Transfer legality response: 00 = Unknown/Idle 01 = Legal 10 = Illegal	-
TFRERROR	Output	Asserted when an error occurs during transfer	
TXDATA	Input	Data for transmit (RT-to-BC) commands	-
RXDATA	Output	Data for receive (BC-to-RT) commands	-
DATARDY	Input	Ready for read/write	High
DATARW	Output	Data word read or write	High
AOUTIN	Output	Bus A Transmitter Inhibit	High inhibit, Low enable
AOUTP	Output	Bus A Transmit Data, Positive	-
AOUTN	Output	Bus A Transmit Data, Negative	-
AINEN	Output	Bus A Receiver Enable	High
AINP	Input	Bus A Receive Data, Positive	-
AINN	Input	Bus A Receive Data, Negative	-
BOUTIN	Output	Bus B Transmitter Inhibit	High inhibit, Low enable
BOUTP	Output	Bus B Transmit Data, Positive	-
BOUTN	Output	Bus B Transmit Data, Negative	-
BINEN	Output	Bus B Receiver Enable	High
BINP	Input	Bus B Receive Data, Positive	-
BINN	Input	Bus B Receive Data, Negative	-
AOUTP_FB	Input	Feedback signal for fail-safe timers, tie to corresponding output signal.	-
AOUTN_FB			
BOUTP_FB			
BOUTN_FB			

71.7 Library dependencies

Table 1127 shows libraries used when instantiating the core (VHDL libraries).

Table 1127. Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	GR1553B_PKG	Component	Component declaration

Frontgrade Gaisler AB
Kungsgatan | SE-411 19 | Goteborg | Sweden
+46 31 7758650 | frontgrade.com/gaisler

72 GRSCRUB - AMD/Xilinx external FPGA scrubber controller

72.1 Overview

Devices in the space environment are vulnerable to radiation-induced particles, and SRAM-based FPGAs are particularly susceptible to Single Event Upsets (SEU) that may affect the configuration memory. Soft errors in the FPGA configuration memory can change the device functionality and lead to errors and malfunction in the system. The scrubbing method is essential to avoid an accumulation of upsets in the FPGA configuration memory.

The GRSCRUB is an external FPGA scrubber controller responsible for programming and monitoring the FPGA configuration memory. The GRSCRUB IP is currently compatible with the Kintex UltraScale and Virtex-5 Xilinx FPGA families. The scrubbing functionality can be set to target the entire FPGA configuration memory or just a defined memory area.

The scrubbing mitigation technique only fixes bit-flips in the FPGA configuration memory, being up to the user to apply any additional method to mask errors and re-establish the state of the system. Scrubbing does not cover soft-errors affecting User memory data. All dynamic data stored in memory elements, such as shift-registers (SRL), LUT RAMs, and Block RAMs (BRAM), are not verified by the GRSCRUB.

Figure 171 shows the system block diagram. The GRSCRUB accesses the FPGA configuration memory through the SelectMap interface. In addition, the GRSCRUB accesses through an AMBA AHB or AXI4 bus a Golden memory that can be ROM or RAM. The original configuration bitstream is stored in the Golden memory, and it is used both to configure the FPGA at start-up and to repair the FPGA configuration memory in case of errors. The Golden memory also stores the mask data and the Cyclic Redundancy Check (CRC) codes used to check the configuration bitstream integrity. If a ROM memory is used, all the required data must be previously stored, such as the frame addresses and CRC codes, as further described, since some functionalities of the GRSCRUB that writes on the memory will not be executed.

The GRSCRUB registers can be accessed and configured through the AMBA APB bus.

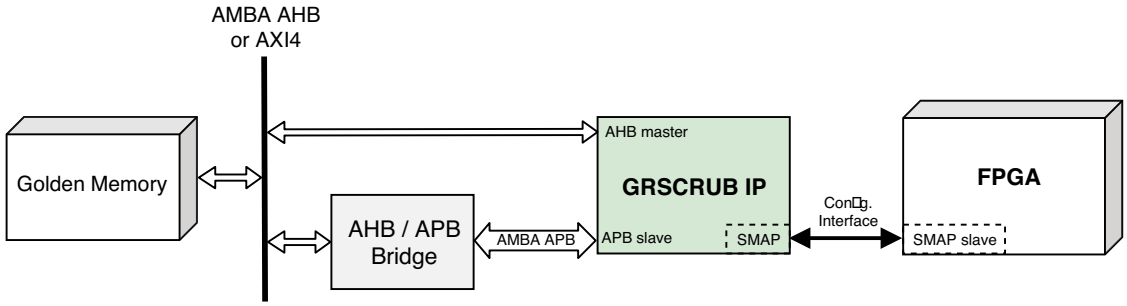


Figure 171. System block diagram

The SelectMap interface must be enabled in slave mode at the target FPGA to allow external control from the GRSCRUB. Figure 172 shows an example of the GRSCRUB and slave SelectMap connection in a Xilinx UltraScale FPGA. In the example, the GRSCRUB port signals are directly attached to the SelectMap pins. The system provides the SelectMap clock to the FPGA configuration interface (CCLK) and the GRSCRUB (SMAPCLKI). As further described, the GRSCRUB works in two clock domains, CLK that is the system clock, and SMAPCLKI that is the SelectMap clock used for synchronization. In addition, the GRSCRUB has a clock enable signal that controls whether the CCLK should be enabled or not. The FPGA mode pins M[2:0], which select the FPGA configuration interface used, must be configured to slave SelectMap. Thus, these pins must be connected to M[2:0] pins from the GRSCRUB or directly tied to high/low levels externally. For more details, see the documentation for the Xilinx FPGA family. Also, refer to the FPGA Data Sheet to define the proper voltage connection.

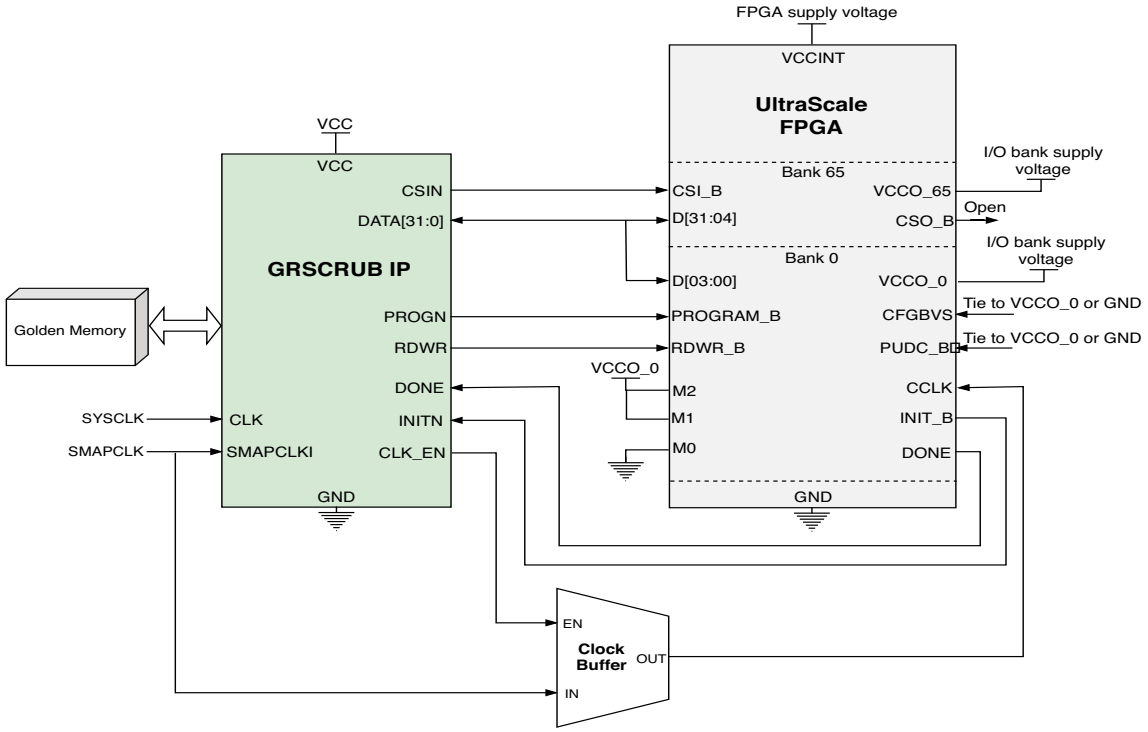


Figure 172. Example of GRSCRUP and slave SelectMap interface connection for Xilinx UltraScale FPGA

72.2 Soft error mitigation

Radiation-induced soft errors are errors provoked by radiation particles that affect the design without damaging the device permanently. SEUs, or bit-flips, in the FPGA configuration memory may lead to persistent errors in the system, changing the architectural implementation of the design. Soft errors can also affect the memory data, registers, and flip-flops, and cause errors in the system outputs. Single Events Transients (SET) are transient pulses that propagate through the combinational logic and may be captured by a memory cell. The Single Event Functional Interrupt (SEFI) occurs when a soft error affects the control logic or a state register and leads to hangs or crashes in the design.

The GRSCRUB targets soft errors affecting the FPGA configuration memory. It does not avoid bit-flips from happening or its effects on the design. The GRSCRUB aims to maintain the configuration memory consistent by repairing the logic and correcting bit-flips, avoiding the accumulation of faults. Memory elements that store dynamic data, such as BRAMs, distributed memory, and flip-flops, are not protected by the GRSCRUB. The scrubbing technique is able to verify only the static data in the configuration memory. The configuration memory stores the architectural logic information of a design and defines the function, behavior, and connectivity of each block. The GRSCRUB must be combined with additional fault mitigation methods at the user level to increase the overall system reliability.

Soft errors affecting the dynamic elements can be mitigated by applying fault tolerance techniques such as redundancy or Error Correction Code (ECC). Triplicating the logic is an efficient method to cope with the effects of single faults in the design. Additional user level techniques can also be applied to deal with Silent Data Corruptions (SDC) that are incorrect results outputs. Moreover, periodic reset may be required to reestablish the system state, and restore the initial state of flip-flops. Since SEFIs may also affect internal control elements of the FPGA or the configuration interface, a complete power cycle might be required to restore the system.

72.3 Operation

The GRSCRUB main operational modes are programming the FPGA and scrubbing the configuration memory correcting errors. Figure 173 describes the basic flow of these two operational modes (OPMODE). After system reset, the GRSCRUB remains in the idle state until the EN bit of the Configuration Mode Register (CONFIG) is set high. If the programming mode is selected, the GRSCRUB starts the configuration sequence to program the FPGA. At the end of the programming phase, the OPDONE bit of the Status Register (STATUS) is set high, and the GRSCRUB return to the idle state.

For the scrubbing operation mode, there are two types of execution: blind and readback scrubbing. Before enabling scrubbing, it is required to map the FPGA frame addresses by executing the Mapping operation mode or previously saving them on the Golden memory. The scrubbing can be defined as periodic with a delay between each memory scrubbing, or just one time. If a periodic execution is configured in the registers, the GRSCRUB repeats the scrubbing operation after a configured delay and only if it is still enabled. Otherwise, the OPDONE bit of the Status Register (STATUS) is set, and the GRSCRUB return to the idle state.

The following sections describe in more detail the GRSCRUB operational modes.

72.3.1 GRSCRUB operational modes

The GRSCRUB has five operation modes that can be selected in the OPMODE of the Configuration Mode Register (CONFIG) and are described in table 1128. The Idle mode is the default state of the GRSCRUB. The operational mode must be configured before, or at the same time, the GRSCRUB is enabled. Also, the OPDONE and SCRERR bits in the Status Register (STATUS) must be cleared, see section 72.13.1 for more details.

Note: After enabling the GRSCRUB to run an operation mode, the values of the configuration registers must not be changed. Otherwise, it can cause an error in the GRSCRUB execution. If the GRSCRUB is disabled during execution, it will stop the operation, send the command to desynchronize the FPGA slave SelectMap, and return to the idle state. Thus, the current operation mode will be aborted.

Table 1128. GRSCRUB operation modes description

Operation Mode	Code	Description
Idle	0000	Idle mode (default after reset).
Programming	0001	FPGA configuration. Program the configuration bitstream into the FPGA.
Scrubbing	0010	FPGA scrubbing. Perform the blind or readback scrubbing.
Mapping	0011	FPGA mapping frames address.
Golden CRC	0100	In this OPMODE the GRSCRUB computes a golden copy of the CRC data check and saves it on the Golden memory.

72.3.2 Idle mode

After reset, the GRSCRUB stays in idle mode waiting to be enabled. When the EN bit of the Configuration Mode Register (CONFIG) is disabled (logically 0), the GRSCRUB remains in idle mode.

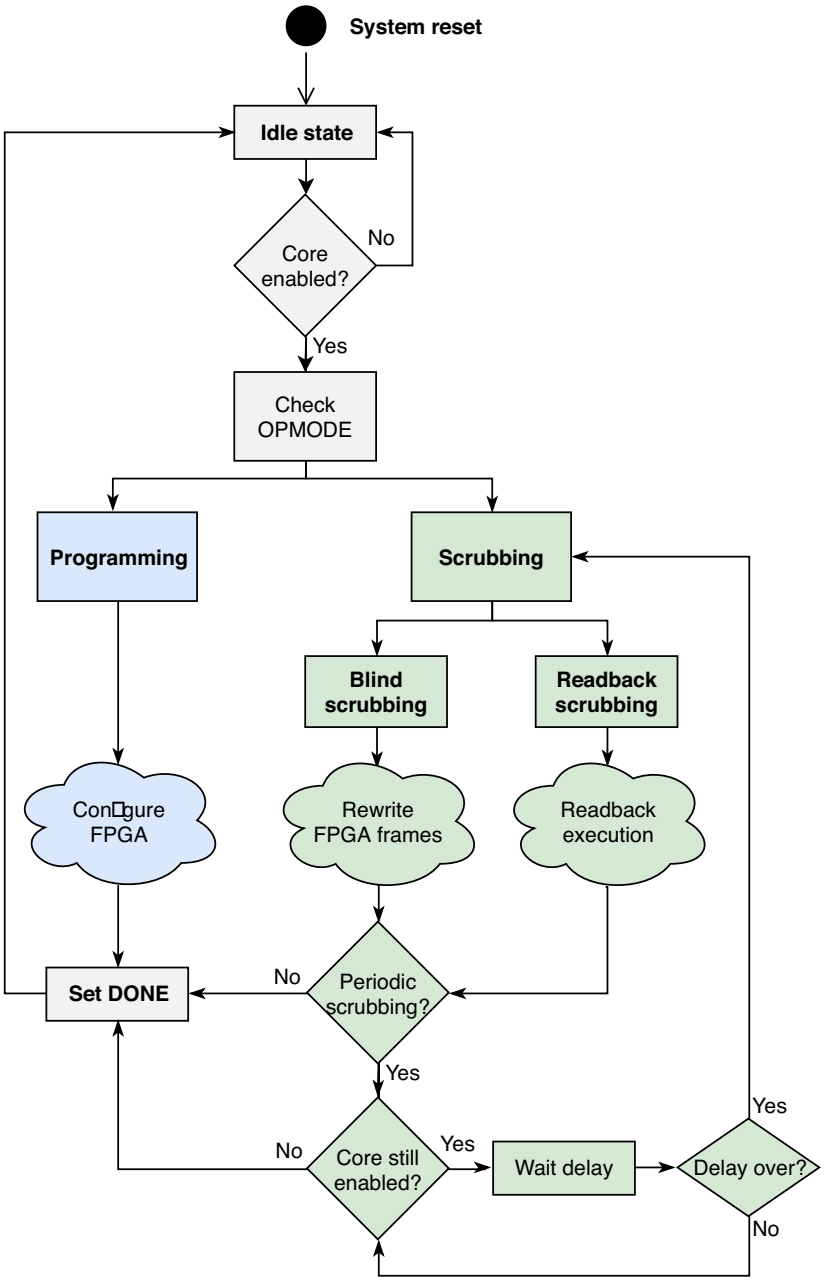


Figure 173. GRSCRUB flow for programming and scrubbing operation modes

72.3.3 Programming mode

The GRSCRUB can configure the FPGA by programming the configuration bitstream through the configuration interface. The configuration bitstream must be stored into the Golden memory, and the configuration bitstream address must be set in the GRSCRUB registers.

The following steps must be performed before enabling the GRSCRUB to execute the programming mode:

- The configuration bitstream must be stored in the Golden memory;
- The GRSCRUB must be disabled to configure the registers;
- The OPDONE and SCRERR bits in the Status Register (STATUS) must be cleared;
- Set the Configuration Mode Register (CONFIG) with the programming OPMODE (0001);

- Set the low and high configuration bitstream addresses in the Low Golden Bitstream Address Register (LGBAR) and High Golden Bitstream Address Register (HGBAR), respectively;
- Set the FPGA Device Identifier Register (IDCODE).

After setting the registers, the GRSCRUB can be enabled by writing the logical 1 in the EN bit of the Configuration Mode Register (CONFIG). When the programming phase is finished, the OPDONE bit goes to high, and an interrupt is generated (if interrupts are enabled).

72.3.4 Scrubbing mode

The user can define the scrubbing operational mode as blind or readback. Since the FPGAs are frame-oriented, the configuration memory is verified considering the frame size. For instance, the frame length of the Xilinx UltraScale FPGA family is 123 words of 32 bits each, whereas it is 41 32-bit words for Virtex-5. The FPGA frame length and the total number of frames must be configured in the Frame Configuration Register (FCR).

A scrubbing run is defined as one execution of the blind or readback scrubbing in all the selected configuration frames. The scrubbing can be set to be periodic, which means that scrubbing runs are periodically executed in time.

Note: The scrubbing mitigation technique only fixes bit-flips in the FPGA configuration memory, it is up to the user to apply any additional method to mask errors and re-establish the state of the system. In addition, scrubbing does not cover soft-errors affecting User memory and dynamic data.

The steps below must be performed before enabling the GRSCRUB to execute the scrubbing mode. If a register has been previously set and the data has not changed, there is no need to reconfigure it.

- The configuration bitstream must be stored in the Golden memory;
- The frames addresses must be stored in the Golden memory;
- The FPGA must be previously programmed;
- The GRSCRUB must be disabled to configure the registers;
- The OPDONE and SCRERR bits in the Status Register (STATUS) must be cleared;
- Set the Configuration Mode Register (CONFIG) with the scrubbing OPMODE (0010);
- Set whether the scrubbing mode is blind (logical 0) or readback (logical 1) in the SCRM bit of the Configuration Mode Register (CONFIG). Blind scrubbing is set by default;
- Set whether the scrubbing run is just one time (logical 0) or periodic (logical 1) in the SCRUN bit of the Configuration Mode Register (CONFIG). One time is set by default. If a periodic run is set, the Delay Register (DELAY) should also be configured. No delay between runs is set by default;
- Set the low and high configuration bitstream addresses in the Low Golden Bitstream Address Register (LGBAR) and High Golden Bitstream Address Register (HGBAR), respectively;
- Set the address for the first golden frame in the Low Golden Start Frame Address Register (LGSFAR). Note that the first golden frame address is different from the low bitstream address. The LGSFAR should be the address in the Golden memory that saves the first word of the first frame in the bitstream;
- Set the address for the first frame of the FPGA configuration memory to be scrubbed in the Low Frame Address Register (LFAR);
- Set the FPGA Device Identifier Register (IDCODE);
- Set the number of frames and the frame length in the Frame Configuration Register (FCR).

Additional configuration is required for readback mode:

- The mask data must be stored in the Golden memory;

- Set the low mask address in the Low Golden Mask Address Register (LMASKAR);
- Set which data check is enabled to detect errors in the FFCEN and CRCEN bits of the Configuration Mode Register (CONFIG);
- Set if the readback scrubbing mode is only detection (logical 1) or detection and correction (logical 0) in the CORM bit of the Configuration Mode Register (CONFIG). Detection and correction is set by default;
- Set the Low Golden Frame Mapping Address Register (LFMAPAR) with the low address of the mapped frames in the Golden memory (only required if correction is selected);
- Optionally, to save the readback data in the Golden memory, the WRBKEN bit of the Configuration Mode Register (CONFIG) must be enabled. Also, the low memory address to save the readback data must be set in the Low Golden Readback Address Register (LGRBKAR). These steps are not required for other modes. Note that if the WRBKEN bit is enabled, the GRSCRUB does not detect or correct any error, it only copies the configuration data read from the FPGA to the Golden memory.

After setting the registers, the GRSCRUB can be enabled by writing 1 in the EN bit of the Configuration Mode Register (CONFIG). The OPDONE bit is only asserted if one time scrubbing is selected. The SCRUND bit goes high after each scrubbing execution.

I. Blind scrubbing mode

In the blind scrubbing operation mode, the GRSCRUB will rewrite each FPGA frame from the configuration memory without any verification. Thus, the blind scrubbing does not provide the error detection capability. The blind scrubbing is configured to reprogram the configuration memory frames set in the Frame Configuration Register (FRC) and Low Frame Address Register (LFAR). Thus, the total number of frames, frame length, and the starting FPGA frame address must be configured. Only frames from the configuration block must be set. The frames from the FPGA configuration block (type 000) includes the configuration elements CLBs, I/Os, and CLKs. Blind scrubbing does not rewrite User memory elements. See the FPGA Configuration Manual for more details about frame block types.

II. Readback scrubbing mode

In the readback scrubbing operation mode, the GRSCRUB verifies the integrity of each FPGA frame of the configuration memory, and then, in case of errors, rewrites the frame with correct data from the Golden memory. The error detection can be performed through CRC verification, and by comparing all frame bits with the golden copy. The latter is here defined as Full Frame Check (FFC). Each type of verification can be configured to be enabled or not. The CRC and FFC data checks are further described in the next sections.

The GRSCRUB can operate in three types of readback scrubbing modes: save the readback data into Golden memory; only error detection; and error detection and correction. For error detection operation, the mask data must be previously stored in the Golden memory.

The mode of saving the readback data into memory can only be used if the Golden memory is RAM, and it is useful when the user needs to check the readback data. The Low Golden Readback Data Address Register (LGRBKAR) must be set. On this mode, the mask data is not required.

The correction mode is used to correct a faulty frame during readback scrubbing. The GRSCRUB replaces the erroneous frame in the FPGA by the golden frame read from the Golden memory. To enable the error correction, the CORM bit in the Configuration Mode Register (CONFIG) must be set to 0. By default, the correction is enabled. However, if the CORM bit is set to 1, the errors are only detected and not corrected in the readback phase.

Figure 174 describes the GRSCRUB flow in scrubbing operation mode for readback execution with error detection and correction enabled. The GRSCRUB starts reading a frame word from the FPGA configuration memory. If CRC is enabled, the GRSCRUB computes the CRC signature of the word. In sequence, the word is compared with the golden copy, if FFC is enabled. At the end of the frame

the CRC code is checked and the frame is corrected if an error is detected. For FFC, the frame is corrected immediately when a faulty word is identified. After correcting a frame, the same frame is reread and rechecked to ensure the erroneous bits are fixed. If the corrected frame still presenting an error, it is defined as an uncorrectable error, and the next frame is read. In case of uncorrectable errors, the UNCORRECTABLE_BIT_ERROR is set, the SCRERR bit is asserted, and it is recommended to reprogram the FPGA.

Differently from the blind scrubbing, the readback mode allows detecting errors and correcting the frame only if necessary, at the expense of more accesses to the Golden memory to read golden and mask data.

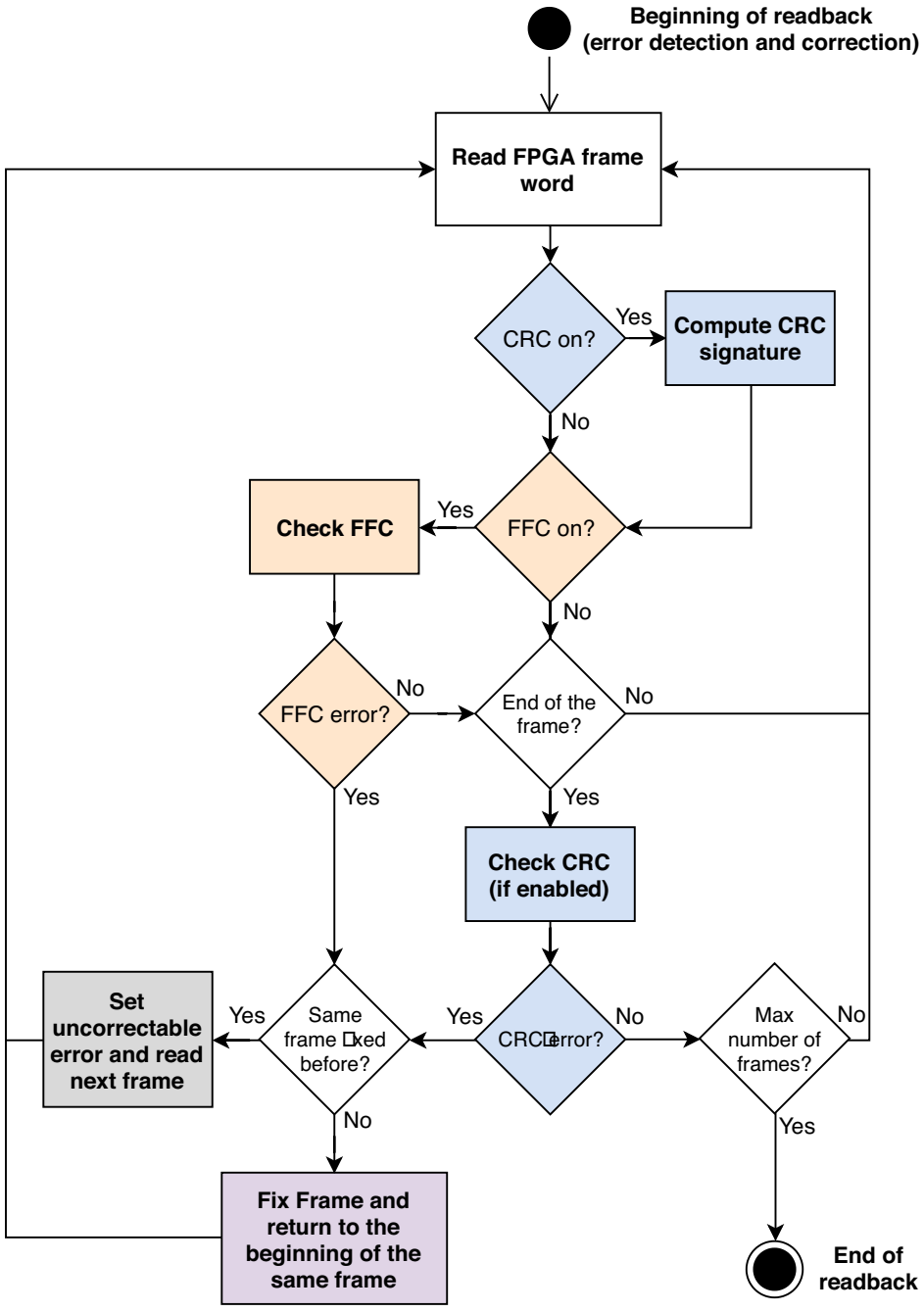


Figure 174. GRSCRUB flow in readback scrubbing mode for error detection and correction

i. Cyclic Redundancy Check (CRC)

CRC is an error detection code that applies redundancy to check inconsistencies. A customized 32-bit CRC checksum algorithm is computed for each FPGA frame.

The CRC32 polynomial is:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x^{1} + 1$$

The defined initial CRC signature for each frame is 0xFFFFFFFF. The first readback frame word is masked, using the corresponding mask word, then the CRC32 is applied using the signature. The output of this computation results in the next signature for the next readback frame word. That cycle is repeated for all words in the frame (frame length). The last computed signature corresponds to the frame CRC code.

During regular readback operation, the GRSCRUB reads the frame word from the FPGA, reads the mask word from the Golden memory, and computes the CRC signature. When it reaches the end of the frame, the generated CRC code is compared with the golden code from the Golden memory. If an error is detected, the faulty frame is corrected (if correction is enabled). Otherwise, the next frame is verified.

The CRC mode does not provide the total numbers of errors in the frame, and neither the error position. The number of errors detected present in Number of Errors Detected Register (ECNT) is related to the number of detected faulty frames. The CRC method has single error detection ensured. However, for more accurate error counting, the FFC method is recommended.

Before performing readback with CRC checking, the golden CRC codes must be previously saved into the Golden memory. The default configuration of the GRSCRUB assumes that the CRC codes are already saved on the Golden memory. Additionally, the GRSCRUB has the functionality of computing and saving the golden CRC codes, see section 72.3.6 for more details.

ii. Full Frame Check (FFC)

In the FFC mode, the GRSCRUB compares all the configuration frame words with the golden words. Thus, the number of error bits per readback word is known. However, when the correction is enabled, the frame is corrected whenever the first error is detected. Thus, even though more faulty bits might be present, only one error bit is counted for the frame. FFC is a more reliable method to detect errors then CRC since each word is compared with its golden. Therefore, all faulty bits can be detected.

72.3.5 Mapping mode

Before scrubbing the FPGA in readback mode with correction, the addresses of the configuration frames must be stored in the Golden memory. These addresses are required to fix a specific frame in the FPGA. The default configuration of the GRSCRUB assumes that the addresses are already saved on the Golden memory.

Additionally, the GRSCRUB has the functionality of mapping the frames and saving the addresses on the memory. Only frames defined as configuration block are mapped, which means that all block memories are excluded from the mapping. It is recommended first to power reset and program the FPGA before executing the mapping mode. Also, the Golden memory must be RAM to allow writing the addresses.

Depending on the FPGA internal architecture, there may exist dummy frames between rows addresses in the FPGA configuration memory. The called row boundary is an internal FPGA address alignment that must be considered during the mapping phase. Since these addresses are not accessible from the GRSCRUB, all dummy frame addresses are indicated as 0x00000000. The number of frames in the rows boundaries must be set in the ROWBND bit of FPGA Setup Register (SETUP). For instance, Xilinx Virtex-5 has two boundary frames between rows.

The steps below must be performed before enabling the GRSCRUB to execute the mapping mode. If a register has been previously set and the data has not changed, there is no need to reconfigure it.

- The GRSCRUB must be disabled to configure the registers;
- The OPDONE and SCRERR bits in the Status Register (STATUS) must be cleared;
- Set the Configuration Mode Register (CONFIG) with the mapping OPMODE (0011);
- Set the low mapping address in the Low Golden Frame Mapping Address Register (LFMAPAR);
- Set the address for the first frame of the FPGA configuration memory to be mapped in the Low FPGA Frame Address Register (LFAR). For instance, set the register to 0x00000000 to start from the first frame in the configuration memory.
- Set the number of rows boundaries in the ROWBND of the FPGA Setup Register (SETUP).
- Set the FPGA Device Identifier Register (IDCODE);
- Set the number of frames and the frame length in the Frame Configuration Register (FCR).

After setting the registers, the GRSCRUB can be enabled by writing the logical 1 in the EN bit of the Configuration Mode Register (CONFIG). When the mapping phase is finished, the OPDONE bit goes to high, and an interrupt is generated (if interrupts are enabled).

72.3.6 Golden CRC mode

The golden CRC codes must be stored in the Golden memory before enabling the readback scrubbing with CRC data check, see section 72.3.4 for more details about CRC. The GRSCRUB has the operation mode to compute and save the golden CRC codes. The Golden memory must be writable to allow writing the CRC golden codes.

The following steps must be performed before enabling the GRSCRUB to execute the golden CRC operational mode:

- The FPGA must be programmed;
- The frames addresses must be stored in the Golden memory;
- The GRSCRUB must be disabled to configure the registers;
- The OPDONE and SCRERR signals in the Status Register (STATUS) must be cleared;
- Set the Configuration Mode Register (CONFIG) with the golden CRC OPMODE (0100);
- Set the address for the first frame of the FPGA configuration memory to be scrubbed in the Low FPGA Frame Address Register (LFAR);
- Set the FPGA Device Identifier Register (IDCODE);
- Set the number of frames and the frame length in the FCR register;
- Set the low address for the first CRC code in the LCRCAR;
- The mask data must be previously stored in the Golden memory;
- Set the low mask address in the Low Golden Mask Address Register (LMASKAR).

After setting the registers, the GRSCRUB can be enabled by writing the logical 1 in the EN bit of the configuration register. When the GRSCRUB finishes the CRC computation, the OPDONE bit goes to high, and an interrupt is generated (if interrupts are enabled).

72.4 Mask data information

In the FPGA configuration memory there are dynamic bits that change their values during design execution, such as LUTRAMs and shift registers (SRL). During the configuration bitstream generation, the synthesis tool creates a mask file that indicates all dynamic bits as '1'. These bits should not be ver-

ified in the error detection steps. For that, the GRSCRUB applies the mask word to the readback word before the data checking, following steps below:

1) Read mask word from Golden memory;
2) All bits of the mask word are inverted: *NOT mask_word*;
3) An *AND* operation of the readback word and the inverted mask word is performed.

- *Example:*

  mask_word: 0x0000FFFF
  readback_word: 0x0FE50400
  Result: readback_word AND (NOT mask_word) = 0x0FE50000
  Thus, only 0x0FE50000 is applied to the fault detection methods.

The mask data must be saved on the Golden memory and the start address should be set in the Low Golden Mask Address Register (LMASKAR). Since the GRSCRUB accesses only the masked data from the configuration frames (i.e., configuration block 000), it is not required to store the entire file in the Golden memory. However, the mask words must exactly corresponding to the readback words. Thus, the first mask word, whose address is set in the LMASKAR register, is related to the first word of the first readback frame, whose address is set in the Low FPGA Frame Address Register (LFAR).

The mask file has the same format as the configuration bitstream file. If the user decides to store only the mask words related to the scrubbed frames, the correlation between both must be matched, misalignment would cause erroneous verification of the bits.

Since the masked bits are not verified in the readback mode, faults affecting those bits cannot be detected.

72.5 FPGA frame information

Table 1129 shows the configuration bitstream and fame length for Xilinx Virtex-5 and Kintex UltraScale FPGAs as example.

Note: Each FPGA device has different configuration bitstream length, number of frames, and frame length. See the FPGA documentation for more information.

*Table 1129.*Configuration bitstream information for Xilinx Virtex-5 and Kintex UltraScale FPGAs

	Virtex-5 (XC5VFX130T)	**Kintex UltraScale (KU060)**
Configuration bitstream length	6,154,368 bytes	24,124,908 bytes
Total number of frames	37,520 frames	49,030 frames (plus 537 overhead words)
Block configuration frames[ab]	25,980 frames (4,260,720 bytes)	37,499 frames (18,449,508 bytes)
Frame length	41 32-bit words	123 32-bit words

a.Only the configuration frames are verified (block type 000: CLBs, DSPs, and IOBs).
b.Block RAM contents are not included (block type 001).

72.6 Golden memory

The GRSCRUB requires access to a Golden memory in which the golden data must be stored. The memory can be ROM or RAM, and the access is made through the AMBA AHB bus. The stored data is related to the golden configuration bitstream, mask file, mapped frame addresses, and golden CRC codes. Figure175 describes an example of data saved in the Golden memory and the GRSCRUB registers that must be configured, as follows:

- Golden configuration bitstream: The entire configuration bitstream must be saved on memory. The Low Golden Bitstream Address Register (LGBAR) and High Golden Bitstream Address Register (HGBAR) correspond to the addresses of the first and last configuration bitstream words, respectively. The Low Golden Start Frame Address Register (LGSFAR) is the address in the Golden memory of the first word of the first frame to be scrubbed.

- Mask data: The required mask data is directly related to the scrubbed frames. The Low Golden Mask Address Register (LMASKAR) is the address of the first mask word of the first frame word to be scrubbed.
- Mapped frames addresses: The mapped addresses are also directly related to the scrubbed frames. The Low Golden Frame Mapping Address Register (LFMAPAR) is the address in the Golden memory of the first mapped frame to be scrubbed.
- CRC codes: One CRC code is saved per frame. The Low Golden CRC Address Register (LCRCAR) is the address in the Golden memory of the first golden CRC code of the first frame to be scrubbed.
- The FCNT and FLEN bitfields of the Frame Configuration Register (FCR) are the number of scrubbed frames and length of each frame, respectively. Therefore, they represent the number of memory positions that must be sequentially accessed by the GRSCRUB in each operation.

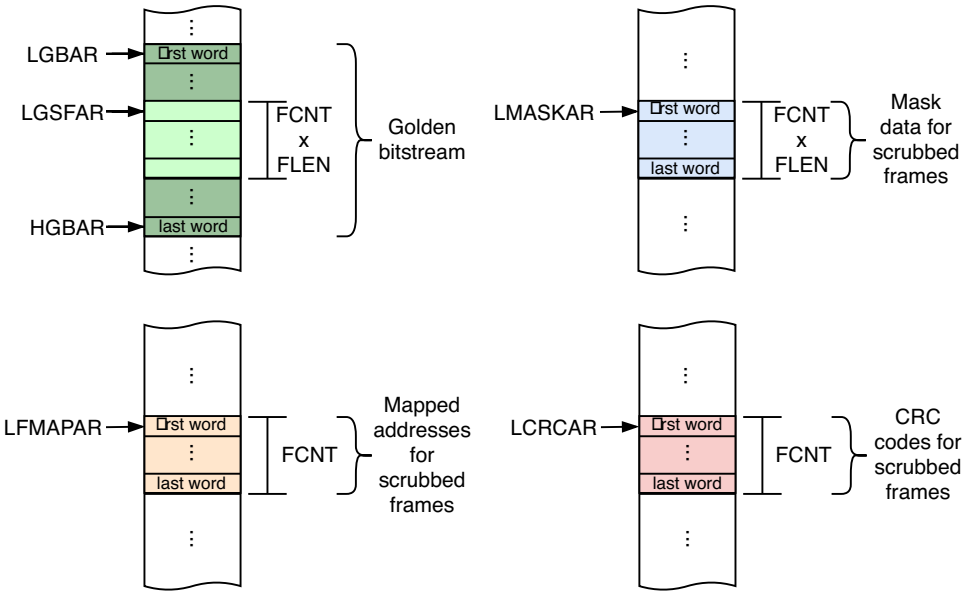


Figure 175. Example of data storage configuration in the Golden memory and the relation with the GRSCRUB registers

As an example, Table 1130 describes the storage length required for Xilinx Virtex-5 and Kintex UltraScale FPGAs. Note that each FPGA device has different configuration bitstream length. See the FPGA documentation for more information.

Table 1130. Description of required Golden memory storage for Virtex-5 and Kintex UltraScale FPGAs

Data Memory	**Virtex-5 [XC5VFX130T] (bytes)**	**Kintex UltraScale [KU060] (bytes)**
Configuration bitstream	6,154,368	24,124,908
Mask[a]	4,260,720	18,449,508
Mapped frame addresses[b]	103,920	149,996
CRC codes	103,920	149,996
Total required storage	**10,645,661**	**42,907,220**

a.Since the GRSCRUB accesses only the mask data related to the configuration frames, it is not required to store the entire file.
b.Only configuration frames are mapped.

72.7 SelectMap configuration interface

The FPGA slave SelectMap configuration interface is used to load the configuration bitstream into the FPGA and for scrubbing operations. The GRSCRUB connects directly to the SelectMap pins, which are controlled and used as follows:

- **CCLK:** The SelectMap configuration clock is provided by the system, and it is controlled by the GRSCRUB clock enable signal (SMAPO.clk_en). By stopping the CCLK, it is possible to pause the data loading or reading from the SelectMap. The system design must provide a buffer to enable/disable the CCLK based on SMAPO.clk_en. The implementation details of the required setup are further described. See the FPGA Data Sheet to details about the maximum slave SelectMap CCLK frequency.
- **PROGRAM_B:** This signal is used to clean the FPGA configuration logic before starting the programming phase. The GRSCRUB set the PROGRAM_B to low during the number of clock cycles defined in the TPROG register. See in the FPGA documentation the minimum pulse width required. By default, TPROG is 150 clock cycles.
- **INIT_B:** It indicates when the FPGA is ready to receive configuration data. It also indicates any configuration or readback errors by low pulse.
- **DONE:** The GRSCRUB monitors the FPGA DONE signal to identify the completion the configuration sequence.
- **M[2:0]:** The configuration mode pins must be configured to logical "110" to select the slave SelectMap interface. For that, the pins can be tied directly to high or low level, or connected to the M pins of the GRSCRUB.
- **D[31:0]:** The bidirectional data bus is used for configuration and scrubbing operations. The RDWR_B signal defines the direction of the pins. SelectMap supports 8-bit (D[7:0]), 16-bit (D[15:0]), or 32-bit (D[31:0]) data bus widths. The bus width must be informed in the smap_buswide configuration option of the GRSCRUB, which has independent input and output data buses. The system is responsible for multiplexing the data signals, based on RDWR_B, and provide the connection to the SelectMap bidirectional data bus.
- **RDWR_B:** It is used to select the direction of the data bus. When RDWR_B is high (logical 1), the SelectMap data pins are outputs (reading from the FPGA). Otherwise, the bus is input (loading data to the FPGA). The GRSCRUB only changes the RDWR_B when the SelectMap interface is disabled (i.e., when CSI_B is logically 1).
- **CSI_B:** This signal enables (logical 0) and disables (logical 1) the SelectMap interface.

In order to allow the GRSCRUB to access and control the slave SelectMap interface, the generated FPGA configuration bitstream must be configured following the requirements presented in table 1131.

Table 1131. Example of configuration bitstream settings for enabling the FPGA slave SelectMap interface

Synthesis tool	- Vivado Design Suite 2018.1; or - ISE Design Suite 14.7
Configuration bitstream settings	- Enable the mask file generation - Do not prohibit readback in the configuration bitstream security settings - Do not use encryption in the configuration bitstream - Set the SelectMap pins to persistent in the configuration bitstream generator (bitgen)[a]: -g Persist:Yes or BITSTREAM.CONFIG.PERSIST YES - Set the constraints to allow programming and readback the FPGA through slave SelectMap interface: CONFIG CONFIG_MODE=S_SELECTMAP32+READBACK

a. The ICAP interface cannot be used in the FPGA design if Persist:Yes is set.

72.8 Soft-errors affecting the FPGA configuration interface

The FPGA configuration interface might also be affected by soft-errors, which may lead to catastrophic results during the scrubbing operation. For instance, if during a blind scrubbing the FPGA frame address register is affected and its value is changed to other valid address, all the following frames would be wrongly overwritten, and the design would be compromised.

Therefore, the GRSCRUB verifies the integrity of the FPGA configuration interface before each new scrubbing run. The verification is performed by reading a specific frame and checking its address. If the returned address is precisely the expected, the interface is considered responsive, and the scrubbing run starts. Otherwise, the CI_INTEGRITY_ERROR is reported. In this case, it is recommended to power cycle the FPGA. To enabling the verification feature, the CICHECK bit in the Setup Register (SETUP) must be 1.

Additionally, a safer approach is to set up the configuration interface for each scrubbed frame, instead of configuring all frames at once. For instance, writing one frame at time during blind scrubbing avoids overwritten the entire memory in case of errors in the FPGA frame address register. The BLKFRAME bitfield in the Setup Register (SETUP) defines the number of frames to be scrubbed sequentially during blind scrubbing or readback. If BLKFRAME is 0, all frames are scrubbed at once. The default is scrubbing only one frame at time (BLKFRAME = 1).

72.9 Interrupts

Interrupts are generated to indicate task finished (when the OPDONE bit is asserted) and internal errors in the GRSCRUB (when the SCRERR bit is asserted). Additionally, when the SCRUND bit is asserted, an interrupt can be generated.

Interrupts are disabled by default. The IRQDEN, IRQEEN and IRQSDEN bits of the Configuration Mode Register (CONFIG) should be high to allow interrupts for OPDONE, SCRERR, and SCRUND, respectively. All interrupts are connected to the interrupt controller to inform the processor of the events. The interrupts are pulse signals, being asserted for one clock cycle.

The usual procedure is that an interrupt routine handles the interrupt bits in the Status Register (STATUS). When it is finished, it clears the status bits by writing one, and the GRSCRUB can be enabled again.

72.10 GRSCRUB error codes

Table 1132 describes the GRSCRUB internal error codes. When an internal error occurs, the GRSCRUB set the ERRID bit of the Status Register (STATUS) with the error code, set the SCRERR to high, and launches an interrupt (if interrupts are enabled).

For all errors, except for UNCORRECTABLE_BIT_ERROR, the GRSCRUB safely stops the current operation, finishes the synchronization with the FPGA, waits for the end of the last memory access, and returns to the idle state. The SCRERR bit must be cleared before enabling the GRSCRUB again and starting a new operation. The UNCORRECTABLE_BIT_ERROR means that the GRSCRUB failed to correct an FPGA frame during readback operation. Then, this error is reported, and the GRSCRUB continues the scrubbing operation.

The DMA errors are related to the Golden memory and are most likely to occur due to invalid memory accesses. For the other errors, it is recommended to properly reset and reprogram the FPGA.

Table 1132. GRSCRUB internal error codes

Error	Code	Description
NO_ERROR	00000	GRSCRUB without errors. The GRSCRUB is working correctly. The SCRERR signal remains low, and no interrupt is launched.

Table 1132. GRSCRUB internal error codes

Error	Code	Description
PROGRAM_ERROR	00001	Error programming the FPGA configuration bitstream. This error is triggered when the INIT_B signal from the slave SelectMap interface pulses to low during the programming phase, which means error to configure the FPGA. Possible causes: - Frame length and number of frames wrongly set in the register; - Wrong configuration bitstream data stored in the Golden memory; - Wrong addresses set in the registers; - Wrong TPROG.
UNCORRECTABLE_BIT_ERROR	00010	Uncorrectable error(s) in the configuration memory during readback. After correcting a faulty frame, the replaced frame still presents an error. That might occur if the GRSCRUB failed to correct the frame or a new error affected the same frame in the time between correction and verification. Moreover, some upsets require two sequential scrubbing cycles to be corrected. The SRAM-based FPGAs present two types of configuration memory errors: a single point error is defined when an upset affects a function logic bit, and a burst of errors is defined when a single upset directly affects the state of multiple bits. The burst of errors may occur due to upsets in configuration bits responsible for controlling the status of other bits in the FPGA. When a control bit is flipped, all other bits under its control are also affected and may only return to the correct state after the original control bit is corrected. In the worst-case scenario, it is required two scrubbing cycles to recover all bits and return the configuration memory to a consistent state. The first scrubbing cycle will report UNCORRECTABLE_BIT_ERROR, and the second cycle will correct all remaining bits. This is the only internal error that does not lead the GRSCRUB to return to the idle state. The readback scrubbing continues the execution. In the UE_FRMID bitfield of the Error Frame Id Code Register (ERRFRMID) is informed the frame id where the last uncorrectable error occurred. The register is overwritten at each uncorrectable error.
DMA_MEM_READ_ERROR	00011	DMA error reading the Golden memory. Possible cause: - Illegal access.
DMA_MEM_WRITE_ERROR	00100	DMA error writing the Golden memory. Possible causes: - Illegal access. - PROM memory.
READBACK_ERROR	00101	Error during readback operation. This error is triggered when the INIT_B signal from the slave SelectMap interface pulses to low during the readback phase, which means error to read the FPGA frames. In the Frame Id Code Register (FRAMEID) is informed the id of the last frame scrubbed. Possible causes: - Frame length and number of frames wrongly set in the register; - Wrong addresses set in the registers; - The FPGA is not programmed; - Frames addresses are not stored in the Golden memory.

Table 1132. GRSCRUB internal error codes

Error	Code	Description
FRAME_MAPPING_ERROR	00110	Error during mapping operation. This error is triggered when the INIT_B signal from the slave SelectMap interface pulses to low during the mapping phase, which means error to read the frame content or the address. In the Frame Id Code Register (FRAMEID) is informed the id of the last frame mapped. Possible causes: - Frame length and number of frames wrongly set in the register; - Wrong addresses set in the registers.
WRITE_FRAME_ERROR	00111	Error to write an FPGA frame. Error due to correcting frame during readback; or error during blind scrubbing. In the Frame Id Code Register (FRAMEID) is informed the id of the last written frame. Possible causes: - Frame length and number of frames wrongly set in the register; - Wrong configuration bitstream data stored in the Golden memory; - Wrong addresses set in the registers; - The FPGA is not programmed; - Frames addresses are not stored in the Golden memory.
CI_INTEGRITY_ERROR	01000	Error during verification of the configuration interface The configuration interface is not responsive or its integrity is compromised. Possible causes: - The FPGA configuration interface is not correctly set. For instance, the bitstream does not configure the SelectMap as specified. - The FPGA configuration interface was affected by soft-errors. It is recommended to power cycle the FPGA.

72.11 Enabling and disabling the GRSCRUB

Before enabling the GRSCRUB, all the registers must be configured according the specific operation mode set. The GRSCRUB is enabled when the EN bit of the Configuration Mode Register (CONFIG) goes to high. The EN bit must stay high while the GRSCRUB is executing the operation. At the end of the execution, the GRSCRUB sets the OPDONE bit to high, sets the EN bit to low, and returns to the idle state.

After enabling the GRSCRUB, the registers should not be written. Otherwise, that can provoke a malfunction in the operation sequence. In addition, if the GRSCRUB is disabled (i.e., by setting the EN bit to low) during execution, the operation will not be finished. In that case, the GRSCRUB stops the current operation, finishes the synchronization with the FPGA, waits for the end of the last memory access, and returns to the idle state. The EN bit should only be enabled again after the GRSCRUB has returned to idle state.

The OPDONE and SCRERR bits should always be cleared before enabling the GRSCRUB and starting a new operation.

72.12 Bus master interface

72.12.1 Protocol support

The GRSCRUB features a generic bus master interface to increase its flexibility. Additionally, Frontgrade Gaisler provides wrappers for both AMBA AHB 2.0 and AXI4.

72.12.2 Bus access

The access width for both writing and reading is a 32-bit word. By default, the GRSCRUB uses a wrapper for AMBA AHB 2.0 features a bridge to convert the generic bus master into an AHB bus master interface.

The wrapper for AXI4 features a bridge to convert the generic bus master into an AXI4 bus master interface. For further information regarding the accesses, contact Frontgrade Gaisler.

72.13 Registers

The GRSCRUB is programmed through registers mapped into APB address space, which are shown in table 1133. Only 32-bit single-accesses to the registers are supported.

Table 1133. GRSCRUB registers

APB address offset	Register description
0x00	Status register (STATUS)
0x04	Configuration mode register (CONFIG)
0x08	FPGA device identifier register (IDCODE)
0x0C	Delay register (DELAY)
0x10	Frame configuration register (FCR)
0x14	Low FPGA frame address register (LFAR)
0x18	Low golden bitstream address register (LGBAR)
0x1C	High golden bitstream address register (HGBAR)
0x20	Low golden start frame address register (LGSFAR)
0x24	Low golden mask address register (LMASKAR)
0x28	Low golden frame mapping address register (LFMAPAR)
0x2C	Low golden CRC address register (LCRCAR)
0x30	Low golden readback data address register (LGRBKAR)
0x34	Number of errors detected register (ECNT)
0x38	FPGA setup register (SETUP)
0x3C	Capability register (CAP)
0x40	Frame id code register (FRAMEID)
0x44	Error frame id code register (ERRFRMID)

72.13.1 Status Register (STATUS)

Table 1134. 0x00 - STATUS - Status register

31 13	12	11	10 6	5	4	3 0
RESVD	SCRUND	HOLD	ERRID	SCRERR	OPDONE	STATE
0	0	0	0	0	0	0
r	wc	r	r	wc	wc	r

31: 13	RESVD	RESERVED
12	SCRUND	Scrubbing run done. It indicates the end of a scrubbing run. The SCRUND bit goes high after each scrubbing execution in periodic run. Write one to clear. Generate interrupt (if enabled).
11	HOLD	GRSCRUB in hold during periodic scrubbing (read only) It identifies when the GRSCRUB is waiting during the delay period. See the DELAY Register for more details. 0 = release; 1 = GRSCRUB is holding due to delay
10: 6	ERRID	Internal error code (read only) It is cleared when SCRERR is set to 0. Table 1132 presents all error codes.
5	SCRERR	GRSCRUB internal error: 0 = no error; 1 = error Write one to clear. Generate interrupt (if enabled). See ERRID to error details.
4	OPDONE	Operation mode completed Write one to clear. Generate interrupt (if enabled).
3:0	STATE	Current state (read only) It indicates the current operation mode of the GRSCRUB. STATE = OPMODE

72.13.2 Configuration Mode Register (CONFIG)

Table 1135. 0x04 - CONFIG - Configuration mode register

31 14	13	12	11	10	9	8	7 4	3	2	1	0
RESVD	IRQSD EN	FFC EN	CRC EN	WRBK EN	IRQE EN	IRQD EN	OPMODE	CORM	SCRM	SCRUN	EN
0	0	0	0	0	0	0	0	0	0	0	0
r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31: 14	RESVD	RESERVED
13	IRQSDEN	Interrupt enable for SCRUND signal: 0 = disable; 1 = enable (pulsed interrupt)
12	FFCEN	Data check FFC enable: 0 = disable; 1 = enable
11	CRCEN	Data check CRC enable: 0 = disable; 1 = enable
10	WRBKEN	Enable writing readback data to Golden memory: 0 = disable; 1 = enable If enabled, the GRSCRUB does not detect or correct errors.
9	IRQEEN	Interrupt enable for SCRERR signal: 0 = disable; 1 = enable (pulsed interrupt)
8	IRQDEN	Interrupt enable for OPDONE signal: 0 = disable; 1 = enable (pulsed interrupt)
7: 4	OPMODE	Operational mode 0000 = idle mode (default) 0001 = programming mode 0010 = scrubbing mode 0011 = mapping mode 0100 = golden CRC mode

		others = reserved For more details, see the operational modes section.
3	CORM	Correction mode: 0 = detection and correction (default); 1 = only detection
2	SCRM	Scrubbing operational mode: 0 = blind (default); 1 = readback Blind scrubbing is set by default.
1	SCRUN	Scrubbing run: 0 = one time (default); 1 = periodic In the periodic execution is considered the delay between runs, and the OPDONE only goes to high if the GRSCRUB is disabled during the delay period (when the HOLD bit is high). The SCRUND bit goes high after each scrubbing execution in periodic run. One time scrubbing is set by default.
0	EN	GRSCRUB enable - starts operation mode: 0 = disable; 1 = enable After enabling the GRSCRUB, EN bit must be preserved high until the end of the operation (when OPDONE bit goes to high). Otherwise, the GRSCRUB is disabled, and the operation is not finished. See section 72.11 for more details.

FFCEN and CRCEN data check selection only works if enabled in the GRSCRUB generics. See Capability Register (CAP).

72.13.3 FPGA Device Identifier Register (IDCODE)

*Table 1136.*0x08 - IDCODE - FPGA device identifier register

31	0
IDCODE	
0	
rw	

31: 0 IDCODE FPGA device identifier code

The 32-bit FPGA identification code is defined in the FPGA specification and represents the specific device id based on the IEEE Std 1149.1 (JTAG).

72.13.4 Delay Register (DELAY)

*Table 1137.*0x0C - DELAY - Delay register

31	0
DELAY	
0	
rw	

31: 0 DELAY Delay, in number of ticks, between full blind or readback scrubbing cycles. The delay is set only for periodic scrubbing runs, and it is applied after all configured frames being scrubbed. Therefore, after the last scrubbed frame, the GRSCRUB waits the number of ticks set in the DELAY register, then returns to the first frame configured, and restarts the scrubbing operation. No delay is set by default.
A tick is counted for each rising edge of the tick_in input signal of the GRSCRUB. The period in which the GRSCRUB stays in hold depends on the tick_in frequency directly.

During the delay period while the GRSCRUB is waiting, the HOLD bit of the Status Register (STATUS) is set to high. When the delay is over, the HOLD bit is set to low.

If the EN bit of the Configuration Mode Register (CONFIG) is disabled while the GRSCRUB is waiting during the delay period, the GRSCRUB stops the delay, sets the OPDONE bit, and returns to idle state.

72.13.5 Frame Configuration Register (FCR)

Table 1138. 0x10 - FCR - Frame configuration register

31 25	24 9	8 2	1 0
RESVD	FCNT	FLEN	RESVD
0	0	0	0
r	rw	rw	r

31: 25 RESVD RESERVED
24: 9 FCNT Number of FPGA memory frames to be scrubbed
8: 2 FLEN Frame length: number of 32-bit words in a frame
1: 0 RESVD RESERVED

72.13.6 Low Frame Address Register (LFAR)

Table 1139. 0x14 - LFAR - Low frame address register

31 0
LFAR
0
rw

31: 0 LFAR The lowest frame address in the FPGA range to be scrubbed. Thus, this is the starting frame address.

72.13.7 Low Golden Bitstream Address Register (LGBAR)

Table 1140. 0x18 - LGBAR - Low golden bitstream address register

31 0
LGBAR
0
rw

31: 0 LGBAR The lowest golden configuration bitstream address in the Golden memory. This must be the address of the first dummy word in the configuration bitstream (0xFFFFFFFF), after the initial header.

All configuration bitstreams have an initial header with ASCII characters that provides some file information, which it is not required to program the FPGA. The synchronization phase starts at the first dummy word (0xFFFFFFFF).

72.13.8 High Golden Bitstream Address Register (HGBAR)

Table 1141. 0x1C - HGBAR - High golden bitstream address register

31 0
HGBAR
0
rw

31: 0 HGBAR The highest golden configuration bitstream address in the Golden memory.

72.13.9 Low golden start frame address register (LGSFAR)

*Table 1142.*0x20 - LGSFAR - Low golden start frame address register

31	0
LGSFAR	
0	
rw	

31: 0 LGSFAR The lowest address to the first configuration bitstream frame in the Golden memory.

72.13.10Low Golden Mask Address Register (LMASKAR)

*Table 1143.*0x24 - LMASKAR - Low golden mask address register

31	0
LMASKAR	
0	
rw	

31: 0 LMASKAR The lowest mask address in the Golden memory.

72.13.11Low Golden Frame Mapping Address Register (LFMAPAR)

*Table 1144.*0x28 - LFMAPAR - Low golden frame mapping address register

31	0
LFMAPAR	
0	
rw	

31: 0 LFMAPAR The lowest address of the FPGA frame mapping in the Golden memory.

72.13.12Low Golden CRC Address Register (LCRCAR)

*Table 1145.*0x2C - LCRCAR - Low golden CRC address register

31	0
LCRCAR	
0	
rw	

31: 0 LCRCAR The lowest address of the CRC data check in the Golden memory.

72.13.13Low Golden Readback Data Address Register (LGRBKAR)

*Table 1146.*0x30 - LGRBKAR - Low golden readback data address register

31	0
LGRBKAR	
0	
rw	

31: 0 LGRBKAR The lowest address to save readback data in the Golden memory.

72.13.14Number of Errors Detected Register (ECNT)

*Table 1147.*0x34 - ECNT - Number of errors detected register

31 16	15 0
UECNT	RECNT
0	0
rw	rw

31: 16 UECNT — Number of uncorrectable errors. UECNT is the number of frames that still presenting an error after correction, which means that the GRSCRUB was not able to correct the previous error or a new error occurred in the same frame in the time between correction and verification.
Only active in readback scrubbing mode and when correction is enabled (CORM=0).

15: 0 RECNT — Number of errors detected. Only enable in readback scrubbing mode. If the error correction is enabled (CORM=0), this counter represents the number of frames detected with error. Since the frame is corrected when the first error is detected, just one error is counted in the frame. If only detection is enabled (CORM=1) with FFC, this counter represents the total of error bits detected in the configuration memory. However, only one error is counter per frame for CRC data check.

The register is only cleared at the system reset, and the error counters accumulate over scrubbing runs. The user should clear the register to initiate a new count.

72.13.15FPGA Setup Register (SETUP)

*Table 1148.*0x38 - SETUP - FPGA setup register

31	30	29 22	21	20	19 12	11 4	3 0
RESVD	CICHECK	BLKFRAME	BITSWPEN	CISEL	TPROG	ROWBND	RESVD
0	1	1	1	0	150	2	0
r	rw	rw	rw	r	rw	rw	r

31 RESVD — RESERVED

30 CICHECK — It enables the verification of the configuration interface integrity before each scrubbing run.
0 = Disabled;
1 = Enabled.

29: 22 BLKFRAME — Maximum number of frames to be scrubbed at time (i.e., reading sequentially during readback, or writing sequentially during blind scrubbing).
If BLKFRAME > 0, the frames are scrubbed by blocks, and a maximum of *BLKFRAME* frames is scrubbed each time. Thus, the number of frames defined in the FCNT bitfield of the Frame Configuration Register (FCR) is split into the *BLKFRAME* size. This feature is used to have more control of the FPGA configuration interface, and to avoid catastrophic results when the interface is affected by soft-errors.
If BLKFRAME = 0, all frames set in the FCR are scrubbed sequentially at once.
Default value = 1 (only one frame is scrubbed at time).

21 BITSWPEN — FPGA bit swapping enable: 0 = disable; 1 = enable. Enabled by default

20 CISEL — Configuration interface selected (read only):
0 = SelectMap;
1 = Not used.

19: 12 TPROG — Number of clock cycles to wait with PROGRAM_B signal asserted for FPGA programming (PROGRAM_B pulse width). Check the target FPGA Data Sheet for the TPROGRAM pulse width. One microsecond is safe for most Xilinx FPGAs.
TPROG >= 1 us / smapclki_period
Default value = 150 (wait 150 clock cycles).

11: 4 ROWBND — Row boundary alignment: number of FPGA frames in the rows boundaries.
Default value = 2 (two frames in the rows boundaries).
It must be zero if there is no row boundaries in the FPGA.

3: 0 RESVD — RESERVED

72.13.16Capability Register (CAP)

*Table 1149.*0x3C - CAP - Capability register

31 23	22 20	19 18	17 16	15 4	3 0
RESVD	DATACK	SMBUS	CINT	FAMILY	RESVD
0	0	0	0	0	0
r	r	r	r	r	r

31: 23	RESVD	RESERVED

22: 20	DATACK	Supported data checks (read only):
		bit#20 = FFC;
		bit#21 = CRC;
		bit#22 = Not used.

19: 18	SMBUS	SelectMap bus wide in bits (read only): 00 = 8-bit wide; 01 = 16-bit wide; 10 = 32-bit wide; 11 = Not used.
17: 16	CINT	Supported configuration interfaces (read only): bit#16 = SelectMap others = Not used.

15: 4	FAMILY	Supported FPGA families (read only):
		bit#4 = Xilinx Virtex-5;
		bit#5 = Xilinx Kintex UltraScale;
		others = Not used.

3: 0	RESVD	RESERVED

The register value reflects the configuration set in the generics.

72.13.17Frame Id Code Register (FRAMEID)

*Table 1150.*0x40 - FRAMEID - Frame id code register

31 16	15 0
RESVD	FRMID
0	0
r	rw

31: 16	RESVD	RESERVED
15: 0	FRMID	Frame id of the current scrubbed frame. It is updated at every new scrubbed frame. Frame id is also updated during mapping phase.

The frame id represents the identification of the frame, which is defined from 0 to FCNT-1. Thus, FRMID 0 means the first frame, whereas the FRMID FCNT-1 means the last one.

In case of READBACK_ERROR, FRAME_MAPPING_ERROR, or WRITE_FRAME_ERROR errors, the FRMID represents the last frame processed by the GRSCRUB. In these cases, the reported FRMID might not be precise. The GRSCRUB has an internal FIFO to receive the amount of data from the FPGA, and the current frame id takes into account only the processed data.

The register is only cleared at the system reset.

72.13.18Error Frame Id Code Register (ERRFRMID)

*Table 1151.*0x44 - ERRFRMID - Error frame id code register

31 16	15 0
UE_FRMID	ERR_FRMID
0	0
rw	rw

31: 16 UE_FRMID — Frame id of the last frame with uncorrectable error.
It is only updated if a new uncorrectable error occur.
This bitfield is only valid if the UECNT from the Number of Errors Detected Register (ECNT) is higher then 0.

15: 0 ERR_FRMID — Frame id of the last frame detected with error during readback.
It is only updated if a new frame with error is detected.

The frame id represents the identification of the frame, which is defined from 0 to FCNT-1. Thus, FRMID 0 means the first frame, whereas the FRMID FCNT-1 means the last one.

The register is only cleared at the system reset.

72.14 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x0C1. For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

72.15 Implementation

72.15.1 Reset

The GRSCRUB changes its reset behavior depending on settings in the GRLIB configuration package (see GRLIB User's Manual). GRSCRUB features an asynchronous reset for the registers if the parameter grlib_async_reset_enable is set. Otherwise, the GRSCRUB implements a synchronous reset. The reset is applied to all registers in GRSCRUB.

The asynchronous reset of the SelectMap IO registers is controlled by ioreg_arst in the GRSCRUB configuration options (section 72.17). If the ioreg_arst is defined as 0, the asynchronous reset is disabled. To enable the asynchronous reset of the SelectMap IO registers, both ioreg_arst and grlib_async_reset_enable must be set. If the asynchronous reset of the SelectMap IO registers is enabled, logic might be inferred between the IOs and flip-flops during implementation in an FPGA tool.

Two reset signals are defined in the GRSCRUB, one for each clock domain. The RSTN is the reset signal considering the CLK clock domain, which is the system clock. The SMAPRSTN is used to reset the internal SelectMap control module and IO registers, and it is related to the SMAPCLKI clock domain.

The reset values of the GRSCRUB signals are defined in section 72.19.

72.15.2 Endianness

The core support both big-endian and little-endian systems. The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User’s Manual). The endianness is configured through the AMBA AHB bus, which is used to set the generic bus master interface to access the external memory.

72.15.3 SelectMap timing parameters

All pins of the slave SelectMap interface on the FPGA are synchronous with the input clock (CCLK), except PROGRAM_B that is an asynchronous reset. Refer to the Xilinx FPGA Configuration Guide to a description of the SelectMap pins and their functionalities.

The CCLK frequency (Fcclk) is not necessarily the same as the system (microcontroller or FPGA design) clock (Fsys_clk). System and SelectMap interface can run in different frequency domains, following the relation Fsys_clk >= Fcclk. See the FPGA Data Sheet to details about the maximum slave SelectMap CCLK frequency. Also, the timing limitation due to the interface connection between the SelectMap and GRSCRUB should be considered to define Fcclk.

Figures 176 and 177 present the timing characteristics of the SelectMap pins related to CCLK for data loading and reading, and table 1152 describes the timing parameters. For timing information of the FPGA SelectMap interface, refer to the FPGA Data Sheet.

Note: The timing parameters are technology dependent.

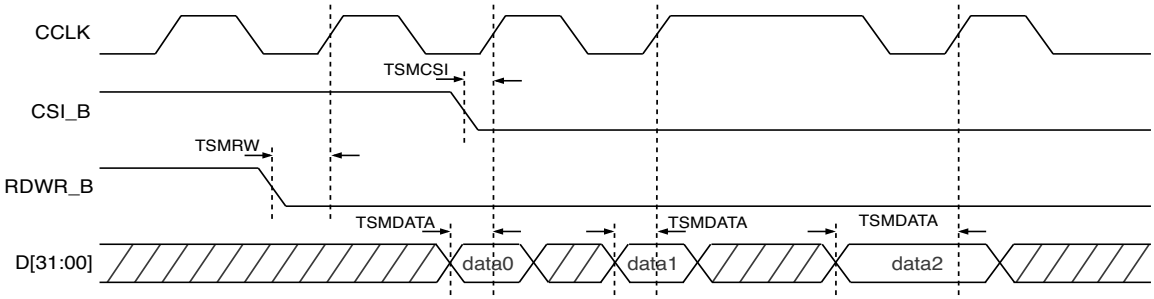


Figure 176. SelectMap timing characteristics for data loading to FPGA

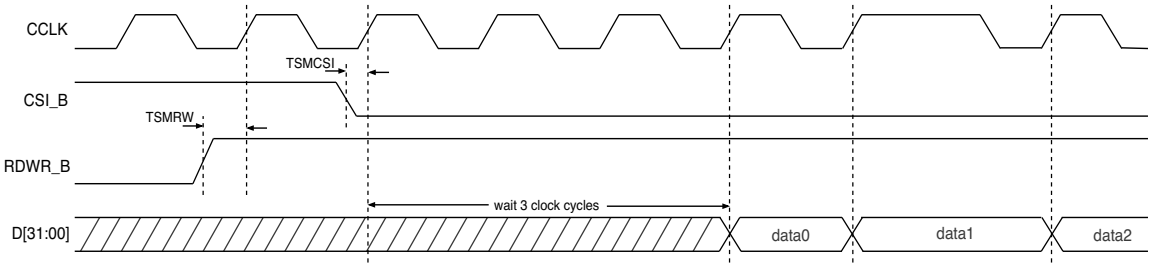


Figure 177. SelectMap timing characteristics for data reading from FPGA

Table 1152. SelectMap timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
TSMDATA	D[31:00] setup/hold	Rising clock edge	TBD	-	ns
TSMCSI	CSI_B setup/hold	Rising clock edge	TBD	-	ns
TSMRW	RDWR_B setup/hold	Rising clock edge	TBD		ns
TSMCO	D[31:00] clock to out in readback	Rising clock edge	-	TBD	ns

72.15.4 SelectMap clock enable

The SelectMap clock CCLK is controlled (enabled/disabled) by the SMAPO.clk_en signal from the GRSCRUB. The GRSCRUB main clock (CLK signal) is the system clock. The system must also provide the SelectMap clock to the GRSCRUB (SMAPCLKI signal) and the CCLK to the FPGA. In addition, the system design must contain a dedicated register buffer to allow enabling/disabling the CCLK based on SMAPO.clk_en.

72.15.5 Scrubbing run period

The period to execute one scrubbing run, which means the time to scrub all frames configured in the registers once, depends on the characteristics of the system presented below. Blind and readback scrubbing modes are affected by the following factors:

- Number of frames to be scrubbed, which is defined in the FCNT bitfield of Frame Configuration Register (FCR);
- Number of words in each frame, which is defined in the FLEN bitfield of Frame Configuration Register (FCR);
- The data bus width (8, 16, or 32-bit wide) for the SelectMap interface defined in the smap_buswide of the GRSCRUB configuration options (section 72.17).
- The target Xilinx FPGA family;
- The GRSCRUB clock frequencies: CLK and SMAPCLKI signals;
- The required time to access the Golden memory.

Additional factors affect the readback scrubbing, as following:

- The selected error detection type, which can be FFC and/or CRC defined in the Configuration Mode Register (CONFIG);
- If error correction is enabled or not, which is defined in the CORM bit of the Configuration Mode Register (CONFIG);
- The number of errors detected in the scrubbing run, which is presented in the RECNT bitfield of the Number of Errors Detected Register (ECNT) (if correction is enabled).

Table 1153 generically describes the time parameters that affects the scrubbing period.

Table 1153. Generic timing description

Name	Description
SCRUBP	Scrubbing period
TFRD	Time to read one frame from the FPGA
TFWR	Time to write one frame to the FPGA
TD	Time to detect error in a frame
TC	Time to correct one frame

Considering the readback scrubbing mode with only detection enabled, the scrubbing period would be:

$$SCRUBP = [time\ to\ read\ frame\ and\ detect\ error] * [number\ of\ frames]$$

$$SCRUBP = (TFRD + TD)*FCNT$$

Considering the readback scrubbing mode with detection and correction enabled, the scrubbing period would be:

$$SCRUBP = [detection\ time] + [correct\ erroneous\ frames] + [recheck\ corrected\ frames]$$

$$SCRUBP = (TFRD + TD)*FCNT + (TC + TFWR)*RECNT + (TFRD + TD)*RECNT$$

72.16 Electrical characteristics and requirements

Since the GRSCRUB signals are directly connected to the SelectMap interface of the target FPGA, one must carefully observe the required voltage levels of the FPGA I/O bank and the I/Os of the system with the GRSCRUB.

72.17 Configuration options

Table 1154 shows the configuration options of the GRSCRUB (VHDL generics).

Table 1154. Configuration options

Generic name	Function	Allowed range
hindex	AMBA AHB master index	0 - NAHBMST-1
pindex	AMBA APB slave index	0 - NAPBSLV-1
paddr	Address field of the APB bar	0 - 16#FFF#
pmask	Mask field of the APB bar	0 - 16#FFF#
pirq	Interrupt line used by the GRSCRUB	0 - NAHBIRQ-1
cfg_tech	Implementation technology	0 - NTECH
ffc_en	Enable the FFC capability (0) disable, (1) enable	0 - 1
crc_en	Enable the CRC capability (0) disable, (1) enable	0 - 1
smap_buswide	SelectMap data bus width [1:0] (00) 8-bit wide (01) 16-bit wide (10) 32-bit wide (11) Not used	"00", "01", "10"
smap_sync_delay	Number of clock cycles delay for SelectMAP input data synchronization. No delay is set as default. A delay might be required due to setup configuration.	0 - 15
config_interface	Configuration interface. Currently, only SelectMap is supported [1:0]. bit#0 SelectMap: (0) disable, (1) enable	"00", "01"
syncfifo_abits	Internal fifo address bits	0 - 16#FFFF#
syncfifo_ft	Internal fifo fault tolerance enable (only applicable for GRLIB FT)	0 - 1
fpga_timeout	Timeout in clock cycles to wait for FPGA response	0 - 16#FFFFFFFF#
ioreg_arst	Enable the asynchronous reset of SelectMap IO registers (0) disable, (1) enable	0 - 1

72.18 Signal descriptions

Table 1155 shows the interface signals of the GRSCRUB (VHDL ports).

Table 1155. Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset signal considering the CLK clock domain.	Low
CLK	N/A	Input	System clock.	-
SMAPRSTN	N/A		Reset signal for the internal SelectMap module, considering the SMAPCLKI clock domain.	Low
SMAPCLKI	N/A	Input	SelectMap clock. This clock is used by the internal module directly connect to the SelectMap interface.	-
TICK_IN	N/A	Input	Tick signal used as reference to apply delay in periodic scrubbing runs. The Delay Register (DELAY) presents the number of ticks that the GRSCRUB must wait during the holding time. The TICK_IN signal must be asserted for one system clock cycle to be counted.	High
AHBMI	*	Input	AMB master input signals	-

*Table 1155.*Signal descriptions

Signal name	Field	Type	Function	Active
AHBMO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
SMAPI	initn	Input	FPGA initialization signal	Low
	done	Input	FPGA done signal	High
	idata[31:0]	Input	Data read from the FPGA	
SMAPO	progn	Output	Clean FPGA configuration logic	Low
	csin	Output	SelectMap chip select	Low
	rdwr	Output	Read (1) and write (0) select	-
	odata[31:0]	Output	Data to be loaded to the FPGA	-
	clk_en	Output	SelectMap clock enable	High
	msel[2:0]	Output	Enables the SelectMap interface (only required if SelectMap is not electrically set, otherwise can be left unconnected)	"110"

* see GRLIB IP Library User's Manual

72.19 Signal definitions and reset values

The GRSCRUB signals and their reset values are described in table 1156.

*Table 1156.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
initn	Input	FPGA initialization signal	Low	-
done	Input	FPGA done signal	High	-
idata[31:0]	Input	Data read from the FPGA	-	-
progn	Output	Clean FPGA configuration logic	Low	Logical 1
csin	Output	SelectMap chip select	Low	Logical 1
rdwr	Output	Read (1) and write (0) select	-	Logical 1
odata[31:0]	Output	Data to be loaded to the FPGA	-	Logical 0
clk_en	Output	SelectMap clock enable	High	Logical 1
msel[2:0]	Output	Enable the SelectMap interface (only required if SelectMap is not electrically set, otherwise can be left unconnected)	"110"	Logical 0

72.20 Library dependencies

Table 1157 shows the libraries used when instantiating the GRSCRUB (VHDL libraries).

Table 1157. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GRLIB	GENERIC BM PKG	Signals, components	Package including the generic bus master interface and the bridges for converting to AHB and AXI
GAISLER	GRSCRUB	Signals, components	GRSCRUB component and signal declarations
TECHMAP	GENCOMP	Components	Technology-dependent components: Syncronous 2-port FIFO with tech selection (syncfifo_2p)

72.21 Instantiation

This example shows how the GRSCRUB can be instantiated using the AMBA 2.0 AHB wrapper.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.grscrub_pkg.all;

entity grscrub_example is
generic (
     padtech: integer := 0
);
port (
     resetn : in std_ulogic;
     clock : in std_ulogic;
     -- SelectMap interface signals
     smap_data : inout std_logic_vector(31 downto 0);
     smap_done : in std_logic;
     smap_init_b : in std_logic;
     smap_cclk : out std_logic;
     smap_prog_b : out std_logic;
     smap_csi_b : out std_logic;
     smap_rdwr_b : out std_logic
);
end;

architecture rtl of grscrub_example is

-- reset and clock signals
signal rstn : std_ulogic;
signal clk : std_ulogic;
signal smclki : std_ulogic;

-- AMBA signals
signal apbi : apb_slv_in_type;
signal apbo : apb_slv_out_vector := (others => apb_none);
signal ahbmi : ahb_mst_in_type;
signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

-- GRSCRUB signals
signal smapi : smap_in_type;
signal smapo : smap_out_type;
signal smap_oen : std_logic_vector(31 downto 0);

begin
-- clock and reset are generated here
...

-- AMBA components are instantiated here
...

grscrub0 : grscrub_ahb
generic map(
     hindex => 1,
     pindex => 10,
     paddr => 10,
     pmask => 16#FFF#,
     pirq => 1,
     cfg_tech => CFG_FABTECH,
     ffc_en => '1',
     crc_en => '1',
     smap_buswide => "00",
     config_interface => "01",
     syncfifo_abits => 7,
     syncfifo_ft => 0,
     fpga_timeout => X"FFFFFFFF")
port map(
     clk => clk,
```

```
    rstn => rstn,
    tick_in => tick_pulse,
    ahbmi => ahbmi,
    ahbmo => ahbmo(0),
    apbi => apbi,
    apbo => apbo(0),
    smapclki => smclki,
    smapi => smapi,
    smapo => smapo
);

-- example of clock buffer instantiation for SelectMap clock based on the enable signal
clk_buf : ODDR
port map (
    Q => smap_cclk, -- 1-bit output
    C => smclki, -- 1-bit clock input
    CE => smapo.clk_en, -- 1-bit clock enable input
    D1 => '1', -- 1-bit data input (positive edge)
    D2 => '0', -- 1-bit data input (negative edge)
    R => '0', -- 1-bit reset input
    S => '0' -- 1-bit set input
);

-- generate tick pulse
tick_pulse ...

-- example of data pad instantiation for SelectMap data signals
smap_oen <= (others => smapo.rdwr);
dat : iopadvv
generic map (tech => padtech, width => 32, oepol => 0)
port map (smap_data(31 downto 0), smapo.odata(31 downto 0), smap_oen(31 downto 0),
smapi.idata(31 downto 0));

-- other signals association
-- inputs
smapi.done <= smap_done;
smapi.initn <= smap_init_b;
-- outputs
smap_rdwr_b <= smapo.rdwr;
smap_prog_b <= smapo.progn;
smap_csi_b <= smapo.csin;

end architecture ;
```

73 GRSHYLOC - Wrapper for the SHYLOC compressor with DMA engine

73.1 Overview

GRSHYLOC implements a wrapper for the SHYLOC compressor, enabling two possible configurations by means of a generic. Both configurations have common control and data interfaces. Additionally, GRSHYLOC features a DMA engine to fetch the raw image and store the compressed bitstream from/to external memory through the AMBA AHB master interface.

The compressed samples are temporary stored into a local SRAM. This reduces the risk of missing output data in case of delays accessing the AHB bus. The content of the memory is written to the external memory when the DMA controller is not busy fetching raw data for the SHYLOC core.

GRSHYLOC also includes an AMBA AHB slave interface for configuration and control. There are three sets of registers: one for configuring the CCSDS123 IP, another for the CCSDS121 encoder (optional) and a global set for controlling the overall compression process.

The block diagrams below depict the two possible configurations for GRSHYLOC. The first one involves the complete SHYLOC system: the CCSDS123 IP performs the prediction stage, whereas CCSDS121 implements the block-adaptive coding. On the other hand, the second configuration only instantiates the CCSDS123, which performs both as predictor and as sample-adaptive encoder.

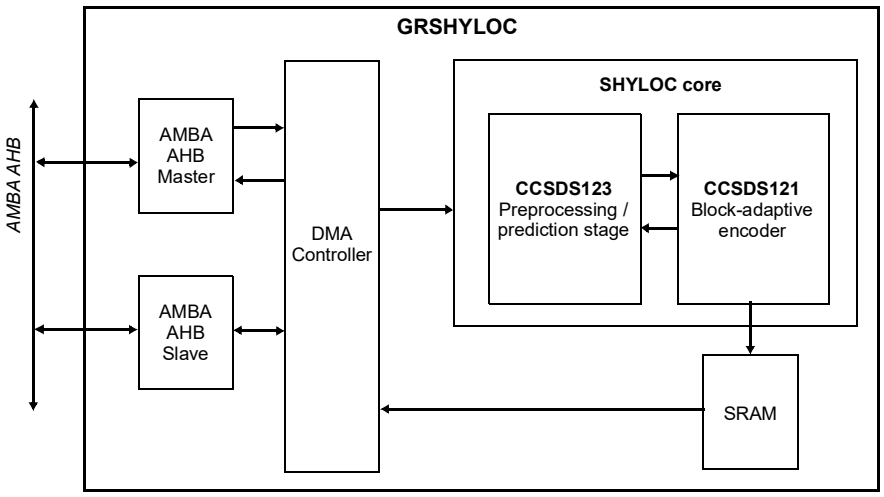


Figure 178. GRSHYLOC block diagram when implementing the SHYLOC compressor

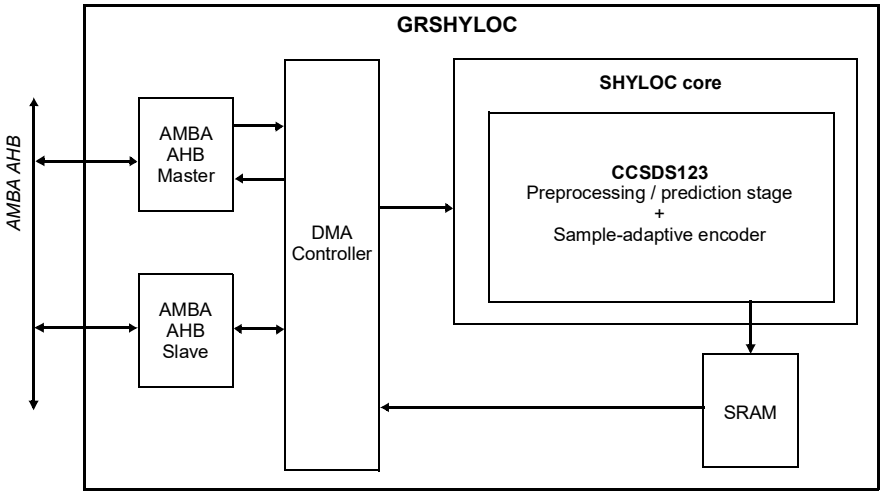


Figure 179. GRSHYLOC block diagram when implementing the standalone CCSDS123

73.2 Compression IPs description

The image compression is performed by two different IPs that may operate standalone or combined to form the so-called SHYLOC system. The following sections briefly describe the purpose of both IPs. For more detailed information please refer to the SHYLOC datasheet, also available in GRLIB.

73.2.1 CCSDS123

The CCSDS123 IP core performs lossless compression for multi-spectral and hyper-spectral images according to the CCSDS123 standard (Lossless Multispectral & Hyperspectral Image Compression. Recommendation for Space Data System Standards, CCSDS 123.0-B-1. Blue Book. Issue 1).

The first stage of the compression is the predictor. The IP is able to compress samples that are received in BSQ, BIL and BIP order. The predictor architecture shall be selected at implementation time by means of a generic, so it is not configurable at run-time. Some of the architectures require external memory for storing intermediate data. A dedicated AHB master interface is used for this purpose.

The second stage of the compression is the encoder. This step can be achieved using either the CCSDS123 or the CCSDS121 IP. For the latter, more information can be found in the next section. If the CCSDS123 is used as encoder, it performs sample-adaptive coding.

The compression process is configured by means of registers mapped into AHB address space. The IP shall receive the configuration prior to starting the compression of the raw data.

73.2.2 CCSDS121

The CCSDS121 IP core is a lossless universal compressor based on Rice adaptive coding, as per the CCSDS121 standard (Lossless Data Compression. Recommendation for Space Data System Standards, CCSDS 121.0-B-2. Blue Book. Issue 2).

The IP may be used in conjunction with the CCSDS123 IP core, forming the SHYLOC system. In this case, the CCSDS123 is solely in charge of the prediction, thus producing the mapped prediction residuals. These residuals are the inputs for the CCSDS121 IP, which performs the block-adaptive coding of the data. The output of the IP is the compressed bitstream.

The compression process is configured by means of registers mapped into AHB address space. The IP shall receive the configuration prior to starting the compression of the raw data.

73.2.3 SHYLOC considerations

When the SHYLOC system is used, both CCSDS121 and CCSDS123 must be separately configured prior to starting any compression. Some compatibility rules must be followed in this case:

- Same reset behavior for both IPs (asynchronous or synchronous).
- Same configured image size for both IPs (Nx, Ny, Nz).
- Same input dynamic range for both IPs (D).
- CCSDS121 shall be configured to be big-endian.

73.3 Operation

This section describes the operation of GRSHYLOC when both CCSDS121 and CCSDS123 IPs are instantiated. The same steps shall be followed if the CCSDS123 is working as a standalone IP, with the exception that the CCSDS121 shall not be configured via the AMBA AHB slave interface, as it is not present.

Before starting a compression, both compression IPs shall be configured through the AHB slave interface. All the registers must be written to, even if the default or the previous values are to be kept. CCSDS121 and CCSDS123 shall be configured separately following the rules as indicated in 73.2.3.

This mainly involves the characteristics of the image, the encoding type and the parameters for the predictor and encoders. Once a valid configuration has been set, the compression core indicates that is no longer waiting for a new configuration, and is ready to receive the first samples.

With the CCSDS121 and CCSDS123 fully configured, the user may need to set the initial address of both the input raw data and the output compressed bitstream, so that the DMA engine can start fetching/storing valid data.

Finally, the compression may start by setting the Start bit in the GRSHYLOC control register to 1b. The samples will be sequentially fetched by the DMA controller and passed to the compressor when ready. The compressed outputs will be internally stored into a local SRAM before being written to the AHB memory.

Optionally, the compression may be aborted by setting the ForceStop bit in the GRSHYLOC control register to 1b. This immediately stops the DMA fetching/storage, as well as the image compression.

The compression may finish successfully or with errors, or it may have been aborted as explained above. In any case, once the compression ends, the user shall clear both Start and ForceStop bits to 0b (GRSHYLOC control register) before starting the next compression.

The overall status of the compression can be obtained by reading the GRSHYLOC Status Register. Two bits determine whether the compression is ongoing or has finished, and the size of the output bitstream is also made available. The content of this register is valid until the user ends the compression by clearing the GRSHYLOC control register. Please note that if the Finished bit is set to 1b, the core has completed both the image compression and the storage into the AHB memory.

For more in-depth status information, the respective Control/Status Register for each compression core may also be checked. These registers tell whether the internal cores are awaiting a new configuration or are ready to receive new samples, as well as if there has been any compression errors.

73.4 Registers

The core is programmed through registers mapped into AHB address space. Only 32-bit single accesses to the registers are supported.

For more information regarding the compression parameters, please refer to the SHYLOC Datasheet.

Table 1158. GRSHYLOC registers

AHB address offset	Register
0x000	GRSHYLOC Capability Register
0x004	GRSHYLOC Status Register
0x008	GRSHYLOC Control Register
0x00C	GRSHYLOC Input Address Register
0x010	GRSHYLOC Output Address Register
0x100	CCSDS123 Control/Status Register
0x104	CCSDS123 External Memory Address Register
0x108	CCSDS123 Configuration Register 0
0x10C	CCSDS123 Configuration Register 1
0x110	CCSDS123 Configuration Register 2
0x114	CCSDS123 Configuration Register 3
0x200	CCSDS121 Control/Status Register
0x204	CCSDS121 Configuration Register 0
0x208	CCSDS121 Configuration Register 1
0x20C	CCSDS121 Configuration Register 2

73.4.1 GRSHYLOC Capability Register

Table 1159. 0x000 - CAP - GRSHYLOC Capability Register

31 … 3	2 … 1	0
Reserved	PredSel	Enc-Sel
All 0's	0	0
r	r	r

2-1: PredSel — Architecture selected for the predictor. BIP-Base (00b), BIP-Mem (01b), BSQ (10b) or BIL (11b).
0: EncSel — Encoder selected:
0b: SHYLOC system. CCSDS121 acts as a block-adaptive encoder.
1b: Standalone CCSDS123. It implements a sample-adaptive encoder.

Note: if the BIP-Mem or the BSQ architectures are selected for the predictor, the compression core makes use of the external memory for intermediate calculation.

73.4.2 GRSHYLOC Status Register

Table 1160. 0x004 - STAT - GRSHYLOC Status Register

31 … 16
NW
0x0000
r

15 … 2	1	0
Reserved	FIN	ON
All 0's	0	0
r	r	r

31-16: NW — Number of words (32 bits) stored into the external AHB memory for the current compression. When FIN is 1b (compression finished), it shall be interpreted as the size of the output bitstream.
1: FIN — Compression finished. Only set to 1b if the compression is successful (i.e. not if interrupted).
0: ON — Compression ongoing.

Note: NW, FIN and ON are cleared when the user finishes the compression by clearing both ForceStop and Start bits in the GRSHYLOC Control Register.

73.4.3 GRSHYLOC Control Register

Table 1161. 0x008 - CTRL - GRSHYLOC Control Register

31 … 3	2	1	0
Reserved	FS	ST	RST
All 0's	0	0	0
r	rw	rw	rw

2: FS — ForceStop. It stops immediately both the compression and the DMA activity.
1: ST — Start. If the compression core is ready, it starts fetching the raw data from memory.
0: RST — Reset. It performs a complete software reset of the core when set to 1b. Self-clearing.

Note: when the compression finishes, the user shall set both FS and ST to 0b before starting a new compression. This effectively clears the content of the GRSHYLOC Status Register.

73.4.4 GRSHYLOC Input Address Register

Table 1162. 0x00C - IADDR - GRSHYLOC Input Address Register

31	2	1	0
InAddr		Reserved	
0x00000000		00	
rw		r	

31-0: InAddr Initial address of the input raw image. It is word-aligned, therefore the 2 LSB are set to 0b.

73.4.5 GRSHYLOC Output Address Register

Table 1163. 0x010 - OADDR - GRSHYLOC Output Address Register

31	2	1	0
OutAddr		Reserved	
0x00000000		00	
rw		r	

31-0: OutAddr Initial address of the output compressed image. It is word-aligned, therefore the 2 LSB are set to 0b.

73.4.6 CCSDS123 Control/Status Register

Table 1164. 0x100 - 123CS - CCSDS123 Control/Status Register

31	30	29	28	27	26	25	24	23	22	21	16
AW	RD	FF	EOP	FIN	ERR	EC				Reserved	
0	0	0	0	0	0	0000				000000	
r	r	r	r	r	r	r				r	

15	1	0
Reserved		EN
All 0's		0
r		rw

31: AW CCSDS123 IP waiting for a new configuration.
30: RD CCSDS123 IP ready to receive new samples.
29: FF FIFO Full. Attempted to write to a full input FIFO.
28: EOP End of Packet. The CCSDS123 IP is processing the last sample of an image.
27: FIN Compression finished.
26: ERR Compression error.
25-22: EC Error Code:
0x0: No error.
0x1: Nx, Ny or Nz are set to 0.
0x2: Configured values are not within the range defined by the hard-coded constants.
0x3: W_BUFFER < D + U_MAX_GEN.
0x4: W_BUFFER_GEN < D_GEN + U_MAX_GEN.
0x5: Invalid FSM state.
0x6: AHB error.
0: EN Enable.
1b: Enable the CCSDS123 IP. The core must have received a valid configuration.
0b: Disable the CCSDS123 IP.

73.4.7 CCSDS123 External Memory Address Register

Table 1165. 0x104 - 123ADDR - CCSDS123 External Memory Address Register

31	0
ExtAddr	
0x00000000	
rw	

31-0: ExtAddr External Memory Address. Used by two predictor architectures (BIP-Mem and BSQ).

73.4.8 CCSDS123 Configuration Register 0

Table 1166. 0x108 - 123CFG0 - CCSDS123 Configuration Register 0

31	16
Nx	
0x0000	
rw	

15 11	10	9	8 7	6 3	2	1 0
D	SG	DH	ENC	P	BYP	Reserved
00000	0	0	0	0000	0	00
rw	rw	rw	rw	rw	rw	r

31-16: Nx Number of samples in a line.
15-11: D Dynamic range of the input samples.
10: SG Is_Signed bit: input samples are unsigned (0b) or signed (1b).
9: DH Disable Header: enable (0b) or disable (1b) the generation of headers.
8-7: ENC Encoder Selection:
00b: Disable encoder.
01b: Select sample-adaptive encoder (CCSDS123).
10b: Select external block-adaptive encoder (CCSDS121).
6-3: P Number of bands used for prediction.
2: BYP Perform normal compression if 0b. Bypass compression if 1b.

73.4.9 CCSDS123 Configuration Register 1

Table 1167. 0x10C - 123CFG1 - CCSDS123 Configuration Register 1

31	16
Ny	
0x0000	
rw	

15	14	13 9	8 2	1 0
PR	LS	OMEGA	R	Reserved
0	0	00000	0000000	00
rw	rw	rw	rw	r

31-16: Ny Number of samples in a row.
15: PR Prediction: full (0b) or reduced (1b) prediction.
14: LS Local Sum: neighbour- (0b) or column- (1b) oriented local sum.
13-9: OMEGA Weight component resolution.
8-2: R Register size.

73.4.10 CCSDS123 Configuration Register 2

Table 1168. 0x110 - 123CFG2 - CCSDS123 Configuration Register 2

31	16
Nz	
0x0000	
rw	

15	11	10	6	5	2	1	0
VMAX		VMIN		T_INC		WI	ED
00000		00000		0000		0	0
rw		rw		rw		rw	rw

31-16: Nz — Number of bands.
15-11: VMAX — Factor for weight update.
10-6: VMIN — Factor for weight update.
5-2: T_INC — Weight update factor change interval.
1: WI — Weight Initialization Mode. Only 0b is supported.
0: ED — Endianness: little-endian (0b) or big-endian (1b).

73.4.11 CCSDS123 Configuration Register 3

Table 1169. 0x114 - 123CFG3 - CCSDS123 Configuration Register 3

31	28	27	26	23	22	19	18	16
ICE		AIT	AIC		RCS		U_MAX	
0000		0	0000		0000		000	
rw		rw	rw		rw		rw	

15	13	12	6	5	0
U_MAX		W_BUF		Reserved	
000		0000000		000000	
rw		rw		r	

31-28: ICE — Initial Count Exponent.
27: AIT — Accumulator Initialization Type:
0b: Accumulator Initialization Table shall not be used.
1b: Accumulator Initialization Table shall be read from the generic values.
26-23: AIC — Accumulator Initialization Constant.
22-19: RCS — Rescaling Counter Size.
18-13: U_MAX — Unary Length Limit.
12-6: W_BUF — Bit width of the output buffer.

73.4.12 CCSDS121 Control/Status Register

Table 1170. 0x200 - 121CS - CCSDS121 Control/Status Register

31	30	29	28	27	26	25	24	23	22	21	16
AW	RD	FF	EOP	FIN	ERR	EC				Reserved	
0	0	0	0	0	0	0000				000000	
r	r	r	r	r	r	r				r	

15	1	0
Reserved		EN
All 0's		0
r		rw

31: AW — CCSDS121 IP waiting for a new configuration.
30: RD — CCSDS121 IP ready to receive new samples.
29: FF — FIFO Full. Attempted to write to a full input FIFO.
28: EOP — End of Packet. The CCSDS121 IP is processing the last sample of an image.
27: FIN — Compression finished.
26: ERR — Compression error.
25-22: EC — Error Code:
0x0: No error.
0x1: Nx, Ny or Nz are set to 0.
0x2: Configured values are not within the range defined by the hard-coded constants.
0x3: W_BUFFER < D.
0x4: W_BUFFER_GEN < D_GEN.
0x5: Invalid FSM state.
0: EN — Enable.
1b: Enable the CCSDS121 IP. The core must have received a valid configuration.
0b: Disable the CCSDS121 IP.

73.4.13 CCSDS121 Configuration Register 0

Table 1171. 0x204 - 121CFG0 - CCSDS121 Configuration Register 0

31	16
Nx	
0x0000	
rw	

15	14	13	7	6	0
CS	DH	J		W_BUF	
0	0	0000000		0000000	
rw	rw	rw		rw	

31-16: Nx — Number of samples in a line.
15: CS — Codeset. Code option.
14: DH — Disable Header: it selects whether to send the header or not.
13-7: J — Block Size. Possible values: 0, 8, 16, 32, 64.
6-0: W_BUF — Bit width of the output buffer.

73.4.14 CCSDS121 Configuration Register 1

*Table 1172.*0x208 - 121CFG1 - CCSDS121 Configuration Register 1

31	16
Ny	
0x0000	
rw	

15	3	2	0
REF		Reserved	
All 0's		000	
rw		r	

31-16: Ny Number of samples in a row.
15-3: REF Reference Sample Interval.

73.4.15 CCSDS121 Configuration Register 2

*Table 1173.*C0x20C - 121CFG2 - CSDS121 Configuration Register 2

31	16
Nz	
0x0000	
rw	

15	10	9	8	7	6	5	4	0
D		Res	ED	PRE		BYP	Reserved	
000000		0	0	000		0	00000	
rw		r	rw	rw		rw	r	

31-16: Nz Number of bands.
15-10: D Dynamic range of the input samples.
8: ED Endianness: little-endian (0b) or big-endian (1b).
7-6: PRE Preprocessor selection:
00b: no preprocessor is present.
01b: CCSDS123 preprocessor is present.
10b: any other preprocessor is present.
5: BYP Perform normal compression if 0b. Bypass compression if 1b.

73.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x0B7 (GRSHYLOC). For description of vendor and device identifiers see GRLIB IP Library User's Manual.

73.6 Implementation

73.6.1 Reset

The core changes its reset behavior depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

GRSHYLOC will feature a synchronous reset if the parameter *grlib_async_reset_enable* is not set in the GRLIB configuration package. On the contrary, it will implement an asynchronous reset if *grlib_async_reset_enable* is set.

73.6.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual).

73.7 Configuration options

Table 1174 shows the configuration options of the core (VHDL generics).

Table 1174. Configuration options

Generic name	Function	Allowed range	Default
tech	Technology for the syncram and FIFOs	0 - NTECH	0 (inferred)
hmindex_dma	AHB master index for the DMA controller master i/f	0 - NAHBMST-1	0
hmindex_core	AHB master index for the CCSDS123 master i/f	0 - NAHBMST-1	1
hsindex	Common AHB slave index	0 - NAHBSLV-1	0
haddr	Addr field of the AHB bar	0 - 16#FFF#	0
hmask	Mask field of the AHB bar	0 - 16#FFF#	16#FFC#
encdsel	Encoder selector: SHYLOC (0) or standalone CCSDS123 (1)	0 - 1	0
predsel	Predictor architecture selector: BIP-Base (0), BIP-Mem (1), BSQ (2) or BIL (3)	0 - 3	0

73.8 Signal descriptions

Table 1175 shows the interface signals of the core (VHDL ports).

Table 1175. Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AMB master input signals	-
AHBMO_DMA	*	Output	AHB master output signals (DMA controller)	-
AHBMO_CORE	*	Output	AHB master output signals (CCSDS123 core)	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-

* see GRLIB IP Library User's Manual

73.9 Library dependencies

Table 1176 shows the libraries used when instantiating the core (VHDL libraries).

Table 1176. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
SHYLOC		Components	SHYLOC components

73.10 Instantiation

This example shows how the core can be instantiated.

```
library  ieee;
use      ieee.std_logic_1164.all;
library  grlib;
use      grlib.amba.all;
library  shyloc;

entity example is
   generic (
      tech : integer := 0
      );
   port (
      clk  : in std_ulogic;
      rstn : in std_ulogic;
      -- Other signals
      ...
      );
end entity example;

architecture rtl of example is

  -- AMBA bus
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);

...

  -- Component instantiation
  grshyloc_dut : entity shyloc.grshyloc
    generic map (
      tech         => tech,
      hmindex_dma  => 1,
      hmindex_core => 2,
      hsindex      => 1,
      haddr        => 16#000#,
      hmask        => 16#FFC#,
      encdsel      => 0,
      predsel      => 2)
    port map (
      clk        => clk,
      rstn       => rstn,
      ahbmi      => ahbmi,
      ahbmo_dma  => ahbmo(1),
      ahbmo_core => ahbmo(2),
      ahbsi      => ahbsi,
      ahbso      => ahbso(1)
      );

...

end architecture rtl;
```

74 GRSPFI - SpaceFibre codec with DMA, RMAP target and Bus Master Interface

74.1 Overview

GRSPFI provides an interface between the system bus (such as AHB or AXI) and a SpaceFibre network. It features a single-lane implementation of a SpaceFibre node designed in accordance to the SpaceFibre specification ECSS-E-ST-50-11C, as well as a flexible DMA engine with a programmable number of independent DMA channels (from 1 to 8). Each channel has its own bus master interface for optimizing the throughput of the IP. There are wrappers available adapting the generic bus master interfaces to either AMBA AHB 2.0 or AXI4.

The core is configured through a set of registers accessed through an AHB slave interface. The SpaceFibre packets are autonomously handled by the DMA channels, both in transmission and reception. Each DMA channel is assigned a sub-set (or all) of the Virtual Channels and/or the Broadcast Channel; this assignment is programmed in run-time via the AHB registers.

An optional RMAP target adhering to the ECSS-E-ST-50-52C standard can be included to handle RMAP commands autonomously without software intervention. If present, every DMA channel features its own hardware RMAP target, which can be enabled and configured separately.

GRSPFI requires an external SerDes for the connection to the SpaceFibre network, either on the chip or external to the chip. In total, there are up to four clock domains: the system clock for the AMBA and DMA layers, the SpaceFibre codec clock, the SpaceFibre transmitter clock (optional) and the SpaceFibre recovered clock for the receiver.

The block diagram of GRSPFI can be seen in the figure below:

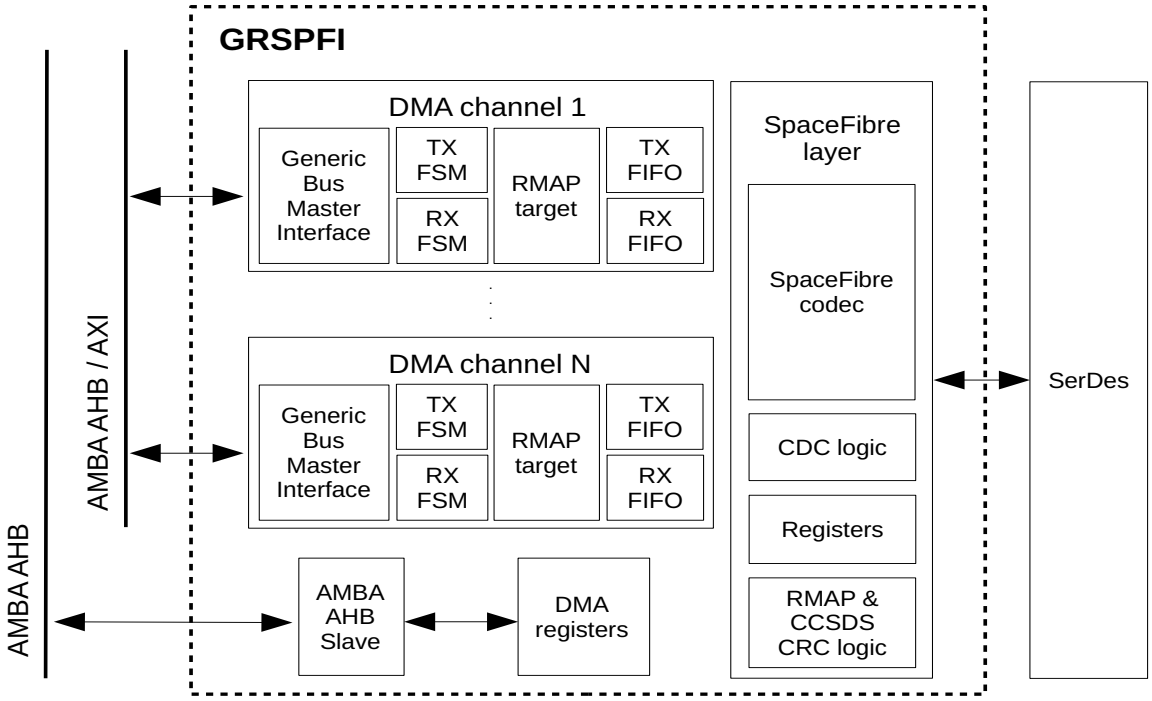


Figure 180. GRSPFI block diagram

74.2 Operation

74.2.1 Functional description

The main sub-blocks of the IP are the SpaceFibre codec, the DMA engine, the optional RMAP target and the IP registers accessed through the AHB slave interface. Note that the SerDes is not part of the IP, but it is always required in order to achieve the typical high bit-rates of SpaceFibre.

The core of the IP is the SpaceFibre codec. It has been designed in accordance with the SpaceFibre specification ECSS-E-ST-50-11C. The codec can be split into several layers from a functional point of view:

- Interface Layer. This layer serves as the interface between the Physical and the Lane layers. It is in charge of the symbol synchronization, 8B/10B encoding, cross-domain crossing and polarity inversion, among other tasks.
- Lane Layer. Functions here are those typically related to the Data-Link layer, according to the OSI model: link initialization management, error words counters, PRBS generator, loss of signal and standby detection, etc.
- Retry Layer. It contains the retry buffers, where transmitted words are stored until they are acknowledged, in case they shall be retransmitted. The Retry Layer also interprets the words received, implements the word priority logic and calculates the CRC.
- Virtual Channel Layer. This layer contains the Virtual Channel buffers for both transmission and reception, and implements FCT counter logic, control character monitoring and bandwidth and time-slot management.
- Broadcast Channel Layer. As the name suggests, the functionality related to the Broadcast Channel is covered here, including the monitoring of the bandwidth allocated to the Broadcast Channel.

The codec (GRSPFI_CODEC) is further described in a dedicated section of the GRLIB IP Core User's Manual as a separate IP. Please refer to it for further information about the configuration of the core and its control and status signals. Notice that the codec does not feature any internal register.

On the other hand, GRSPFI features a flexible DMA engine, in which the number of DMA channels is configurable from 1 to 8 via a generic. Each DMA channel comprises a transmit and a receive finite state machine, as well as RX and TX FIFOs. Additionally, every DMA channel has its own bus master interface for accessing the AMBA bus and an optional RMAP target.

In runtime, the user shall assign the Virtual Channels and the Broadcast Channel to the existing DMA channels. There are no restrictions on how many VC/BC channels can be handled by each DMA channel: arbitration among the channels is resolved in a Round-Robin fashion, except for the Broadcast Channel which always has the highest priority. However, it must absolutely be avoided to assign the same VC or the BC to more than one DMA channel, as this could potentially result in a packet being transmitted or stored multiple times, thus compromising the functionality of the IP.

Transmission and reception of packets for the Virtual Channels are based on descriptors. There are two descriptor tables per VC: one for transmission and another one for reception. Once a descriptor has been read and verified by the DMA channel handling the Virtual Channel, the packet can be fetched from external memory and transmitted using the codec (transmitter) or read from the codec and stored into external memory (receiver). Packets and descriptors are read and written through the generic bus master interface.

A DMA channel also contains an RX and a TX FIFO for insertion/extraction of control characters (FILL, EOP and EEP) and for 32-bit word alignment. The FIFOs serve as the interface between the FSMs and the codec. Once a complete packet has been read by the SpaceFibre codec (transmission) or written to external memory (reception), the FIFO is emptied, ready for the next SpaceFibre packet.

The RMAP target is an optional part of the core which can be enabled with a VHDL generic. The RMAP target handles incoming packets which are determined to be RMAP commands instead of the receiver DMA engine. The RMAP command is decoded and, if it is valid, the operation is performed on the bus. If a reply was requested it is automatically transmitted back to the source by the RMAP transmitter.

Every DMA channel features its own RMAP target and associated AHB registers, thus making it possible to enable and configure the RMAP targets separately. A DMA channel only checks the format of packets received over the assigned virtual channels.

The bus master interfaces present in the IP are generic, so that the user can connect a wrapper in order to adapt it to their custom protocol. Wrappers for AMBA 2.0 AHB and AXI4 are readily available from Frontgrade Gaisler.

GRSPFI includes the registers to configure and control the IP. These registers are accessible through the AMBA AHB slave interface.

GRSPFI will generate AMBA interrupts under certain circumstances:

- SpaceFibre Link events
- Errors during read/write accesses through the bus master interface
- Transmission and reception of SpaceFibre packets associated with a specific Virtual Channel
- Transmission and reception of broadcast messages

The user can configure which events shall cause interrupts both at IP level (by an AHB register) and per individual packet (a specific bit in the descriptor).

74.2.2 Protocol support

The SpaceFibre codec has been designed in accordance with the SpaceFibre standard ECSS-E-ST-50-11C. It is a Single-Lane implementation of the protocol; therefore the optional Multi-Lane layer is not part of the IP. The optional RMAP target has been implemented as per the ECSS-E-ST-50-52C specification.

The core only accepts packets with a valid destination address in the first byte received. Packets with address mismatch will be silently discarded. The second byte of a packet is typically the protocol ID and determines how a packet is processed by the IP. The RMAP protocol (ID = 0x01) is the only protocol handled separately in hardware while other packets are stored in external memory. If the RMAP target is present and enabled, all RMAP commands will be processed, executed and replied automatically in hardware. Otherwise RMAP commands are stored in memory via hardware descriptors in the same way as any other packets. RMAP replies are always handled via descriptors.

When the RMAP target is not present or is disabled, there is no need to include a protocol ID in the packets and the data can start immediately after the address. In transmission, the address and protocol-ID fields must be included in the buffers from where data is fetched. They are *not* automatically added by the core under any circumstances. The figure below depicts the packet formats supported by the core, depending on whether RMAP is present and enabled or not.

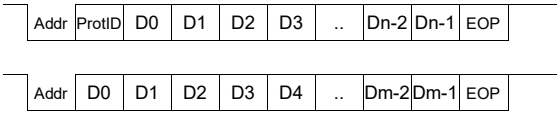


Figure 181. SpaceFibre packet types supported by the core

When a package is received with a protocol ID = 0x02 (CCSDS) the core can, if enabled, automatically calculate and check the CCSDS/CCITT CRC-16 and 16-bit ISO-checksum (J.G. Fletcher, ISO 8473-1:1998) of the data bytes. CCSDS packets are always handled via descriptors.

74.2.3 Endianness

The core is designed for big-endian and little-endian systems.

When connected to an AHB bus, GRSPFI automatically detects the endianness of the system by reading a dedicated sideband signal included in the AMBA records. If the AXI wrapper is used instead, the endianness is configured via the VHDL generic *lendian*. When accessing the bus, the data may be swapped depending on the selected endianness and the type of data involved in the transfer:

- Words belonging to TX and RX descriptors are never swapped and shall be read and written using 32-bit accesses. These words shall be located at the same address offsets (0x0, 0x4, 0x8 and 0xC) regardless of the endianness.
- Words associated with broadcast messages are never swapped and shall be accessed using 32-bit accesses, similarly to descriptors. Likewise the address offsets shall be preserved as well.
- Data in standard SpaceFibre packets are treated as a stream of bytes, i.e. the bytes will be located in the same memory positions in both big- and little-endian systems. If *addr* is the address configured in the corresponding field of a descriptor, the first byte shall be located at *addr*, the second byte at *addr+1*, and so on. This implies that GRSPFI will internally swap the data words in little-endian configurations when accessing the AMBA bus.
- Data in RMAP packets are also treated as a stream of bytes for all commands (Read, Write and Read-Modify-Write). The first data byte of an RMAP packet is associated with the address in the packet. GRSPFI will swap the data bytes in little-endian configurations. Note: the user shall take the endianness of the host system into account. This means that, when accessing APB registers and 32-bit descriptors via RMAP, the user shall swap the bytes of the command in little-endian host systems.

74.3 SpaceFibre codec

This section provides some guidelines for configuring and controlling the internal SpaceFibre codec. For a detailed description of generics and signals, please refer to the specific section dedicated to the SpaceFibre codec IP (GRSPFI_CODEC).

74.3.1 Configuration

Most generics available in GRSPFI are related to the functionality to implement in the SpaceFibre codec. Some of the most important generics are described below:

The generic *num_vc* determines the number of Virtual Channels featured by the core, which ranges from 1 to 32. Each Virtual Channel includes a pair of TX and RX buffers, whose depths are configured by *depth_vc_tx_buf* and *depth_vc_rx_buf*, respectively.

The Retry Layer contains separate buffers for data, broadcast and FCT words. Their depths are configured through the generics *depth_rbuf_data*, *depth_rbuf_bc* and *depth_rbuf_fct*, respectively.

The SpaceFibre standard requires 8B/10B encoding. However, some SerDes may already incorporate the encoding; otherwise this can be performed inside the codec by setting the generic *use_8b10b*. Depending on the SerDes interface, the IP can be configured to have a 20- or 40-bit wide interface with the generic *sel_16_20_bit_mode*. If the narrow interface of 20 bits is chosen, the generic *use_sep_txclk* shall be enabled, and the transmitter clock input signal (tx_clk) shall be provided.

Additional generics are required in order to get proper timing parameters in the IP, as well as to define the width of some counters. Please refer to 74.11.2 for a comprehensive description of the generics.

74.3.2 Control

The SpaceFibre link is controlled by means of the Codec Control Register (CTRL). The link can be reset by setting the Link Reset bit (CTRL.LIR). As a consequence, outbound packets are spilled up to and including the next EOP and inbound packets are terminated by an EEP.

If the Lane Start bit (CTRL.LS) is set, the Lane Layer in the codec will actively try to build up a connection with the far-end by sending out INIT1 words. If the Auto-start bit (CTRL.AS) is set, the lane layer will passively listen to incoming words from the far-end node and start the handshake procedure once INIT1 words are received. To keep the link disabled, both CTRL.LS and CTRL.AS shall be kept low. The lane can be reinitialized by setting the CTRL.LNR bit to '1'; note that this does not reset either the registers nor the codec. Ongoing packets will probably get delayed, but never canceled.

The scrambling of data is controlled by the CTRL.SCR bit. The codec also supports loop-back mode for debugging purpose by enabling the CTRL.LB bit. The standby reason can be specified by writing the CTRL.SBR field.

74.3.3 Status

The codec provides several signals to inform about its internal status. These signals are mapped to the Lane Layer Status Register (LNSTS) and the Retry Layer Status Register (RTSTS). Among others, the following signals are available:

- Lane state
- Far-end status: link reset, standby, loss of signal
- Error counters
- Polarity inversion
- Link errors: CRC-16 and CRC-8 errors, frame errors, sequence errors, protocol error, etc.
- Connection timeout

The capability registers contain information regarding how the IP was generated. Most generics can be read back by reading the Codec General Capability Register and the Codec Internal Buffers Capability Register.

74.4 Virtual Channels

The core supports from 1 to 32 Virtual Channels. Each Virtual Channel has its own sets of registers for individual configuration, control and monitoring. The Quality of Service (QoS) in SpaceFibre is mostly configured in these registers. Additionally, transmission and reception of SpaceFibre packets are handled through descriptors; each channel has its own pair of TX and RX descriptor tables. The VC registers also include mechanisms to filter out and truncate packets based on their length and the first address byte.

74.4.1 Global configuration and status of the channel

Before transmitting and receiving packets over a specific Virtual Channel, the Quality of Service parameters shall be configured for the channel.

The VC control register contains 16 bits to establish the bandwidth allocated to the channel (VCCTRL.VBW). The expected bandwidth is expressed in 8.8 fixed point format and calculated as follows:

$$ExpectedBandwidth = \left| 1 - \frac{100}{Percentage} \right|$$

For example, the expected bandwidth value of 15% is expressed as: abs(1-(100/15)) = 5.6667. In 8.8 fixed point notation: 0x05ab. For convenience, typical integer values are shown in the table below, ranging from 1% to 95%:

Table 1177. Hexadecimal representation of bandwidth percentages for the Virtual Channels

BW	Hex	BW	Hex	BW	Hex	BW	Hex
1%	0x6300	25%	0x0300	49%	0x010a	73%	0x005f
2%	0x3100	26%	0x02d9	50%	0x0100	74%	0x005a
3%	0x2055	27%	0x02b4	51%	0x00f6	75%	0x0055
4%	0x1800	28%	0x0292	52%	0x00ec	76%	0x0051
5%	0x1300	29%	0x0273	53%	0x00e3	77%	0x004c
6%	0x0fab	30%	0x0255	54%	0x00da	78%	0x0048
7%	0x0d49	31%	0x023a	55%	0x00d1	79%	0x0044
8%	0x0b80	32%	0x0220	56%	0x00c9	80%	0x0040
9%	0x0a1c	33%	0x0208	57%	0x00c1	81%	0x003c
10%	0x0900	34%	0x01f1	58%	0x00b9	82%	0x0038
11%	0x0817	35%	0x01db	59%	0x00b2	83%	0x0034
12%	0x0755	36%	0x01c7	60%	0x00ab	84%	0x0031
13%	0x06b1	37%	0x01b4	61%	0x00a4	85%	0x002d
14%	0x0625	38%	0x01a2	62%	0x009d	86%	0x002a
15%	0x05ab	39%	0x0190	63%	0x0096	87%	0x0026
16%	0x0540	40%	0x0180	64%	0x0090	88%	0x0023
17%	0x04e2	41%	0x0170	65%	0x008a	89%	0x0020
18%	0x048e	42%	0x0162	66%	0x0084	90%	0x001c
19%	0x0443	43%	0x0153	67%	0x007e	91%	0x0019
20%	0x0400	44%	0x0146	68%	0x0078	92%	0x0016
21%	0x03c3	45%	0x0139	69%	0x0073	93%	0x0013
22%	0x038c	46%	0x012d	70%	0x006e	94%	0x0010
23%	0x0359	47%	0x0121	71%	0x0069	95%	0x000d
24%	0x032b	48%	0x0115	72%	0x0064		

The control register also has 4 bits for the priority of the channel (VCCTRL.PR), the lower values having higher priority (0 being the highest). On the other hand, since there are up to 64 different time-slots, two different registers are used for specifying when a Virtual Channel is allowed to transmit: VC Time-slot 1 Register (VCTS1) for time-slots 0-31 and VC Time-Slot 2 Register (VCTS2) for time-slots 32-63. Please refer to the SpaceFibre specification for further information regarding the QoS built into the protocol.

The status of a specific Virtual Channel can be obtained by reading the Virtual Channel Status Register. The register informs about bandwidth utilization (under-use and over-use flags). The FCT Overflow bit tells whether the FCT credit counter has overflowed; this indicates that the number of bits for the FCT counter (configured via the generic *remote_fct_cnt_max*) is too small. The Destination Credit bit indicates whether there is space in the virtual channel input buffer of the far end node. The Input Buffer Overflow bit informs about the VC input buffers being full.

74.4.2 TX descriptors

Transmission of SpaceFibre packets is handled by means of descriptors. In transmission, descriptors indicate where the packet is to be fetched from (both its header and its data segments) and control several aspects of the transmission process, such as interrupt generation and how the hardware pointers handling the descriptors shall be updated.

The core reads descriptors from an area in memory pointed to by the Transmit Descriptor Table Address Register. The register consists of a base address and a descriptor selector. The base address points to the beginning of the area and must start on an address that is aligned to the size of the descriptor table. Each Virtual Channel has its own descriptor table.

The number of descriptors in the table can be configured by the generic *num_txdesc*. Accepted values are 64, 128, 256 and 512. The number of bits allocated to the base address and descriptor selector in the Transmit Descriptor Table Address Register depends on the value of this generic. A TX descriptor consists of four 32-bit words (16 bytes). Therefore, the size of the descriptor table can be determined as *num_txdesc * 16* bytes. The number of TX descriptors can also be obtained by reading the General Capabilities Register.

To transmit packets, one or more descriptors have to be initialized in memory. Each descriptor shall specify the number of bytes to transmit and the location of the data. There are two different length and address fields in the transmit descriptors because there are separate pointers for header and data. If a length field is zero the corresponding part of a packet is skipped and if both are zero no packet is sent. The maximum header length is 255 bytes and the maximum data length is 16 Mbyte - 1. When the pointer and length fields have been configured the descriptor enable bit should be set to validate the descriptor. This must always be done last. The other control bits must also be set before enabling the descriptor.

The figures below depict each word and describe the use of every field:

Table 1178. GRSPFI TX descriptor word 0 (offset 0x0)

31 20	19 18	17	16	15 12	11	10	9	8	7 0
RESERVED	CRCT	DC	HC	NONCRCLEN	R	IE	WR	EN	HEADERLEN

31: 20	RESERVED
19: 18	CRC Type (CRCT) - Defines the type of CRC or checksum to use for the data bytes. 00: RMAP CRC 01: CCSDS/CCITT CRC-16 10: 16-bit ISO Checksum (J.G. Fletcher, ISO 8473-1:1998) 11: Reserved
17	Append data CRC (DC) - Append CRC calculated according to the algorithm selected in the CRCT field (RMAP specification, CCSDS/CCITT CRC-16 or 16-bit ISO-checksum) after the data sent from the data pointer. The CRC covers all the bytes from this pointer. A null CRC will be sent if the length of the data field is zero.
16	Append header CRC (HC) - Append CRC calculated according to the RMAP specification after the data sent from the header pointer. The CRC covers all bytes from this pointer except a number of bytes in the beginning specified by the NONCRCLEN field. The CRC will not be sent if the header length field is zero.
15: 12	Non-CRC bytes (NONCRCLEN)- Sets the number of bytes in the beginning of the header which should not be included in the CRC calculation. This is necessary when using path addressing since one or more bytes in the beginning of the packet might be discarded before the packet reaches its destination.
11	RESERVED
10	Interrupt enable (IE) - If set, an interrupt will be generated once the packet has been transmitted as long as the transmitter interrupt enable bit in the DMA IRQ control register is also set.
9	Wrap (WR) - If set, the descriptor pointer will wrap and the next descriptor read will be the first one in the table (at the base address). Otherwise the pointer is increased with 0x10 to use the descriptor at the next higher memory location.
8	Enable (EN) - Enable transmitter descriptor. When all control fields (address, length, wrap) are set, this bit should be set. While the bit is set the descriptor should not be touched since this might corrupt the transmission. The core clears this bit when the transmission has finished.
7: 0	Header length (HEADERLEN) - Header Length in bytes. If set to zero, the header is skipped.

Table 1179. GRSPFI TX descriptor word 1 (offset 0x4)

31	0
HEADERADDRESS	

31: 0 Header address (HEADERADDRESS) - Address from where the packet header is fetched. No alignment restriction.

Table 1180. GRSPFI TX descriptor word 2 (offset 0x8)

31	24	23	0
RESERVED		DATALEN	

31: 24 RESERVED

23: 0 Data length (DATALEN) - Length in bytes of data part of packet. If set to zero, no data will be sent. If both data and header lengths are set to zero no packet will be sent.

Table 1181. GRSPFI TX descriptor word 3 (offset 0xC)

31	0
DATAADDRESS	

31: 0 Data address (DATAADDRESS) - Address from where data is read. No alignment restriction.

Once the TX descriptors have been written to the descriptor table, the TX descriptors available bit (TDA) in the corresponding VC descriptor control register must be set. The transmission process will not start until a DMA channel is assigned the current VC. The sequence to set up the DMA channels is explained later. New descriptors can be activated in the table on the fly (while transmission is active). Each time a set of descriptors is added the TDA bit shall be set again, as it is automatically cleared when the DMA channel finds a disabled descriptor.

The internal pointer which is used to keep the current position in the descriptor table can be read and written through the AHB interface. This pointer is set to zero during reset and is incremented each time a descriptor is used. It wraps automatically when the limit for the descriptor table is reached, or it can be set to wrap earlier by setting a bit in the current descriptor.

The descriptor table register can be updated with a new table anytime when the transmitter of the DMA channel handling the current VC channel is not active. A status bit is available in the DMA Channel Status Register (one register per DMA channel). A value of ‘0’ in the TX Active bit (TA) shall be read to consider it safe to modify the TX descriptor table. Once that happens, the user may change the location of the descriptor table, the content of the current table or the pointer to the next descriptor to be fetched by the core.

74.4.3 RX descriptors

As in transmission, reception of SpaceFibre packets is handled by means of descriptors. In this case, descriptors indicate where the packet is to be stored. Some additional control bits are available as well to handle the interrupt generation and how the descriptor pointer shall be updated once the next reception finishes.

The core reads the descriptors from an area in memory pointed to by the RX Descriptor Table Address Register. The register consists of a base address and a descriptor selector. The base address points to the beginning of the area and must start on an address that is aligned to the size of the descriptor table. Each Virtual Channel has its own descriptor table.

The number of descriptors in the table can be configured by the generic *num_rxdesc*. Accepted values are 128, 256, 512 and 1024. The number of bits allocated to the base address and descriptor selector in the RX Descriptor Table Address Register depends on the value of this generic. An RX descriptor consists of two 32-bit words (8 bytes). Therefore, the size of the descriptor table can be determined as

*num_rxdesc * 8* bytes. The number of RX descriptors can also be obtained by reading the General Capabilities Register.

SpaceFibre packets are received by the codec regardless of which descriptors have been initialized. The packet is temporarily stored in the corresponding Virtual Channel buffer, waiting for a DMA channel to read it and store it in external memory. For the latter to happen, the descriptors corresponding to the selected VC shall be initialized.

Similarly to the transmitter, descriptors include a pointer to where the data shall be stored. In this case, however, the header and data are not discriminated, and the length of the packet is to be written by the core once the reception finishes. Once the pointer is set, and the control bits (wrap bit, interrupt enable) are also configured, the enable bit should be set to validate the descriptor. This must always be done last to avoid reading a corrupt descriptor.

The figures below depict each word of the RX descriptor and describe the meaning of every bit field:

Table 1182. GRSPFI RX descriptor word 0 (offset 0x0)

31	30	29	28	27	26	25	24	0
DC	HC	TR	EP	IE	WR	EN	PACKETLENGTH	

31	Data CRC Error (DC) - Set by the core if an RMAP CRC error was detected for the data. If the Protocol ID of the received packet is 0x02, the interpretation of this field changes: a 1 indicates that a CCSDS/CCITT CRC-16 error was detected.
30	Header CRC Error (HC) - Set by the core if an RMAP CRC error was detected for the header. If the Protocol ID of the received packet is 0x02, the interpretation of this field changes: a 1 indicates that a 16-bit ISO-Checksum error was detected.
29	Packet truncated (TR) - Set by the core when the reception has finished if the length of the packet exceeds the maximum length. The rest of the packet is spilled.
28	EEP termination (EP) - This packet ended with an Error End of Packet control character.
27	Interrupt enable (IE) - If set, an interrupt will be generated once a packet has been received as long as the receiver interrupt enable bit in the DMA IRQ control register is also set.
26	Wrap (WR) - If set, the descriptor pointer will wrap and the next descriptor read will be the first one in the table (at the base address). Otherwise the pointer is increased with 0x8 to use the descriptor at the next higher memory location.
25	Enable (EN) - Enable (EN) - Set to one to activate this descriptor. This means that the descriptor contains valid control values and the memory area pointed to by the packet address field can be used to store a packet. When all control fields are set, this bit should be set. While the bit is set the descriptor should not be touched since this might corrupt the reception. The core clears this bit when the reception has finished.
24: 0	Packet length (PACKETLENGTH) - The number of bytes received to this buffer. It is set by the IP, and is only valid after EN has been set to 0 by the GRSPFI core.

Table 1183. GRSPFI RX descriptor word 1 (offset 0x4)

31	0
PACKETADDRESS	

31: 0	Packet address (PACKETADDRESS) - The address pointing to the buffer which will be used to store the packet received.

Once the RX descriptors are written to the descriptor table, the RX descriptors available bit (RDA) in the corresponding VC descriptor control register must be set. The packet is not stored in external memory until a DMA channel is assigned the current VC. The sequence to set up the DMA channels is explained later. New descriptors can be activated in the table on the fly (while reception is active). Each time a set of descriptors is added the RDA bit shall be set again, as it is automatically cleared when the DMA channel finds a disabled descriptor.

The internal pointer which is used to keep the current position in the descriptor table can be read and written through the AHB interface. This pointer is set to zero during reset and is incremented each

time a descriptor is used. It wraps automatically when the limit for the descriptor table is reached, or it can be set to wrap earlier by setting a bit in the current descriptor.

The descriptor table register can be updated with a new table anytime when the receiver of the DMA channel handling the current VC channel is not active. A status bit is available in the DMA Channel Status Register (one register per DMA channel). A value of 0b in the RX Active bit (RA) shall be read to consider it safe to modify the RX descriptor table. Once that happens, the user may change the location of the descriptor table, the content of the current table or the pointer to the next descriptor to be fetched by the core.

74.4.4 Packet maximum length filter (RX only)

GRSPFI features a mechanism to limit the size of the incoming packets that will be stored in memory. A separate maximum length filter exists per virtual channel.

The VC RX Maximum Length Register defines the maximum number of bytes that will be accepted by the core for a given SpaceFibre packet. If the length of a packet exceeds the value of the register, the Truncate bit in the RX descriptor is set and the rest of the packet is spilled, i.e. not stored in external memory. It is mandatory to configure this register prior to starting a reception, otherwise the default length is zero and no bytes are accepted.

74.4.5 Address filters (RX only)

GRSPFI features two address filters, one general and one specific to each VC. When a SpaceFibre packet is received over a Virtual Channel, the first byte is interpreted as the packet local address. An acceptance filter is applied to this address in order to decide whether the packet shall be accepted (stored in memory) or ignored (read from the SpaceFibre codec but never written to external memory). The Address Enable bit in the VC control register determines which filter to apply:

- Default filter. If the Address Enable bit is 0b, the default filter is used. The filter consists of 2 bytes: a default address and a default mask, which are configured in the GRSPFI Default Address Register. The SpaceFibre packet address is compared with the default address after applying the mask: those bits set to 1b in the corresponding position of the mask are ignored. The following condition must be fulfilled in order to accept a SpaceFibre packet when using the default filter:

$$PKTADDR \text{ and not } DEFMASK = DEFADDR \text{ and not } DEFMASK$$

- VC filter. If the Address Enable bit is 1b, the VC-specific filter is used. The filter consists of 2 bytes: a VC address and a VC mask, which are configured in the VC Address Register. The SpaceFibre packet address is compared with the VC address after applying the mask: those bits set to 1b in the corresponding position of the mask are ignored. The following condition must be fulfilled in order to accept a SpaceFibre packet when using the VC filter:

$$PKTADDR \text{ and not } VCMASK = VCADDR \text{ and not } VCMASK$$

Setting all bits of the corresponding mask to 1b disables the filter, and all packet addresses are therefore accepted.

74.5 Broadcast Channel

GRSPFI features a Broadcast Channel which can be controlled by any of the existing DMA channels. However, only one DMA channel can handle the Broadcast Channel at a time to avoid duplicity of transmission or reception of messages. In SpaceFibre, broadcast messages contain 3 main fields (8 bytes of data, broadcast type and broadcast channel number) and two flags (delayed and late). For further information please refer to the SpaceFibre standard.

Unlike the Virtual Channels, communication through the Broadcast Channel is not based on descriptors. Since the length of the message is defined and much shorter than SpaceFibre packets may potentially be, the messages are stored in a circular buffer, externally to the IP. The content of those buffers is handled via hardware pointers. The following chapters describe how the Broadcast Channel is managed and how the messages are mapped in memory.

74.5.1 General

The Broadcast Channel Configuration Register can be used to configure the main parameters of the channel.

The transmission of broadcast messages is limited to the expected bandwidth of the Broadcast Channel. The user shall specify the expected bandwidth by writing the 16-bit BW field in the BC Configuration Register. The bandwidth is expressed in 0.16 format, and calculated as follows:

$$ExpectedBandwidth = \frac{Percentage}{400}$$

For example, the expected bandwidth value of 10% is expressed as: 10/400 = 0.025. In 0.16 fixed point notation: 0x0666. For convenience, the following table shows the most common bandwidth values, from 1% to 95%:

Table 1184. Hexadecimal representation of bandwidth percentages for the Broadcast Channel

BW	Hex	BW	Hex	BW	Hex	BW	Hex
1%	0x00a4	25%	0x1000	49%	0x1f5c	73%	0x2eb8
2%	0x0148	26%	0x10a4	50%	0x2000	74%	0x2f5c
3%	0x01ec	27%	0x1148	51%	0x20a4	75%	0x3000
4%	0x028f	28%	0x11ec	52%	0x2148	76%	0x30a4
5%	0x0333	29%	0x128f	53%	0x21ec	77%	0x3148
6%	0x03d7	30%	0x1333	54%	0x228f	78%	0x31ec
7%	0x047b	31%	0x13d7	55%	0x2333	79%	0x328f
8%	0x051f	32%	0x147b	56%	0x23d7	80%	0x3333
9%	0x05c3	33%	0x151f	57%	0x247b	81%	0x33d7
10%	0x0666	34%	0x15c3	58%	0x251f	82%	0x347b
11%	0x070a	35%	0x1666	59%	0x25c3	83%	0x351f
12%	0x07ae	36%	0x170a	60%	0x2666	84%	0x35c3
13%	0x0852	37%	0x17ae	61%	0x270a	85%	0x3666
14%	0x08f6	38%	0x1852	62%	0x27ae	86%	0x370a
15%	0x099a	39%	0x18f6	63%	0x2852	87%	0x37ae
16%	0x0a3d	40%	0x199a	64%	0x28f6	88%	0x3852
17%	0x0ae1	41%	0x1a3d	65%	0x299a	89%	0x38f6
18%	0x0b85	42%	0x1ae1	66%	0x2a3d	90%	0x399a
19%	0x0c29	43%	0x1b85	67%	0x2ae1	91%	0x3a3d
20%	0x0ccd	44%	0x1c29	68%	0x2b85	92%	0x3ae1
21%	0x0d71	45%	0x1ccd	69%	0x2c29	93%	0x3b85
22%	0x0e14	46%	0x1d71	70%	0x2ccd	94%	0x3c29
23%	0x0eb8	47%	0x1e14	71%	0x2d71	95%	0x3ccd
24%	0x0f5c	48%	0x1eb8	72%	0x2e14		

On the other hand, the broadcast channel number of the core shall also be configured in the same register by writing the corresponding 8-bit CHN field. The broadcast channel number is unique for a node in a SpaceFibre network, so all broadcast messages originated in the core will use this channel.

74.5.2 Setting up the transmission of broadcast messages

Broadcast messages to be transmitted are stored in a circular buffer external to the IP. The representation of the message is explained in 74.5.4. The content of the buffer is handled by means of two hardware pointers, which inform about any new available message and about the messages already transmitted. This section describes how to set up the transmitter of the Broadcast Channel. However, a transmission cannot start until a DMA channel is assigned the Broadcast Channel. This is discussed in a separate chapter corresponding to the DMA engine.

The Broadcast Channel TX Address Register shall be written in order to set the address of the circular buffer where messages are located. The address shall be aligned to 1kB boundary. The last valid position in the circular buffer is specified in the Broadcast Channel TX size register; in this case, the address is 16-byte aligned.

The hardware pointers are accessed via the Broadcast Channel TX Write Pointer Register and Broadcast Channel TX Read Pointer Register, respectively. The pointers are 16-byte aligned. The write pointer is handled by the user to indicate that there are new messages ready to be transmitted. It shall point to the position + 1 of the last broadcast message written to the buffer. On the contrary, the read pointer is normally modified by the core to inform about the messages already transmitted. It will point to the position + 1 of the last message read from the buffer. The user may modify the read pointer if a different starting point is desired; however, this may only be done when the DMA handling the Broadcast Channel is disabled and not active.

Once a pointer reaches the last valid position of the buffer, it will wrap around to zero. At least one position of the buffer must be left empty at any time to handle the content correctly. Otherwise, a full buffer would be simply interpreted as an empty buffer, so no transmission would happen.

74.5.3 Setting up the reception of broadcast messages

Received broadcast messages are stored in a circular buffer external to the IP. The representation of the message is explained in 74.5.4. The content of the buffer is handled by means of two hardware pointers, which inform about any new message received and about messages already read from the circular buffer by the user. This section describes how to handle the receiver of the Broadcast Channel. However, a message will not be stored in the circular buffer until a DMA channel is assigned the Broadcast Channel. This is discussed in a separate chapter corresponding to the DMA engine.

The Broadcast Channel RX Address Register shall be written in order to determine the location of the circular buffer where received messages are to be stored. The address shall be aligned to 1kB boundary. The last valid position in the circular buffer is specified in the Broadcast Channel RX size register; in this case, the address is 16-byte aligned.

The hardware pointers are accessed via the Broadcast Channel RX Write Pointer Register and Broadcast Channel RX Read Pointer Register, respectively. The pointers are 16-byte aligned. The write pointer is normally handled by the core to indicate that there are new received messages ready to be read by software. It shall point to the position + 1 of the last broadcast message written to the buffer. The user may modify the write pointer if a different starting point is desired; however, this may only be done when the DMA handling the Broadcast Channel is disabled and not active. On the contrary, the read pointer is modified by the user to inform about messages already read from the buffer, thus making room for new messages.

Once a pointer reaches the last valid position of the buffer, it will wrap around to zero. At least one position of the buffer will be left empty at any time to handle the content correctly. Otherwise, a full buffer would be simply interpreted as an empty buffer, so new messages would overwrite others yet pending to be read by the user.

74.5.4 Broadcast message memory mapping

Each broadcast message is represented by 16 bytes. This representation is common to both transmission and reception. The four words are described below:

Table 1185. Broadcast message word 0 (offset 0x0)

31 18	17	16	15 8	7 0
RESERVED	DL	LT	BCH	BTYPE

31: 18	RESERVED
17	Delayed flag (DL) - This flag is set when a broadcast message is delayed in a routing switch because either a previous broadcast message was still in the process of being sent, or several broadcast messages arrived at the same time.
16	Late flag (LT). The LATE flag is set when a broadcast message has been delayed because of an error. The LATE flag is used in the broadcast mechanism to prevent duplication of a broadcast message when a copy of the broadcast message is delayed.
15: 8	Broadcast channel (BCH). Broadcast channel number from which the message is being sent. It is ignored in transmission (the channel number is configured via an AHB register instead).
7: 0	Broadcast type (BTYPE). This byte specifies the meaning of the data being transmitted in the current message.

Table 1186. Broadcast message word 1 (offset 0x4)

31 0
RESERVED

31: 0	RESERVED

Table 1187. Broadcast message word 2 (offset 0x8)

31 0
DATA_H

31: 0	DATA_H: 4 most significant bytes of the broadcast data (equivalent to data (63:32)).

Table 1188. Broadcast message word 3 (offset 0xC)

31 0
DATA_L

31: 0	DATA_L: 4 least significant bytes of the broadcast data (equivalent to data (31:0)).

The broadcast channel number is unique for a node in a SpaceFibre network. Therefore, when transmitting a broadcast message, the BCH field in the first word is ignored; instead, the channel number is configured in the Broadcast Channel Configuration Register through the AHB slave interface.

74.6 DMA engine

74.6.1 Functional description

The DMA engine featured in GRSPFI is very flexible: the number of DMA channels is configurable through the generic *num_dmach*, whereas the Virtual or Broadcast channels assigned to each DMA channel can be selected in runtime through the AHB slave interface. Every DMA channel has its own set of registers: a DMA Control Register, a DMA Status Register, a DMA IRQ Control Register and a DMA VC Mapping Register. All implemented DMA channels operate in parallel.

There are no restrictions regarding how many channels (Virtual or Broadcast) can be assigned to a specific DMA channel. Likewise, a DMA does not have to be assigned any channel, in such a case the

channel would simply remain idle. Similarly, a Virtual Channel may not be assigned to any DMA channel: this effectively switches off that channel and no packets will be transmitted or received.

In any case, the user must ensure that a specific Virtual Channel is not assigned to more than one DMA channel simultaneously. Otherwise, a packet may be transmitted or stored twice, thus compromising the functionality of the IP.

The behavior is similar for both DMA transmitters and receivers. When idle, if a specific DMA channel is in charge of the Broadcast Channel, it will first check if there is any broadcast message pending to be fetched and transmitted (TX) or read and stored (RX). Otherwise, it will proceed to check if there is any descriptor available among the Virtual Channels assigned to it. The Virtual Channels are served in a Round-Robin fashion at DMA level, and the priority of the channel is later applied within the internal SpaceFibre codec, as part of the protocol built-in Quality of Service.

After the transmission or reception of a packet finishes, GRSPFI updates the corresponding AHB registers accordingly. For broadcast messages, the pointers to the circular buffers are updated; for Virtual Channels, the first word of the descriptor is overwritten with the updated information.

Once a DMA channel takes up a task, it will attempt to finish it before serving any other virtual / broadcast channel. For transmission, that means that a complete SpaceFibre packet will be sent; once this is done, the DMA channel will check again which channel is to be served next. The same applies to reception. If the transmitter or receiver are disabled while the DMA channel is still operating, it will still attempt to complete the operation before effectively switching off. This can be observed in the DMA Channel Status Register, by checking the bits TX Active and RX Active.

Every DMA channel comprises its own bus master interface, to be shared by its transmitter and receiver. The receiver has priority over the transmitter when it comes to accessing the bus, in order not to delay the reception of packets and thus preventing potential loss of data due to buffer overrun.

Optionally, GRSPFI can include a hardware RMAP target to decode incoming RMAP commands, perform the corresponding AMBA accesses and transmit the RMAP reply if applicable. Each DMA channel features its own RMAP target and verifies only the commands received over the virtual channels assigned to the channel. Even though the RMAP target is part of the DMA engine, it is described in a separate subsection of this manual.

74.6.2 Assigning Virtual / Broadcast channels to a DMA channel

In order to start fetching and storing messages from/to external memory, the DMA channel needs to be assigned any Virtual Channel or the Broadcast Channel. In the case of the Virtual Channels, this is done in the DMA Channel VC Mapping Register; each position represents a Virtual Channel. Setting a bit to 1b implies that the VC will be attended by that DMA channel.

Likewise, the Broadcast Channel is assigned to a DMA channel by writing the MAP bits in the Broadcast Channel Mapping Register. These bits are encoded as the DMA channel number (from 0 to 31). The user shall avoid assigning the BC to a non-implemented DMA Channel, as the behavior is undefined. By default, the DMA Channel 0 is in charge of handling the broadcast communication.

It is advisable to distribute the assignment of channels among all the available DMA channels in an efficient way. For instance, the Broadcast Channel and Virtual Channels only transmitting short packets could be assigned to a DMA channel. Another DMA channel would then be in charge of longer packets with lower priority, in order to avoid delaying high priority packets.

74.6.3 Receive DMA channel

Once Virtual and Broadcast Channels have been properly assigned, the receiver of the DMA channel can be enabled by setting the RE bit of the DMA Channel Control Register to 1b. After that, the DMA channel will automatically poll the channels assigned to it to determine if a packet processed, either from the Virtual or the Broadcast channels. Different tasks are given a different priority, as listed below (from top to lowest priority):

1. Store a broadcast message in memory.
2. Decode and process an incoming RMAP command.
3. Store a packet received over a virtual channel in memory if there are prefetched RX descriptors.
4. Prefetch a new RX descriptor.

Therefore, the receive channel will only perform a task if it does not have any higher priority task pending. For instance, a new RX descriptor can only be prefetched if there are no broadcast messages, RMAP commands to decode or packets ready to store in memory.

The receiver of the DMA channel needs to be enabled to perform any of the previous operations, so this requirement will not be listed in the subsequent paragraphs.

The Broadcast Channel has priority over any other Virtual Channel. If the Broadcast Channel is assigned to the DMA channel, the following conditions must be met in order to write a broadcast message to the circular buffer:

- BC mapped to the current DMA channel.
- The SpaceFibre codec has already received a valid broadcast message which is temporarily stored in a local FIFO in the DMA layer.
- There is space in the circular buffer (both hardware pointers are compared. At least 2 positions shall be free before storing the broadcast message, as the circular buffer must not be completely full at any time).

When the conditions above are met, the broadcast message is written to the circular buffer through the bus master interface. Once it finishes, the write pointer is updated, and an AMBA interrupt is generated if the BRE bit of the Interrupt Control Register is set.

The second highest priority task is the processing of incoming RMAP commands. The processing is autonomous and requires no software intervention, which implies that no descriptor is involved in the process. The conditions to start processing an RMAP command are listed below:

- RMAP target present (generic *rmap* set to 1) and enabled for a given virtual channel.
- The Virtual Channel is assigned to the current DMA channel.
- The packet has RMAP format: the first word of the packet contains no control characters, the second byte is set to 0x01 (RMAP Protocol ID) and the two most significant bits of the third byte (RMAP instruction) are set to 01b (RMAP command).

Processing an RMAP command means that the receiver decodes the fields of the command, parsing the data length and checking the header and data CRC. The address and key are checked and the transmitter is requested to send a reply if it was requested by the RMAP initiator, so long as the RMAP decoding was successful; otherwise the corresponding error code is returned. The RMAP target is explained later in a dedicated section.

If no broadcast or RMAP packets are available, standard packets received over the Virtual Channels assigned to the DMA channel are checked. The DMA engine must first prefetch the descriptors, regardless of whether data has been received over a virtual channel. Once the descriptor is prefetched and the codec has detected the beginning of the packet, the DMA engine can start storing the data in external memory via the bus master interface. The following conditions must be met in order for the DMA engine to start storing a SpaceFibre packet received over a given Virtual Channel in external memory:

- The Virtual Channel is assigned to the current DMA channel.
- The DMA engine had already prefetched and validated an RX descriptor for the Virtual Channel, i.e. the descriptor is enabled.
- The Virtual Channel in question shall have valid non-RMAP data in its output buffer, within the SpaceFibre codec.

If these conditions are satisfied, the entire SpaceFibre packet is stored in external memory through the bus master interface. Only if there are no Virtual Channels fulfilling the conditions above, the DMA engine may try to prefetch RX descriptors. In this case, the requirements for a given Virtual Channel are as follows:

- The Virtual Channel is assigned to the current DMA channel.
- There are descriptors available for the selected Virtual Channel. This is done by checking the RDA bit of the corresponding VC Descriptor Control Register.
- The DMA engine does not have an RX descriptor already prefetched for the Virtual Channel in question.

Once the reception ends and the packet is completely stored in memory, GRSPFI overwrites the RX descriptor with the corresponding status bits: length of the packet received, EEP detected and whether the packet was truncated due to maximum length violation. The enable bit is cleared to indicate that the descriptor has already been used. If the CRC logic is implemented, two fields indicate whether there were CRC or checksum errors. An AMBA interrupt will be generated if enabled both in the descriptor and in the DMA IRQ Control Register. A new RX descriptor can then be prefetched for that Virtual Channel.

The DMA Channel Status Register contains valuable information regarding the reception process. The RX Active bit indicates that the receive FSM is enabled and may be in a status other than idle: it may have started processing a SpaceFibre packet or a broadcast message. Disabling the receive channel will not have any effect until the ongoing operation ends (if there is any). The RX packet bit indicates that a SpaceFibre has been received and stored in external memory by this DMA channel, and the descriptor has been updated accordingly.

74.6.4 Transmit DMA channel

Once Virtual and Broadcast Channels have been properly assigned, the transmitter of the DMA channel can be enabled. This is done by setting the TE bit of the DMA Channel Control Register to 1b. After that, the DMA channel will automatically poll the channels assigned to it to determine if a packet can be fetched from external memory and transmitted. Different tasks are given a different priority, as listed below (from top to lowest priority):

1. Fetch and transmit a broadcast message.
2. Build and send an RMAP reply if requested by the receiver of the DMA channel.
3. Fetch and transmit a packet over a Virtual Channel.

Therefore, the transmit channel will only perform a task if it does not have any higher priority task pending. For instance, a new packet can only be fetched and transmitted if there are no broadcast messages or RMAP replies to transmit.

The transmitter of the DMA channel needs to be enabled to perform any of the previous operations, so this requirement will not be listed in the subsequent paragraphs.

The Broadcast Channel has priority over any other Virtual Channel. The following conditions must be met in order to transmit a broadcast message:

- BC mapped to the current DMA channel.
- There are new broadcast messages pending for transmission (both hardware pointers are compared. They shall not point to the same location).

When the conditions above are met, the broadcast message is fetched from the circular buffer through the bus master interface. Once the complete message is read, it is immediately written to the Broadcast Channel input buffer in the SpaceFibre codec. The codec then acknowledges the message, so the DMA engine can move on and deal with the next packet or broadcast message. The read pointer is updated, and an AMBA interrupt is generated if the BTE bit of the Interrupt Control Register is set.

Please note that there may be a delay until the internal codec has finished the transmission of the message; the interrupt indicates that the DMA layer has passed the message to the SpFi codec.

If there are no pending broadcast messages, the transmitter checks if the receiver is requesting an RMAP reply to be sent. In this case, the conditions are as follows:

- RMAP target present (generic *rmap* set to 1) and enabled for a given virtual channel.
- Receive channel is requesting an RMAP reply.

The transmitter checks the current RMAP operation. For read commands the transmit performs AMBA read accesses to form the reply. For write and read-modify-write commands no further AMBA accesses are required and the reply can be sent immediately. The RMAP target is expanded in a dedicated section of this manual.

If any of the conditions above are not met, the Virtual Channels assigned to the DMA channel are checked. The following situations must be given prior to fetching a SpaceFibre packet from external memory:

- The Virtual Channel is assigned to the current DMA channel.
- There are descriptors available for the selected Virtual Channel. This is done by checking the TDA bit of the corresponding VC Descriptor Control Register.

If these conditions are satisfied, the TX descriptor is fetched. A packet will then be fetched as long as the descriptor determines that either (or both) the length of the header or the data is different from 0. If the length of the header or the data field is 0 (but not both), that part of the packet is skipped. However, if both are 0, no packet is transmitted, and the descriptor is overwritten with a 0b in the Descriptor Enabled bit.

When the packet is completely fetched, control characters (FILL, EOP or EEP) have been properly inserted and the last word is written to the Virtual Channel input buffer inside the codec, the transmission at DMA level is completed. GRSPFI overwrites the first word of the TX descriptor with the Descriptor Enabled bit set to 0b, since it has already been used by the IP. An AMBA interrupt will be generated if enabled.

It is important to realize that the AMBA interrupt does not imply that the packet has been transmitted over the SpaceFibre link. There may be delays in the link interface due to multiple packets from different Virtual Channels sharing the lane layer, as well as delays inherent to the codec itself. The codec will transmit the SpaceFibre packet as soon as possible, according to the SpaceFibre protocol. The interrupt does indicate that the DMA layer is done with the packet.

The DMA Channel Status Register contains valuable status information regarding the transmission. The TX Active bit indicates that the transmit FSM is enabled and may be in a state different from idle (processing a descriptor, passing a packet to the SpaceFibre codec or transmitting a broadcast message). If there is an ongoing operation, disabling the channel will not have any effect until the DMA engine finishes the transmission completely. The TX packet bit indicates that a SpaceFibre packet has been successfully fetched, aligned and passed to the codec ready for transmission, and the descriptor has been updated accordingly.

74.6.5 RMAP and CCSDS CRC logic

GRSPFI supports several algorithms to protect the content of the SpaceFibre packets. RMAP CRC logic is included in the implementation if the *rmapcrc* or *rmap* VHDL generics are set to 1. CCSDS/CCITT CRC-16 and 16-bit ISO Checksum logic is included in the implementation when the VHDL generic *ccsdscrc* is set to 1, as long as the RMAP CRC logic is also instantiated.

In reception, the RMAP CRC calculation is always active for all received packets and covers all data bytes. The packet is by default assumed to be an RMAP packet and the length of the header is determined by checking the third byte, which should be the command field. The calculated CRC value is then checked when the entire header has been received (according to the calculated number of bytes) and if it is non-zero the HC bit is set to indicate a header CRC error.

The CRC value is not set to zero after the header has been received, instead the calculation continues in the same way until the complete packet has been received. When the header is not corrupted the CRC value will always be zero at the beginning of the data field and the behavior will be as if the CRC calculation was restarted. Then if the resulting CRC value is non-zero the DC bit is set to indicate a data CRC error. This means that the core can indicate a data CRC error even if the data field was correct when the header CRC was incorrect. However, the data should not be used when the header is corrupt and therefore the DC bit is unimportant in this case.

If the received packet is not of RMAP type the header CRC error indication bit cannot be used. It is still possible to use the DC bit if the complete packet is covered by a CRC calculated using the RMAP CRC definition. This is because the core does not restart the calculation after the header has been received but instead calculates a complete CRC over the packet. Therefore any packet format with one CRC at the end of the packet calculated according to RMAP standard can be checked using the DC bit.

When a packet is received with a protocol ID set to 0x02 (CCSDS), the CCSDS CRC-16 and ISO checksum will replace the RMAP CRC logic. The result is presented by the same status bits as for the RMAP head/data CRC error, but the interpretation of these bits is changed to ISO-checksum/CCSDS CRC error instead.

Note that the DC/HC bits are only updated if the packet is not handled by the RMAP target. Otherwise, the handling of the packet does not involve RX descriptors and the CRC logic is internally managed by the RMAP target when decoding the RMAP command.

It the received packet is not covered by either an RMAP CRC or CCSDS CRC/checksum, then both the HC and DC bits should be ignored.

In transmission, TX descriptors have several fields dedicated to setting up the CRC or checksum to be appended to the message. The core can calculate and insert the RMAP CRC of the header if the HC bit is set. Likewise, the data CRC can be calculated and inserted if the HC bit is set. However, in the case of the data field, three different algorithms are supported, selectable via the Data CRC Type field, *CRCT*: RMAP CRC, CCSDS/CCITT CRC-16 and 16-bit ISO checksum. When the CCSDS CRC/ISO-checksum logic is not available, the CRC-type field is assumed to be RMAP CRC.

The header CRC will be calculated from the data fetched from the header pointer and the data CRC is generated from data fetched from the data pointer. The CRCs are appended after the corresponding fields. The NON-CRC bytes field is set to the number of bytes in the beginning of the header field that should not be included in the CRC calculation, commonly used to exclude path addressing bytes that are discarded before reaching the destination.

When the RMAP target is sending an RMAP reply, it automatically configures the CRC logic so that the RMAP CRC is inserted for both the header and data fields.

74.7 RMAP target

74.7.1 Overview

The Remote Memory Access Protocol (RMAP) is used to implement access to resources in the node via the SpaceWire link. Some common operations are reading and writing to memory, registers and FIFOs. The GRSPFI core has an optional hardware RMAP target adapted to operate over SpaceFibre which can be instantiated by setting the VHDL generic *rmap* to 1. The DMA Capability Register can be read to determine whether the RMAP target is present in the design.

If RMAP is instantiated, every DMA channel has a dedicated RMAP target, and will only check commands received over virtual channels assigned to the DMA channel in question. This allows to enable or disable each RMAP target separately. Likewise, each virtual channel has dedicated configuration registers for the node address and destination key.

When enabled, the RMAP target processes all incoming packets with RMAP protocol ID (second byte set to 0x01) and type field in the instruction set to RMAP command (bits 7:6 of the third byte of

the packet set to 0b01). This is done autonomously without software intervention, meaning that no descriptors are needed. If the RMAP target is disabled or not present, RMAP commands are treated as any other SpaceFibre packet and are therefore stored in memory by using the hardware descriptors.

Once GRSPFI receives an RMAP command, it checks the header to determine if the command is valid or not. The header CRC is first checked; if valid, the command is processed, otherwise it is discarded and no reply is sent under any circumstances.

The next step is to apply the target address and destination key filters. The address filter is similar to the one applied to standard packets and described in 74.4.5; the only difference is that the packet is always passed to the DMA engine in case a reply is requested. If so, the invalid address code is returned in the RMAP status field. Likewise, the destination key of the command is compared with the key set in the Virtual Channel Destination Key register. In case of mismatch an invalid key code is returned if a reply has been requested.

If the header is accepted, the DMA processes the data field of the command. Depending on whether it is a verified or non-verified operation, the RMAP target will wait until the data length and CRC have been verified before initiating any AMBA access. A reply is formatted and sent if the ACK bit of the RMAP command is set.

When a failure occurs during a bus access the error code is set to 1 (General Error). There is predetermined order in which error-codes are set in the case of multiple errors in the core. It is shown in table 1189. Header CRC errors are not included as they do not result in an RMAP reply being sent.

Table 1189. The order of error detection in case of multiple errors. The error detected first has number 1.

Detection Order	Error Code	Error
1	12	Invalid destination logical address
2	3	Invalid destination key
3	2	Unused RMAP packet type or command code
4	9	Verify buffer overrun
5	11	RMW data length error
6	10	Authorization failure
7*	1	General Error (bus master errors during non-verified writes)
8	5/7	Early EOP / EEP (if early)
9	4	Invalid Data CRC
10	6	Cargo Too Large
11	7	EEP
12	1	General Error (bus master errors during verified writes or RMW)
*The bus master error is not guaranteed to be detected before Early EOP/EEP or Invalid Data CRC. For very long accesses the bus error detection might be delayed causing the other two errors to appear first.		

Read accesses are performed on the fly, that is they are not stored in a temporary buffer before transmitting. This means that the error code 1 will never be seen in a read reply since the header has already been sent when the data is read. If the bus master error occurs the packet will be truncated and ended with an EEP.

A verified command is not executed until the complete packet has been received. This means that all errors have been checked before performing any access to the bus, including EEP and cargo being too large.

The specific implementation of the different commands supported by the core is presented in the following subsections. Unsupported commands still defined in the RMAP standard will not be executed and the reply with error code 10 is returned, if requested.

74.7.2 Write commands

The write commands are divided into two subcategories when examining their capabilities: verified writes and non-verified writes. Verified writes have a length restriction of 4 bytes and the address must be aligned to the size. That is 1 byte writes can be done to any address, 2 bytes must be halfword aligned, 3 bytes are not allowed and 4 bytes writes must be word aligned. Since there will always be only one AHB operation performed for each RMAP verified write command the incrementing address bit has no effect and can be set to any value.

Non-verified writes have no restrictions when the incrementing bit is set to 1. If it is set to 0 the number of bytes must be a multiple of 4 and the address word aligned. There is no guarantee how many words will be written when early EOP/EEP is detected for non-verified writes.

74.7.3 Read commands

Read commands are performed on the fly when the reply is sent. Thus if a bus error occurs the packet will be truncated and ended with an EEP, but no error code will reflect this as the header has already been sent. There are no restrictions for incrementing reads but non-incrementing reads have the same alignment restrictions as non-verified writes. Note that the “Authorization failure” error code will be sent in the reply if a violation was detected even if the length field was zero. Also note that no data is sent in the reply if an error was detected i.e. if the status field is non-zero.

74.7.4 RMW commands

All read-modify-write sizes are supported except 6 which would have caused 3 B being read and written on the bus. The RMW bus accesses have the same restrictions as the verified writes. The incrementing address bit shall be set, otherwise the command will not be authorized by the core. The packet is verified and all errors are checked before attempting any AMBA access, as in the case of verified write commands. No data is sent in a reply if an error is detected i.e. the status field is non-zero.

74.7.5 Control

The RMAP target mostly runs in the background without any external intervention, but there are a few control possibilities.

There is an enable bit in the Virtual Channel Control Registers that allows to enable RMAP decoding for a subset of the virtual channels in the system. If all virtual channels assigned to a given DMA channel have RMAP disabled, the RMAP target of that DMA channel will be effectively disabled as well. Any RMAP command received over a virtual channel with RMAP decoding disabled will result in the packet being processed as any other standard packet, i.e. stored via hardware descriptors.

The RMAP target makes use of the Virtual Channel Address Register and the Default Address Register to determine whether a command shall be accepted or not, similarly to the standard DMA channel. The selection between the address registers is done via the Virtual Channel Control Register.

The last control option for the target is the possibility to set the destination key which is found in the Virtual Channel Destination Key register. This allows to configure a different key for every virtual channel.

Table 1190. GRSPFI hardware RMAP handling of different packet type and command fields.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Command	Action
Reserved	Command / Reply	Write / Read	Verify Data	Reply	Increment Address		
0	0	-	-	-	-	RMAP reply	Ignored by the RMAP target. Handled via DMA descriptors.
0	1	0	0	0	0	Not used	Does nothing. No reply is sent.
0	1	0	0	0	1	Not used	Does nothing. No reply is sent.
0	1	0	0	1	0	Read single address	Executed normally. Address has to be word aligned and data size a multiple of four. Reply is sent. If alignment restrictions are violated error code is set to 10.
0	1	0	0	1	1	Read incrementing address.	Executed normally. No restrictions. Reply is sent.
0	1	0	1	0	0	Not used	Does nothing. No reply is sent.
0	1	0	1	0	1	Not used	Does nothing. No reply is sent.
0	1	0	1	1	0	Not used	Does nothing. Reply is sent with error code 2.
0	1	0	1	1	1	Read-Modify-Write	Executed normally. If length is not one of the allowed RMW values nothing is done and error code is set to 11. If the length was correct, alignment restrictions are checked next. 1 byte can be RMW to any address. 2 bytes must be halfword aligned. 3 bytes are not allowed. 4 bytes must be word aligned. If these restrictions are violated nothing is done and error code is set to 10. If an bus error occurs error code is set to 1. Reply is sent.
0	1	1	0	0	0	Write, single-address, do not verify before writing, no reply	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done. No reply is sent.
0	1	1	0	0	1	Write, incrementing address, do not verify before writing, no reply	Executed normally. No restrictions. No reply is sent.
0	1	1	0	1	0	Write, single-address, do not verify before writing, send reply	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done and error code is set to 10. If a bus error occurs error code is set to 1. Reply is sent.

Table 1190. GRSPFI hardware RMAP handling of different packet type and command fields.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Command	Action
Reserved	Command / Reply	Write / Read	Verify Data	Reply	Increment Address		
0	1	1	0	1	1	Write, incrementing address, do not verify before writing, send reply	Executed normally. No restrictions. If bus error occurs error code is set to 1. Reply is sent.
0	1	1	1	0	0	Write, single address, verify before writing, no reply	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restrictions apply as for RMW. If they are violated nothing is done. No reply is sent.
0	1	1	1	0	1	Write, incrementing address, verify before writing, no reply	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restrictions apply as for RMW. If they are violated nothing is done. No reply is sent.
0	1	1	1	1	0	Write, single address, verify before writing, send reply	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for RMW. If they are violated nothing is done and error code is set to 10. If a bus error occurs error code is set to 1. Reply is sent.
0	1	1	1	1	1	Write, incrementing address, verify before writing, send reply	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for RMW. If they are violated nothing is done and error code is set to 10. If a bus error occurs error code is set to 1. Reply is sent.
1	0	-	-	-	-	Unused	Ignored by the RMAP target. Handled via DMA descriptors.
1	1	-	-	-	-	Unused	Ignored by the RMAP target. Handled via DMA descriptors.

74.8 Time-slot generation

SpaceFibre has a built-in mechanism to divide transmission time into time-slots. Each Virtual Channel is only allowed to transmit during certain slots; this can be configured by writing the 2 VC Time-slot Registers (as explained in 74.4.1).

Additionally, the codec needs to be informed about the current time-slot. There are 2 broad approaches when it comes to managing the time-slot generation in GRSPFI:

- External generation. The time-slots are continuously updated by software every time that a change is required. In this case only the Time-Slot Register needs to be written with the value of

the current time-slot. This value is directly passed to the SpaceFibre codec without any further modification.

- Internal generation. The time-slot generation can be enabled, so that the core autonomously increments the time-slot on a periodic basis. The length (in clock cycles) of the time-slot is defined in the Time-slot Length Register; then the internal generation is enabled by setting the Enable Internal Generation bit in the Time-Slot Control Register. The current time-slot can be checked at any time by reading the Time-Slot Register, together with the current value of the internal hardware counter used to update the time-slots. Writing to this register overwrites the current time-slot, but the counter is read-only. Note that this is done in the SpaceFibre codec clock domain, so all calculations shall refer to this clock frequency.

When internal generation is enabled, there is an additional feature to correct drifts in the generation of the time-slots, configured via the Time-Slot Control Register. This is a one-time correction, applied to the next time-slot. The user shall indicate how many clock cycles are to be corrected, the sign (extend or shorten the time-slot) and assert the Valid Adjustment bit so that the adjustment may be applied. The core overwrites the Valid Adjustment bit with 0b when the adjustment has been applied.

74.9 External FIFO interfaces

74.9.1 Functional description

GRSPFI can implement external FIFO interfaces to directly expose the virtual and broadcast channel data to the ports of the IP. This functionality is enabled by two VHDL generics. The number of FIFO interfaces instantiated for virtual channel data is controlled by *numextvc*, whereas the generic *numextbc* determines if an additional FIFO interface is implemented for the broadcast messages. These generics can be configured separately, meaning that the IP may have external virtual channel interfaces without a broadcast channel FIFO port and vice-versa.

When implemented, these FIFO ports allow the user to directly access the data interfaces of the SpaceFibre codec inside GRSPFI, thus bypassing the DMA engine altogether. By default, this functionality is disabled and all virtual channels and the broadcast channel are handled by the DMA engine. To turn it on, the Bypass DMA bit shall be set: VCCTRL.BD for virtual channels, BCCONF.BD for the broadcast channel. For virtual channels, an additional field called VCCTRL.EFC exists to select which FIFO channel number shall be used to forward the data.

The user may dynamically configure these FIFO interfaces, meaning that the DMA bypass can be turned off and on in run time, and a virtual channel can be reassigned to a different FIFO channel. However, it is essential to avoid assigning a specific FIFO channel to more than one virtual channel simultaneously, otherwise the functionality may get compromised. Note that it is possible to expose only a subset of the virtual channels and/or the broadcast channel while still serving the remaining channels with the DMA engine.

To reconfigure this feature in run time it is always recommended to keep both DMA transmitter and receiver off while switching in order to guarantee that ongoing packets are handled properly. Moreover, if the user wants to switch from one FIFO interface to a different one, the IP will automatically forward the virtual channel data to the new FIFO port. It is up to the user to guarantee that the logic connected to GRSPFI is not attempting to write or read from the FIFO ports involved while the switch is taking place.

Please refer to the GRSPFI_CODEC section of this manual for further information regarding the timing of the signals in the external FIFO interfaces.

74.9.2 GRSPFI to GRSPWROUTER bridges

A potential application for the external FIFO interfaces of GRSPFI is the interconnection with other IPs or accelerators with FIFO-like interfaces. A relevant example is the SpaceWire router, thanks to

its optional FIFO ports. The HSSL VHDL package includes 2 bridges for converting SpaceFibre data into SpaceWire and vice-versa: *grspfi_spwdatabr* and *grspfi_spwtcbr*.

The entity *grspfi_spwdatabr* is a bridge that transforms virtual channel data coming from the SpaceFibre codec into SpaceWire data. It instantiates an internal buffer to handle the clock-domain-crossing between the SpaceFibre and SpaceWire IPs and to adapt the bit width of the user data: from SpaceFibre 32-bit words plus 4 control flags to SpaceWire 8-bit characters plus 1 control flag. The bridge also inserts and extracts FILL characters in SpaceFibre traffic when the data length is not multiple of 4 bytes and adjusts the encoding of the EOP and EEP markers on either side.

The data bridge includes input clocks and resets for the SpaceWire/AMBA (*clk*, *rstn*) and SpaceFibre (*hssl_clk*, *hssl_rstn*) clock domains. The rest of signals are included in the records *bi* and *bo*. The bridge can be connected seamlessly to GRSPFI by tying *bi.spfi* and *bo.spfi* to *extvco(i)* and *extvci(i)*, where *i* is the desired external channel number. For the SpaceWire side, the *bi* and *bo* records also include signals with names matching those in the *ri* and *ro* records of the router (save for the *spw_* prefix) that should be connected together without any additional logic in between.

The second entity, *grspfi_spwtcbr*, provides a mechanism to convert SpaceFibre broadcast messages into SpaceWire time-codes. The bridge samples broadcast messages and extracts the relevant byte containing the time-code information, which is passed on to the FIFO port of the SpaceWire router. The IP can optionally filter broadcast messages based on the received broadcast type, in case only a subset of those correspond to time-code information. Likewise, time-codes received by the router are packetized in a broadcast message and forwarded to GRSPFI.

This bridge has the same input clocks and resets as the data bridge. The *bi* and *bo* records are obviously different: *bi.spfi* and *bo.spfi* should be coupled with the GRSPFI ports *extbco* and *extbci*, respectively, whereas *bi.spw_tickout*, *bi.spw_timeout*, *bo.spw_tickin* and *bo.spw_timein* can be directly connected to the FIFO ports of the router to drive the time-code interface.

Moreover, this bridge contains 3 configuration signals for mapping time-codes into broadcast messages:

- *bi.map_bctype[7:0]* and *bi.bcmask[7:0]* are used to filter broadcast messages out based on the broadcast type. The received broadcast type is compared with the value in the configuration register *bi.map_bctype*, except for those bits set to 1 in the filter mask, *bi.bcmask*. In the opposite direction, when a time-code is being converted into a broadcast message, the transmit broadcast type is set to *bi.map_bctype* and the mask is unused.
- *bi.map_bcsel[2:0]* allows to select which byte in the broadcast message contains time-code information. The LSB (bits 7 to 0) is selected when *bi.map_bcsel* is set to 0, while a value of 7 corresponds to the MSB (bits 63 to 56).

Both bridges have the VHDL generics *tech* and *ft* for their internal RAM blocks. The values allowed are the same as for GRSPFI and are therefore described in Table 1231.

74.10 AMBA interfaces

This section explains the AMBA interfaces present in the GRSPFI codec, as well as the AMBA interrupts that the IP may generate, and how to configure them.

74.10.1 Bus master interfaces

Every DMA channel has its own generic bus master interface to fetch and store descriptors, SpaceFibre packets and broadcast frames from/to external memory. The generic approach is chosen for maximum flexibility, so that the user can connect a wrapper adapting the bus master to any custom protocol. Frontgrade Gaisler provides wrapper for AHB 2.0 and AXI4.

The size of the read or write access depends on the operation being performed:

- **Fetching a descriptor**: the descriptor size varies between TX and RX, as explained in 74.4.2 and 74.4.3. In particular, the size of the read access will be of 16 bytes for transmission and of 8 bytes for reception.
- **Storing a SpaceFibre packet** (RX): when the packet is being read from the codec by the DMA, a write access will be performed over the bus master interface as soon as there are at least 256 bytes available or the end of the packet has been detected. This means that the size of the write access may vary between 1 and 256 bytes.
- **Fetching the header of a SpaceFibre packet** (TX): once the descriptor is read and validated, GRSPFI will start fetching the header. This is done with a single burst read access with the size set to the length of the header (maximum 256 bytes).
- **Fetching the data of a SpaceFibre packet** (TX): the data is read with read burst accesses of up to 256 bytes. If there are less than 256 bytes pending, then the size is set to the remaining bytes. Therefore the read size may vary between 1 and 256 bytes.
- **Updating a descriptor**: once the transmission or reception finishes, the descriptor is updated with the status of the transaction. However, only the first word of the descriptor is overwritten: that implies a fixed write access of 4 bytes for both transmission and reception.
- **Storing or fetching a broadcast message**: since the size of a broadcast message is fixed to 16 bytes, as explained in 74.5.4, write and read accesses of broadcast messages will always consist of a burst of 16 bytes.
- **Access triggered by an RMAP command**: The size depends on the type of command and the data length. Verified operations will be completed in a single AMBA access, as the maximum is limited to 4 bytes.

In the particular case of the AHB master interface (i.e. the AHB wrapper is instantiated), the IP has the *hmindex* generic to set the index of the AHB master interface belonging to the first DMA channel. If there are multiple DMA channels, the remaining indexes are assigned consecutively: *hmindex* + 1, *hmindex* + 2, (...) *hmindex* + *num_dmach* - 1.

74.10.2 AHB slave interface

GRSPFI has a single AHB slave interface to access the registers of the core. These registers are used to configure and control the IP. The size of write and read accesses over the AHB interface is always 32 bits.

The IP has 2 major clock domains: the AMBA clock, used by the DMA engine and the AMBA interface, and the SpaceFibre clock, which involves the internal codec and its interface with the DMA FIFOs. Therefore, the internal registers are split into both clock domains. All accesses to registers implemented in the SpaceFibre clock domain are properly handled by means of cross domain crossing techniques.

The register address offsets can help determine if the register is implemented in the AMBA clock domain or in the SpaceFibre clock domain:

- 0x000 - 0x7FF: registers implemented in the SpaceFibre clock domain. These registers are closely related to the SpaceFibre codec and the Virtual Channels: control and status of the codec, configuration of the Virtual Channels and the Broadcast Channel, Quality of Service, timeslot generation, etc.
- 0x800 - 0xFFF: registers implemented in the AMBA clock domain. These registers are related to the DMA channels, interrupt handling and external memory management (i.e. descriptor tables and broadcast channel external buffers).

Due to the internal cross domain crossing techniques, there will be a delay when accessing registers implemented in the SpaceFibre clock domain; waitstates are inserted in order to delay the reply from the internal core to guarantee the integrity of the signals. The AHB interface waits for a maximum of 256 AMBA clock cycles to avoid locking up the AMBA bus if there is no clock running on the codec

side. After that time, the IP simply replies with zeros and sets the TO flag in the SpaceFibre Status Register.

74.10.3 AMBA interrupts

GRSPFI triggers AMBA interrupts upon certain events. The IP is highly configurable when it comes to selecting which events shall produce an interrupt. This section describes how the interrupts are handled and monitored.

The Interrupt Control Register shall be used to configure 3 types of interrupts:

- Broadcast message transmitted.
- Broadcast message received.
- SpaceFibre link events.

These interrupts can only be generated when the Interrupt Enable bit of the Interrupt Control Register is asserted. Additionally, they are individually controlled by means of the BC TX Interrupt bit, BC RX Interrupt bit and Link Interrupt bit. The General Interrupt Status Register provides information regarding the status of these interrupts.

Note that there are several reasons that may trigger a Link interrupt, such as CRC, sequence or protocol errors or a reset triggered by the far-end node. The specific event which has triggered the interrupt can be observed by reading the Lane Layer Status Register and Retry Layer Status Register.

Additionally, every DMA channel can trigger interrupts based on events solely associated to the specific channel. In this case, another 3 events are relevant:

- SpaceFibre packet transmitted by the DMA channel.
- SpaceFibre packet received by the DMA channel.
- Error while reading or writing over the bus master interface of the DMA channel.
- RMAP command received and processed.

These interrupts can be controlled via the DMA Channel IRQ Control Register (one per DMA channel), by writing the TX Interrupt, RX Interrupt, Bus Error Interrupt and RMAP Interrupt bits.

The user can read the DMA Channel Status Register to know which DMA-related event has caused an interrupt. In case it was a bus master error, further information is provided: which Virtual Channel and specific descriptor was being served (not applicable to broadcast messages), as well as a 4-bit code to identify when the bus error occurred (fetching the descriptor, storing the packet, etc.). Please refer to the chapter covering the Register definition to learn how these fields are encoded.

Since there may be multiple DMA channels coexisting in GRSPFI, a register is made available to quickly identify which DMA channels have provoked an interrupt: the DMA Interrupts Status Register, where each position corresponds to a DMA channel. This avoids the process of polling all the DMA channels one by one until finding out which one produced an interrupt.

74.11 Implementation

74.11.1 Reset

The core changes its reset behavior depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

GRSPFI will feature a synchronous reset if the parameter *grlib_async_reset_enable* is not set in the GRLIB configuration package. Otherwise, it will implement an asynchronous reset if *grlib_async_reset_enable* is set.

74.11.2 Endianness

The core automatically changes its endianness behavior depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for both big-endian and little-endian systems.

74.11.3 Clocking scheme

As explained throughout this section, there are up to 4 clock domains in the IP. These paragraphs describe their use and how they shall be generated and connected.

- AMBA clock (clk): the DMA and AMBA layers of the IP operate in this clock domain. It shall be connected to the AMBA clock of the system where the IP is instantiated.
- SpaceFibre clock (spfi_clk): used by the core logic, including the SpFi codec and the interface between the DMA FIFOs and the codec, the timeslot generation logic and the address filters. Some AHB registers are also implemented in this domain. This clock is normally generated from the SerDes output transmitter clock by using a Clock Conditioning Circuit. The target frequency shall be the SpaceFibre bit-rate divided by 40 (i.e. assuming a 40-bit SerDes interface). This is dependent on the specific SerDes device used. Example: if the link operates at 2.5 Gbps, the target frequency shall always be 62.5 MHz. If the SerDes transmitter clock is 125 MHz, then the multiplication factor of the CCC shall be 0.5.
- SpaceFibre transmitter clock (spfi_txclk): this clock is used by the transmitter logic within the codec if using a narrow SerDes interface (16/20 bits; both generics *use_sep_txclk* and *sel_16_20_bit_mode* set to 1). The target frequency shall be the SpaceFibre bit-rate divided by 20 (i.e. a 20-bit SerDes interface, twice the frequency of the SpaceFibre clock). Is is normally generated from the SerDes output transmitter clock. In the example above, the frequency of this clock shall be 125 MHz and could be generated from the SerDes TX clock without multiplication factor.
- SpaceFibre RX recovered clock (spfii.rx_clk): if the data received from the SerDes is not synchronous to the SpaceFibre clock, the SerDes recovered RX clock shall be connected to this input. If the data is already synchronous, for instance because the SerDes comprises its own elastic buffer, this clock shall be connected to the SpaceFibre clock directly.

Any exchange of signals between clock domains, especially between the AMBA and SpaceFibre clocks, is protected by clock domain crossing techniques in order to avoid metastability problems in the IP.

Please note that the SpaceFibre clock and the AMBA clock have separate reset signals; the user shall ensure that these resets are properly synchronized with their respective clocks if a synchronous reset scheme is chosen.

74.11.4 Relation between GRSPFI and GRHSSL

GRHSSL is a High Speed Serial Link controller providing support for both SpaceFibre and WizardLink. The IP can be configured to implement either or both codecs by means of VHDL generics. When both protocols are implemented, the user shall select which codec is active at a time via AHB registers. The selected core will be in charge of driving the SerDes and bus master interfaces.

GRSPFI internally uses the GRHSSL IP and configures it so that only SpaceFibre is instantiated. Generics relevant to SpaceFibre are exposed in the top-level entity, whereas those related to WizardLink are tied to predefined values in order to guarantee that the WizardLink layer is not present in the design.

74.12 Registers

The core is programmed through registers mapped into AHB address space. Only 32-bit single accesses to the registers are supported. Registers within the range 0x000-0x7FF are implemented in the SpaceFibre clock domain, whereas the range 0x800-0xFFF is related to the AMBA/DMA layer.

Table 1191. GRSPFI registers

AHB address offset	Register
0x000	Codec General Capabilities Register
0x004	Codec Buffers Capabilities Register
0x008	Codec Control Register
0x00C	Lane Layer Status Register
0x010	Retry Layer Status Register
0x014	Default Address Register
0x020	Time-slot Register
0x024	Time-slot Control Register
0x028	Time-slot Length Register
0x030	Broadcast Channel Configuration Register
0x100, 0x120, 0x140... 0x4E0	VC Control Register (VC channels 1 - 32)[1)]
0x104, 0x124, 0x144... 0x4E4	VC Status Register (VC channels 1 - 32)[1)]
0x108, 0x128, 0x148... 0x4E8	VC RX Max Length Register (VC channels 1 - 32)[1)]
0x10C, 0x12C, 0x14C... 0x4EC	VC Address Register (VC channels 1 - 32)[1)]
0x110, 0x130, 0x150... 0x4F0	VC Timeslot 1 Register (VC channels 1 - 32)[1)]
0x114, 0x134, 0x154... 0x4F4	VC Timeslot 2 Register (VC channels 1 - 32)[1)]
0x118, 0x138, 0x158... 0x4F8	VC Destination Key Register (VC channels 1 - 32)[1)]
0x800	DMA Layer Capabilities Register
0x804	DMA Layer Control Register
0x808	DMA Layer Status Register
0x80C	SpaceFibre Enable Register
0x810	Broadcast Channel TX Address Register
0x814	Broadcast Channel TX Size Register
0x818	Broadcast Channel TX Write Pointer Register
0x81C	Broadcast Channel TX Read Pointer Register
0x820	Broadcast Channel RX Address Register
0x824	Broadcast Channel RX Size Register
0x828	Broadcast Channel RX Write Pointer Register
0x82C	Broadcast Channel RX Read Pointer Register
0x830	Broadcast Channel Mapping Register
0x900, 0x910, 0x920... 0xAF0	VC TX Descriptor Table Address Register (VC channels 1 - 32)[1)]
0x904, 0x914, 0x924... 0xAF4	VC RX Descriptor Table Address Register (VC channels 1 - 32)[1)]
0x908, 0x918, 0x928... 0xAF8	VC Descriptor Control Register (VC channels 1 - 32)[1)]
0x90C, 0x91C, 0x92C... 0xAFC	VC Statistics Register (VC channels 1 - 32)[1)]
0xB00, 0xB10, 0xB20... 0xB70	DMA Channel Control Register (DMA channels 1 - 8)[1)]
0xB04, 0xB14, 0xB24... 0xB74	DMA Channel Status Register (DMA channels 1 - 8)[1)]
0xB08, 0xB18, 0xB28... 0xB78	DMA Channel VC Mapping Register (DMA channels 1 - 8)[1)]

Table 1191. GRSPFI registers

AHB address offset	Register
0xB0C, 0xB1C, 0xB2C... 0xB7C	DMA Channel IRQ Control Register (DMA channels 1 - 8)[1)]
0xB80, 0xB90, 0xBA0... 0xBF0	DMA Channel Extended Status Register (DMA channels 1 - 8)[1)]

Note 1: Registers for non implemented VC and DMA channels are reserved. The number of available VC and DMA channels is indicated in the General Capabilities Register and DMA Capabilities Register, respectively.

74.12.1 Codec General Capabilities Register

Table 1192. 0x000 - GENCAP - Codec General Capabilities Register

31 29	28	27 22	21 17	16
Reserved	NEB	NEV	WBWC	RFC
000	-	-	-	-
r	r	r	r	r

12	11 9	8 4	3	2	1	0
RFC	FCTM	NVC	PI	BM	IED	SCK
-	-	-	-	-	-	-
r	r	r	r	r	r	r

28 Number of External Broadcast channels (NEB). Value determined by the VHDL generic numextbc.

27: 22 Number of External Virtual channels (NEV). Value determined by the VHDL generic numextvc.

21: 17 Width of the bandwidth credit counter (WBWC) encoded as number of bits - 1. Value determined by the VHDL generic *width_bw_credit*.

16: 12 Width of the remote FCT counter (RFC) encoded as number of bits - 1. Value determined by the VHDL generic *remote_fct_cnt_max*.

11: 9 FCT multiplier (FCTM), encoded as the value of the multiplier - 1 (from 0 to 7). Value determined by the VHDL generic *fct_multiplier*.

8: 4 Number of Virtual Channels (NVC) encoded as the number of channels - 1 (from 0 to 31). Value determined by the VHDL generic *num_vc*.

3 PRBS INIT1 (PI). If set, INIT1 sequence during lane initialisation is embedded into a stream of pseudo-random numbers. Determined by the VHDL generic *prbs_init1*.

2 16/20 Bit Mode Selector (BM). If set, the SerDes interface is 16+2 bit (without 8b10b) or 20 bit (with 8b10b) wide instead of 36/40 bit. Determined by the VHDL generic *sel_16_20_bit_mode*.

1 Internal 8b10b encoder/decoder (IED). If set, internal 8B10B encoding and decoding is activated. Determined by the VHDL generic *use_8b10b*.

0 Use separate clock for TX (SCK). If set, the SerDes transmission clock is decoupled from the SpaceFibre port clock. An additional transmit buffer is instantiated for this reason. Determined by the VHDL generic *use_sep_txclk*.

74.12.2 Codec Buffers Capabilities Register

Table 1193. 0x004 - BUFCAP - Codec Buffers Capabilities Register

31 25	24 20	19
Reserved	VT	VR
00	-	-
r	r	r

15	14 10	9 5	4 0
VR	RTB	RTF	RTD
-	-	-	-
r	r	r	r

24: 20 Depth of the VC TX buffer (VT). Encoded as depth (address bits) - 1. Value determined by the VHDL generic *depth_vc_tx_buf*.

19: 15 Depth of the VC RX buffer (VR). Encoded as depth (address bits) - 1. Value determined by the VHDL generic *depth_vc_rx_buf*.

14: 10 Depth of the Broadcast retry buffer (RTB). Encoded as depth (address bits) - 1. Value determined by the VHDL generic *depth_rbuf_bc*.

9: 5 Depth of the FCT retry buffer (RTF). Encoded as depth (address bits) - 1. Value determined by the VHDL generic *depth_rbuf_fct*.

4: 0 Depth of the Data retry buffer (RTD). Encoded as depth (address bits) - 1. Value determined by the VHDL generic *depth_rbuf_data*.

74.12.3 Codec Control Register

Table 1194. 0x008 - CCTRL - Codec Control Register

31															16
Reserved															
0x0000															
r															
15	14	13	12	11					4	3	2	1	0		
Reserved		SCR	LB	SBR						AS	LS	LNR	LIR		
00		0	0	0x00						0	0	0	0		
r		rw	rw	rw						rw	rw	rw	rw		

13 Scramble Enable (SCR). The retry layer comprises a data scrambler, which can be enabled/disabled with this bit. The data scrambler becomes directly active right after asserting this flag. This will lead to continuous data retries if the de-scrambler of the remote node is not activated. Therefore, a lane reset must be triggered after asserting/de-asserting this flag. By doing so, the remote node is informed during the subsequent handshake if the data is scrambled or not and will then switch on or off the de-scrambler accordingly.

12 Loop-back mode (LB). The SpaceFibre codec has an internal loop-back feature that can be enabled / disabled with this bit. The codec must be reset after asserting/de-asserting this flag.

11: 4 Standby Reason (SBR). Byte to be included in the SpaceFibre STANDBY control words with the reason why the STANDBY control was sent or any other status information.

3 Auto-Start (AS). When asserted, the lane layer will passively listen to incoming words from the far end and start the handshake procedure once INIT1 words are received.

2 Lane Start (LS). When asserted, the lane layer will actively try to build up a connection with the far end by sending out INIT1 words.

1 Lane Reset (LNR). The lane can be reset at any time by asserting this bit. In contrast to link resets, a lane reset is only reinitialising the lane. Therefore, ongoing transmissions are probably delayed but never cancelled.

0 Link Reset (LIR). The SpaceFibre link can be reset by asserting this bit. As a consequence, outbound packets are spilled up to and including the next EOP and inbound packets are terminated by an EEP.

74.12.4 Lane Layer Status Register

Table 1195. 0x00C - LLSTAT - Lane Layer Status Register

31							24	23			20	19			
Reserved								FEC				REC			
0x00								0x0				0x00			
r								r				r			
		12	11	10	9	8	7	6	5	4	3				0
REC			REO	RXP	TO	FELC		FEL	FES	FER	LSTS				
0x00			0	0	0	00		0	0	0	0x1				
r			wc	r	wc	r		wc	wc	wc	r				

23: 20 Far End Capabilities (FEC). Capability field sent by the far-end node in an INIT3 control word.

19: 12 RX Error words Count (REC). 8-bit wide counter for received RXERR words. The counter is automatically decreased by one every time 16 384 more words have been received.

11 RX Error Count Overflow (REO). Set when the RXERR word counter has overflown.

10	RX Polarity (RXP). Set when the lane layer detects inverted receive polarity during handshake. If internal 8b10b decoding is activated, the polarity of the incoming bitstream is automatically inverted.
9	Timeout (TO). Set when a connection timeout occurs during lane initialization.
8: 7	Far-end Loss of Signal Cause (FELC). Reason field received inside a Lost Signal control word.
6	Far-end Loss of Signal (FEL). Set if a new handshake has started due to a loss of signal at the SerDes receiver.
5	Far-end Standby (FES). Set when STANDBY control words are received. This is always the case when the link initialization state machine in the far end node is deactivated.
4	Far-end Link Reset (FER). Set if a link reset was triggered due to a system reset in the far end node.
3: 0	Lane State (LSTS). 4-bit wide unsigned value encoding the current lane initialization state as follows: 0x1: Clear Line 0x2: Disabled 0x3: Wait 0x4: Started 0x5: Invert RX Polarity 0x6: Connecting 0x7: Connected 0x8: Active 0x9: Loss of Signal 0xA: Prepare Standby

74.12.5 Retry Layer Status Register

Table 1196. 0x010 - RLSTAT - Retry Layer Status Register

31										16
Reserved										
0x0000										
r										

15	11	10	9	8	7	4	3	2	1	0
Reserved		RBE	PE	TME	RTC		SE	FE	C8E	CE
00000		1	0	0	0x0		0	0	0	0
r		r	wc	wc	r		wc	wc	wc	wc

10	Retry Buffer Empty (RBE). Set when the error recovery buffer is empty, i.e. all data frames, broadcast frames and FCT words have been sent and acknowledged.
9	Protocol Error (PE). Set when an ACK / NACK error has been detected.
8	Too Many Errors (TME). Set when the RETRY counter has overflown (i.e. the link is automatically reset).
7: 4	Retry Count (RTC). 4-bit wide counter that increments for every RETRY event initiated. It is automatically reset when an ACK word is received from the far end. The SpaceFibre link is automatically reset when the counter reaches 16.
3	Sequence Error (SE). Set when a sequence number error occurred during the reception of data frames, broadcast frames, idle frames, FCT or FULL words.
2	Frame Error (FE). Set when an invalid data, broadcast, or idle frame is received.
1	CRC-8 Error (C8E). Set when a CRC error has occurred during the reception of a control word or broadcast frame.
0	CRC-16 Error (CE). Set when a CRC error has occurred during the reception of a data frame.

74.12.6 Default Address Register

Table 1197. 0x014 - DEFADDR - Default Address Register

31	16
Reserved	
0x0000	
r	

15	8	7	0
DEFMSK		DEFADDR	
0x00		0x00	
rw		rw	

15: 8 Default Mask (DEFMSK). Mask used in the default address filter. When a bit of the mask is set, the corresponding bit of the SpaceFibre address is not compared with the Default Address. The default address filter is applied to the first address byte received of a SpaceFibre packet when the VC-specific address filter is not enabled.

7: 0 Default Address (DEFADDR). Default address compared with the masked SpaceFibre address byte. The default address filter is applied when the VC-specific address filter is not enabled.

74.12.7 Time-slot Register

Table 1198. 0x020 - TS - Time-slot Register

31	30	29
Reserved	CNT	
00	0x000000	
r	r	

8	7	6	5	0
CNT		Reserved	TS	
0x000000		00	0x00	
r		r	rw	

29: 8 Time-slot internal counter (CNT). Hardware counter used to update the value of the current time-slot if the internal generation is enabled in the Time-slot Control Register. When it matches the nominal length plus/minus the adjustment programmed by the user, the time-slot is incremented and the internal counter resets to 0. It is expressed in number of SpaceFibre clock cycles. Read-only.

5: 0 Time-Slot (TS). Current time-slot used to schedule the transmission over the SpaceFibre link among the Virtual Channels. If the internal time-slot generation is enabled, this value is automatically incremented by the core. In this case, the register may still be written in order to modify the value of the current time-slot. If the internal generation is disabled, this register must be written periodically by software in order to update the time-slot.

74.12.8 Time-slot Control Register

Table 1199. 0x024 - TSCTRL - Time-slot Control Register

31	16
Reserved	
0x0000	
r	

15	14	13	4	3	2	1	0
Reserved		ADJ		AS	AV	Res	EG
00		0x000		0	0	0	0
r		rw		rw	rw	r	rw

13: 4 Adjustment (ADJ). Unsigned value of the adjustment to be applied to the length of the next time-slot. It represents the number of clock cycles that need to be corrected with respect to the nominal value defined in the Time-slot Length Register. The Adjustment Sign determines if the time-slot shall be extended (AS = 0b) or shortened (AS = 1b). This correction is applied only for the next time-slot if the Adjustment Valid bit is set (AV = 1b). When shortening the time-slot, this value must be smaller than the Time-slot Nominal Length.

3 Adjustment Sign (AS). Determines if the next time-slot shall be extended (0b) or shortened (1b).

2 Adjustment Valid (AV). If set, GRSPFI will apply the adjustment specified in ADJ and AS. This is only done once; after that, the core sets this bit to 0b, in order to indicate that the correction has already been applied.

0 Enable Internal Generation of Time-slots (EG). If set to 1b, GRSPFI will autonomously increment the value of the time-slot: a counter will be compared with the valued defined in the Time-slot Length Register; once it reaches that value, the time-slot is updated. If set to 0b, the value of the time-slot can only be updated by writing directly to the Time-slot Register.

74.12.9 Time-slot Length Register

Table 1200. 0x028 - TSLEN - Time-slot Length Register

31	21	20	0
Reserved		NOMLEN	
0x000		0x000000	
r		rw	

NOMLEN
0x000000
rw

20: 0 Time-slot Nominal Length (NOMLEN), expressed in number of clock cycles - 1. When the internal generation of time-slots is enabled, the IP will update the value of the time-slot when the internal counter reaches the value defined in this field.

74.12.10 Broadcast Channel Configuration Register

Table 1201. 0x030 - BCCONF - Broadcast Channel Configuration Register

31															16
BW															
0x0000															
rw															

15							9	8	7						0
Reserved								BD	CHN						
0x00								0	0x00						
r								rw	rw						

31: 16 Broadcast Channel Bandwidth (BW). For information regarding the encoding of this field, please refer to 74.5.1.

8 Bypass DMA Engine (BD). When this bit is set, broadcast messages are directly exposed via the external FIFO ports of the IP (EXTBCI, EXTBCO), thus bypassing the DMA engine. If not set, broadcast messages are handled by the DMA engine by means of circular buffers. This bit is only available if the generic *numextbc* is set to 1.

7: 0 Broadcast Channel Number (CHN). When a broadcast message is transmitted by GRSPFI, it will use the value defined in this field as the Broadcast Channel number.

74.12.11 Virtual Channel Control Register

Table 1202. 0x100, 0x120, 0x140... 0x4E0 - VCCTRL - Virtual Channel Control Register

31															16
VBW															
0x0000															
rw															

15				12	11		9	8			4	3	2	1	0
PR					Reserved			EFC				BD	Res	RE	AE
0x0					000			0x0				0	0	0	0
rw					r			rw				rw	r	rw	rw

31: 16 Virtual Channel Bandwidth (VBW). Bandwidth allocated to this Virtual Channel. For information regarding the encoding of this field refer to 74.4.1.

15: 12 Priority (PR). Establishes the priority of the Virtual Channel from 0 to 15, 0 being the highest priority and 15 the lowest. Used by the codec to guarantee the QoS over the SpaceFibre link.

8: 4 External FIFO Channel (EFC). Select the FIFO channel, ranging from 0 to *numextvc* - 1, that will handle this virtual channel when the VCCTRL.BD bit is set. This field is only available if the generic *numextvc* is not 0.

3 Bypass DMA Engine (BD). When this bit is set, packets for this virtual channel are directly exposed via the external FIFO ports of the IP (EXTVCI, EXTVCO), thus bypassing the DMA engine. If not set, they are handled by the DMA engine by means of hardware descriptors. This bit is only available if the generic *numextvc* is not 0.

1 RMAP Enable (RE). If set to 1b, RMAP commands received over this virtual channel will be handled by the RMAP target, otherwise they will be stored in memory via descriptors. Only used when the generic *rmap* is set.

0 Virtual Channel Address Enable (AE). If set to 1b, SpaceFibre packets received over this Virtual Channel will be compared with the address (and mask) defined in the VC Address Register in order to accept or filtered them out. If set to 0b, the default address filter is applied instead.

74.12.12 Virtual Channel Status Register

Table 1203. 0x104, 0x124, 0x144... 0x4E4 - VCSTAT - Virtual Channel Status Register

31	16
Reserved	
0x0000	
r	

15	5	4	3	2	1	0
Reserved		FCO	IBO	DHC	BUU	BOU
0x000		0	0	0	0	0
r		wc	r	r	r	r

4 FCT Counter Overflow (FCO). Set when the corresponding FCT credit counter has overflowed. This indicates that the width of the FCT counter, defined by *remote_fct_cnt_max*, is too small.

3 Input Buffer Overrun (IBO). Set when this Virtual Channel input buffer is receiving data when it is full. This indicates a fatal protocol error.

2 Destination Has Credit (DHC). Set when there is space in the Virtual Channel input buffer of the far end node.

1 Bandwidth Under-use (BUU). Indicates that this Virtual Channel uses less bandwidth than expected. Note that this flag is also set when the Virtual Channel is deactivated due to an expected bandwidth value of 0, even if the user application tries to transmit data over this Virtual Channel.

0 Bandwidth Over-use (BOU). Set when the Virtual Channel uses more bandwidth than expected, i.e. when the bandwidth overuse mechanism is active.

74.12.13 Virtual Channel RX Max Length Register

Table 1204. 0x108, 0x128, 0x148... 0x4E8 - VCMAXLEN - Virtual Channel RX Max Length Register

31	25	24
Reserved		MAXLGTH
0x00		0x000000
r		rw

0
MAXLGTH
0x000000
rw

24: 0 Virtual Channel RX Maximum Length (MAXLGTH). Maximum number of bytes accepted for SpaceFibre packets received over this Virtual Channel. When the length of the packet exceeds this value, the rest of the packet is spilled and the Truncated bit is set when updating the RX descriptor. Not applicable when the RMAP target is processing an RMAP command.

74.12.14 Virtual Channel Address Register

Table 1205. 0x10C, 0x12C, 0x14C... 0x4EC - VCADDR - Virtual Channel Address Register

31	16
Reserved	
0x0000	
r	

15	8	7	0
VCMSK		VCADDR	
0x00		0x00	
rw		rw	

15: 8 Virtual Channel Mask (VCMSK). Mask used in the VC address filter. When a bit of the mask is set, the corresponding bit of the SpaceFibre address is not compared with the VC Address. The VC address filter is applied to the first address byte received of a SpaceFibre packet when the Address Enable bit is asserted in the VC Control Register.

7: 0 Virtual Channel Address (VCADDR). VC address compared with the masked SpaceFibre address byte. The VC address filter is applied when the Address Enable bit is asserted in the VC Control Register.

74.12.15 Virtual Channel Time-slot 1 Register

Table 1206. 0x110, 0x130, 0x150... 0x4F0 - VCTS1 - Virtual Channel Time-slot 1 Register

31	0
TSLOT1	
0x00000000	
rw	

31: 0 Virtual Channel Time-slot 1 (TSLOT1). This register specifies during which time-slots this Virtual Channel is allowed to transmit. It corresponds to the time-slots 31 (MSB) to 0 (LSB).

74.12.16 Virtual Channel Time-slot 2 Register

Table 1207. 0x114, 0x134, 0x154... 0x4F4 - VCTS2 - Virtual Channel Time-slot 2 Register

31	0
TSLOT2	
0x00000000	
rw	

31: 0 Virtual Channel Time-slot 2 (TSLOT2). This register specifies during which time-slots this Virtual Channel is allowed to transmit. It corresponds to the time-slots 63 (MSB) to 32 (LSB).

74.12.17 Virtual Channel Destination Key Register

Table 1208. 0x118, 0x138, 0x158... 0x4F8 - VCKEY - Virtual Channel Destination Key Register

31	16
Reserved	
0x0000	
r	

15	8	7	0
Reserved		KEY	
0x00		0x00	
r		rw	

7: 0 Virtual Channel Key (KEY). This register specifies the destination key of the virtual channel to compare with the key in the RMAP command received. Only used if the generic *rmap* is set.

74.12.18DMA layer Capabilities Register

Table 1209. 0x800 - DLCAP - DMA Layer Capabilities Register

31	30	16
SPFI	Reserved	
1	0x0000	
r	r	

15	11	10	9	8	7	6	4	3	2	1	0
Reserved		CC	RC	RT	Res	NDMA		NTXD		NRXD	
0x00		-	-	-	0	-		-		-	
r		r	r	r	r	r		r		r	

31 SpaceFibre protocol implemented (SPFI). Read-only bit to indicate that GRSPFI is present in the design.

10 CCSDS CRC logic implemented (CC). Value determined by the VHDL generic *ccsdscrc*.

9 RMAP CRC logic implemented (RC). Value determined by the VHDL generic *rmapcrc*.

8 RMAP Target implemented (RT). Value determined by the VHDL generic *rmap*.

6: 4 Number of DMA channels (NDMA) encoded as the number of channels - 1 (from 0 to 7). It is determined by the value of the VHDL generic *num_dmach*.

3: 2 Number of TX descriptors (NTXD). It shows the size of the TX descriptor tables. Value determined by the VHDL generic *num_txdesc*. Encoding:
- 00b: 64 descriptors
- 01b: 128 descriptors
- 10b: 256 descriptors
- 11b: 512 descriptors

1: 0 Number of RX descriptors (NRXD). It shows the size of the RX descriptor tables. Value determined by the VHDL generic *num_rxdesc*. Encoding:
- 00b: 128 descriptors
- 01b: 256 descriptors
- 10b: 512 descriptors
- 11b: 1024 descriptors

74.12.19DMA Layer Control Register

Table 1210. 0x804 - DLCTRL - DMA Layer Control Register

31	16
Reserved	
0x0000	
r	

15	3	2	1	0
Reserved		BTE	BRE	LE
0x0000		0	0	0
r		rw	rw	rw

2 Broadcast TX Interrupt Enable (BTE). Enables interrupt generation when a complete broadcast message is fetched from external memory and passed on to the codec for transmission.

1 Broadcast RX Interrupt Enable (BRE). Enables interrupt generation when a broadcast message is received and stored in external memory.

0 Link Interrupt Enable (LE). Enables interrupt generation when a link event is detected by the codec. Several situations may lead to a link event; for the specific reason, the codec status registers can be read.

74.12.20DMA Layer Status Register

Table 1211. 0x808 - DLSTAT - DMA Layer Status Register

31	x+1	x	16
Reserved		DCI	
0x0000		0x000	
r		wc	

15	4	3	2	1	0
Reserved		TO	BT	BR	LE
0x0000		0	0	0	0
r		wc	wc	wc	wc

x: 16 DMA Channel Interrupt (DCI). Set when the core has generated an interrupt associated with a DMA channel. There is one status bit per DMA channel. The specific cause can be obtained by reading the corresponding DMA Channel Status Register. The value of *x* is calculated as *num_dmach* + 15.

3 AHB Register Timeout (TO). Set when the DMA engine has reached the internal timeout (256 AMBA clock cycles) while waiting for a reply when accessing the AHB registers implemented on the codec clock domain.

2 Broadcast Transmission (BT). Set when a broadcast message has been transmitted successfully. This implies that the message has been fetched from external memory and passed on to the internal codec.

1 Broadcast Reception (BR). Set when a broadcast message has been received successfully. This implies that the message has been received by the codec, passed to the DMA layer and stored in external memory.

0 Link event (LE). Set when an event has occurred in the SpaceFibre Link. The specific event can be checked by reading the Lane Layer Status Register and especially the Retry Layer Status Register.

74.12.21 SpaceFibre Enable Register

Table 1212. 0x80C - SPFIEN - SpaceFibre Enable Register

31																															1	0
Reserved																															EN	
0x0000																															1	
r																															rw	

0 SpaceFibre Enable (EN). This bit is used to enable the SpaceFibre mode of operation if GRSPFI is implemented together with other protocols as part of the GRHSSL IP. If SpaceFibre is the only protocol implemented, this bit will read as 1b and writing will have no effect.

74.12.22 Broadcast Channel TX Address Register

Table 1213. 0x810 - BCTXADDR - Broadcast Channel TX Address Register

31																															10	9										0
TXADDR																															Reserved											
0x000000																															0x000											
rw																															r											

31: 10 Broadcast Channel TX Buffer Address (TXADDR). Sets the base address of the transmit buffer where broadcast messages shall be fetched from before being transmitted.

74.12.23 Broadcast Channel TX Size Register

Table 1214. 0x814 - BCTXSIZE - Broadcast Channel TX Size Register

| 31 | | | | | | | | | | | | | 20 | 19 | | | | | | | | | | | | | | | | | | |

74.12.24 Broadcast Channel TX Write Pointer Register

Table 1215. 0x818 - BCTXWPTR - Broadcast Channel TX Write Pointer Register

31	20	19	0
Reserved		WRPTR	
0x000		0x0000	
r		rw	

4	3	0
WRPTR	Reserved	
0x0000	0x0	
rw	r	

19: 4 Broadcast Channel TX Write Pointer (WRPTR). Used to indicate GRSPFI that there are new broadcast messages to transmit. It shall be handled by software, and it shall point to the position + 1 of the last broadcast message written to the transmit buffer. It operates as a relative address, the base address being defined in the Broadcast Channel TX Address Register. Note that at least one position of the buffer shall be empty at all times.

74.12.25 Broadcast Channel TX Read Pointer Register

Table 1216. 0x81C - BCTXRPTR - Broadcast Channel TX Read Pointer Register

31	20	19	0
Reserved		RDPTR	
0x000		0x0000	
r		rw	

4	3	0
RDPTR	Reserved	
0x0000	0x0	
rw	r	

19: 4 Broadcast Channel TX Read Pointer (RDPTR). The core updates the read pointer when a new broadcast message has been fetched from the transmit buffer; it shall point to the position + 1 of the last message read. If the write and read pointers are equal, the transmit buffer is empty (i.e. the core has already fetched all the messages from the buffer). It operates as a relative address, the base address being defined in the Broadcast Channel TX Address Register. This register can still be written by software if the user wants to modify the sequence of messages, but this must be done when the DMA channel managing the Broadcast Channel is not active to guarantee the proper functionality of the channel.

74.12.26 Broadcast Channel RX Address Register

Table 1217. 0x820 - BCRXADDR - Broadcast Channel RX Address Register

31	0
RXADDR	
0x000000	
rw	

10	9	0
RXADDR	Reserved	
0x000000	0x000	
rw	r	

31: 10 Broadcast Channel RX Buffer Address (RXADDR). Sets the base address of the receive buffer where broadcast messages shall be written to once received.

74.12.27 Broadcast Channel RX Size Register

Table 1218. 0x824 - BCRXSIZE - Broadcast Channel RX Size Register

31	20	19	0
Reserved		SIZE	
0x000		0x0000	
r		rw	

4	3	0
SIZE	Reserved	
0x0000	0x0	
rw	r	

19: 4 Broadcast Channel RX Buffer Size (SIZE). Encoded as number of broadcast messages - 1, it defines the size of the Broadcast RX buffer. When updating a hardware pointer which equals the value defined in this field, the pointer is set to zero (i.e. pointing to the base address of the receive buffer). Otherwise, if the pointer is lower than the last position, it is incremented by 1.

74.12.28 Broadcast Channel RX Write Pointer Register

Table 1219. 0x828 - BCRXWPTR - Broadcast Channel RX Write Pointer Register

31	20	19	0
Reserved		WRPTR	
0x000		0x0000	
r		rw	

4	3	0
WRPTR	Reserved	
0x0000	0x0	
rw	r	

19: 4 Broadcast Channel RX Write Pointer (WRPTR). The core updates the write pointer when a new broadcast message has been written to the receive buffer; it shall point to the position + 1 of the last message written. It operates as a relative address, the base address being defined in the Broadcast Channel RX Address Register. Note that at least one position of the buffer shall be left empty at all times, so GRSPFI will not write any more messages until that condition is met. This register can still be written by software if the user wants to modify the sequence of messages, but this must be done when the DMA channel managing the Broadcast Channel is not active to guarantee the proper functionality of the channel.

74.12.29 Broadcast Channel RX Read Pointer Register

Table 1220. 0x82C - BCRXRPTR - Broadcast Channel RX Read Pointer Register

31	20	19	0
Reserved		RDPTR	
0x000		0x0000	
r		rw	

4	3	0
RDPTR	Reserved	
0x0000	0x0	
rw	r	

19: 4 Broadcast Channel RX Read Pointer (RDPTR). Used to indicate GRSPFI which broadcast messages have been read from the receive buffer, in order to make room for new messages. It shall be handled by software, and it shall point to the position + 1 of the last broadcast message read from the receive buffer. If the write and read

pointers are equal, the receive buffer is empty (i.e. all messages have already been read and may be overwritten). It operates as a relative address, the base address being defined in the Broadcast Channel RX Address Register.

74.12.30 Broadcast Channel Mapping Register

Table 1221. 0x830 - BCMAP - Broadcast Channel Mapping Register

31	16
Reserved	
0x0000	
r	

15	5	4	0
Reserved		MAP	
0x000		0x00	
r		rw	

4: 0 Broadcast Channel Mapping (MAP). It selects which DMA channel is in charge of handling broadcast messages (both transmission and reception). It is encoded as the number of the DMA channel, from 0 to 31. It must be set to an implemented DMA channel. It is assigned to the DMA channel 0 by default.

74.12.31 VC TX Descriptor Table Address Register

Table 1222. 0x900, 0x910, 0x920... 0xAF0 - VCTXDADDR - VC TX Descriptor Table Address Register

31	x+1
TXADDR	
All 0b	
rw	

x	4	3	2	1	0
TXDESCSEL		Reserved			
All 0b		0x0			
rw		r			

31: x+1 TX Descriptor Table Address (TXADDR). Sets the base address of the descriptor table. The number of bits of this field depends on the size of the DMA transmit descriptor table, determined by the generic *num_txdesc*. The value of x is calculated as follows: $x = log_2(num_txdesc) + 3$.

x: 4 TX Descriptor Selector (TXDESCSEL). Offset within the descriptor table. Shows which descriptor is currently used by the core. For each new descriptor read, the selector will increase by 16 and eventually wrap to zero again. The number of bits of this field depends on the size of the DMA transmit descriptor table, determined by the generic *num_txdesc*. The value of x is calculated as follows: $x = log_2(num_txdesc) + 3$.

74.12.32 VC RX Descriptor Table Address Register

Table 1223. 0x904, 0x914, 0x924... 0xAF4 - VCRXDADDR - VC RX Descriptor Table Address Register

31	x+1
RXADDR	
All 0b	
rw	

x	3	2	1	0
RXDESCSEL		Reserved		
All 0b		0x0		
rw		r		

31: x+1 RX Descriptor Table Address (RXADDR). Sets the base address of the descriptor table. The number of bits of this field depends on the size of the DMA receive descriptor table, determined by the generic *num_rxdesc*. The value of x is calculated as follows: $x = log_2(num_rxdesc) + 2$.

x: 3 RX Descriptor Selector (RXDESCSEL). Offset within the descriptor table. Shows which descriptor is currently used by the core. For each new descriptor read, the selector will increase by 8 and eventually wrap to zero again. The number of bits of this field depends on the size of the DMA receive descriptor table, determined by the generic *num_rxdesc*. The value of x is calculated as follows: $x = log_2(num_rxdesc) + 2$.

74.12.33 VC Descriptor Control Register

Table 1224. 0x908, 0x918, 0x928... 0xAF8 - VCDCTRL - VC Descriptor Control Register

31	30	29	16
TDA	RDA	Reserved	
0	0	0x0000	
r	r	r	

15	4	3	2	1	0
Reserved		CTD	CRD	NTD	NRD
0x000		0	0	0	0
r		w	w	w	w

31 Transmit Descriptor Available (TDA). Read-only bit indicating whether the TX descriptor table contains descriptors yet to be fetched and processed. Cleared when the core finds a disabled descriptor (Descriptor Enable bit set to 0b inside the descriptor) or when a bus master error occurs during transmission.

30 Receive Descriptor Available (RDA). Read-only bit indicating whether the RX descriptor table contains descriptors yet to be fetched and processed. Cleared when the core finds a disabled descriptor (Descriptor Enable bit set to 0b inside the descriptor) or when a bus master error occurs during reception.

3 Clear Transmit Descriptor Table (CTD). Write-only bit that, when set, clears the content of the TX Descriptor Table. It takes effect only when the DMA channel handling the current Virtual Channel is in IDLE state.

2 Clear Receive Descriptor Table (CRD). Write-only bit that, when set, clears the content of the RX Descriptor Table. It takes effect only when the DMA channel handling the current Virtual Channel is in IDLE state.

1 New Transmit Descriptor Available (NTD). Write-only bit that indicates to GRSPFI that new descriptors have been added to the TX Descriptor table. It takes effect only when the DMA channel handling the current Virtual Channel is in IDLE state.

0 New Receive Descriptor Available (NRD). Write-only bit that indicates to GRSPFI that new descriptors have been added to the RX Descriptor table. It takes effect only when the DMA channel handling the current Virtual Channel is in IDLE state.

74.12.34 VC Statistics Register

Table 1225. 0x90C, 0x91C, 0x92C... 0xAFC - VCSTAT - VC Statistics Register

31	16
TX	
0x0000	
wc	

15	0
RX	
0x0000	
wc	

31: 16 Transmit counter (TX). Number of packets transmitted over the selected virtual channel. The counter is reset when writing to this field a value other than 0.

15: 0 Receive counter (RX). Number of packets received over the selected virtual channel. The counter is reset when writing to this field a value other than 0.

74.12.35 DMA Channel Control Register

Table 1226. 0xB00, 0xB10, 0xB20... 0xB70 - DCCTRL - DMA Channel Control Register

31	16
Reserved	
0x0000	
r	

15	4	3	2	1	0
Reserved		TS	RS	TE	RE
0x000		0	0	0	0
r		w	w	w	w

3 Transmitter Stop (TS). If set to 1b, GRSPFI will disable the transmitter of the current DMA channel as soon as the Transmit FSM enters IDLE state. Any pending TX enable request is cleared.

2 Receiver Stop (RS). If set to 1b, GRSPFI will disable the receiver of the current DMA channel as soon as the Receive FSM enters IDLE state. Any pending RX enable request is cleared.

1 Transmitter Enable (TE). If set to 1b, GRSPFI will enable the transmitter of the current DMA channel as soon as the Transmit FSM enters IDLE state. Any pending TX stop request is cleared.

0 Receiver Enable (RE). If set to 1b, GRSPFI will enable the receiver of the current DMA channel as soon as the Receive FSM enters IDLE state. Any pending RX stop request is cleared.

74.12.36 DMA Channel Status Register

Table 1227. 0xB04, 0xB14, 0xB24... 0xB74 - DCSTAT - DMA Channel Status Register

31	26	25	24	20	19	18	17
Reserved		RD	BMEVC		Reserved		BMEDSC
0x00		0	0x00		00		0x000
r		wc	wc		r		wc

8	7	4	3	2	1	0
BMEDSC	BMECD		TX	RX	TA	RA
0x000	0x0		0	0	0	0
wc	wc		wc	wc	r	r

25 RMAP done (RD). Set when the RMAP target has completed the processing of an RMAP command, including the transmission of a reply, if applicable. Only used when the VHDL generic *rmap* is set.

24: 20 Bus Master Error - Virtual Channel (BMEVC). This field identifies which Virtual Channel was being served when an error occurred during an access through the bus master interface. If no Virtual Channels are involved (i.e. Broadcast Channel) then this field returns 0x00. Cleared to 0x00 when the Bus Master Error Code field is written.

17: 8 Bus Master Error - Descriptor (BMEDSC). This field identifies which descriptor was being used when an error occurred during an access through the bus master interface. If no descriptors are involved (i.e. Broadcast Channel) then this field returns 0x000. Cleared to 0x000 when the Bus Master Error Code field is written.

7: 4 Bus Master Error - Code (BMECD). This field specifies what type of error occurred when accessing the bus. Writing to this field clears the content of all the bus master error-related fields: BMEVC, BMEDSC and BMECD. Encoding:

- 0x0: No errors
- 0x1: Fetching RX descriptor
- 0x2: Storing received SpaceFibre packet
- 0x3: Updating RX descriptor
- 0x4: Storing received broadcast message
- 0x5: Fetching TX descriptor
- 0x6: Fetching the header of a SpaceFibre packet for transmission
- 0x7: Fetching the data of a SpaceFibre packet for transmission
- 0x8: Updating TX descriptor
- 0x9: Fetching broadcast message for transmission
- 0xA: Write access to the bus while processing an RMAP command
- 0xB: Read access to the bus while processing an RMAP command

3 Packet transmitted (TX). If set to 1b, this DMA channel has fetched and passed a complete SpaceFibre packet on to the codec, and the associated descriptor has been updated. Cleared to 0b when written.

2 Packet received (RX). If set to 1b, this DMA channel has read a complete SpaceFibre packet from the codec and written it to external memory, and the associated descriptor has been updated. Cleared to 0b when written.

1 Transmitter Active (TA). If set to 1b, the transmitter of this DMA channel is active. It may imply that the Transmitter Enable bit in the DMA Channel Control Register is set, or that the DMA channel is still completing an ongoing operation before being disabled.

0 Receiver Active (RA). If set to 1b, the receiver of this DMA channel is active. It may imply that the Receiver Enable bit in the DMA Channel Control Register is set, or that the DMA channel is still completing an ongoing operation before being disabled.

74.12.37 DMA Channel VC Mapping Register

Table 1228. 0xB08, 0xB18, 0xB28... 0xB78 - DCMAP - DMA Channel VC Mapping Register

31

VCMAP
0x00000000
rw

0

VCMAP
0x00000000
rw

31: 0 DMA channel - Virtual Channel Mapping (VCMAP). This register configures which Virtual Channels are served by the current DMA channel. A specific Virtual Channel may only be handled by a single DMA channel at a time. Each bit (from 0 to 31) represents a different Virtual Channel; positions for Virtual Channels not implemented are reserved and will always return 0b.

74.12.38 DMA Channel IRQ Control Register

Table 1229. 0xB0C, 0xB1C, 0xB2C... 0xB7C - DCICTRL - DMA Channel IRQ Control Register

31	16
Reserved	
0x0000	
r	

15	4	3	2	1	0
Reserved		RMI	BEI	TI	RI
0x000		0	0	0	0
r		rw	rw	rw	rw

3 RMAP Interrupt Enable (RMI). If set to 1b, GRSPFI will generate an AMBA interrupt when an RMAP command has been processed, including the transmission of a reply when applicable. Only used if the VHDL generic *rmap* is set.

2 Bus Master Error Interrupt Enable (BEI). If set to 1b, GRSPFI will generate an AMBA interrupt when an error is detected during an access over the bus master interface of this DMA channel.

1 Transmit Interrupt Enable (TI). If set to 1b, GRSPFI will generate AMBA interrupts when this DMA channel has completely fetched a SpaceFibre packet and passed it on to the codec for transmission, and the corresponding descriptor has been updated. Note that the Interrupt Enable bit in the descriptor associated with the packet shall also be set to 1b.

0 Receive Interrupt Enable (RI). If set to 1b, GRSPFI will generate AMBA interrupts when this DMA channel has completely read a SpaceFibre from the codec and stored it in external memory, and the corresponding descriptor has also been updated. Note that the Interrupt Enable bit in the descriptor associated with the packet shall also be set to 1b.

74.12.39 DMA Channel Extended Status Register

Table 1230. 0xB80, 0xB90, 0xBA0... 0xBF0 - DCSTAT2 - DMA Channel Extended Status Register

31	16
RC	
0x0000	
wc	

15	14	13	12	8	7	6	5	4	0
TAF	TF	TS	TFS		RAE	RE	RS	RFS	
0	0	0	0x01		1	1	0	0x01	
r	r	r	r		r	r	r	r	

31: 16 RMAP counter (RC). Number of RMAP commands successfully processed by the core. Cleared when written to a value other than 0. Only used if the VHDL generic *rmap* is set.

15 TX FIFO Almost Full (TAF). Read-only flag generated by the local TX FIFO indicating that it is almost full, i.e., its available space is lower than 256 bytes.

14 TX FIFO Full (TF). Read-only flag generated by the local TX FIFO indicating that it is full.

13 TX Start (TS). Read-only signal generated by the DMA FSM to indicate an ongoing transmission.

12: 8 TX FSM State (TFS). Read-only field containing the current state of the TX FSM. Encoding:
0x01 Idle | 0x02 Select next Virtual Channel | 0x03 Fetch TX descriptor | 0x04 Read TX descriptor | 0x05 Start TX | 0x06 Fetch header bytes | 0x07 Burst access for header bytes | 0x08 Fetch data bytes | 0x09 Burst access for data bytes | 0x0A Wait for TX end | 0x0B Update TX descriptor | 0x0C End of TX | 0x0D Fetch broadcast message | 0x0E Read broadcast message | 0x0F Transmit broadcast message | 0x10 Form RMAP reply header | 0x11 RMAP read operation | 0x12 RMAP single read access | 0x13 RMAP incremental read access | 0x14 RMAP RMW data | 0x15 End of RMAP reply.

7 RX FIFO Almost Empty (RAE). Read-only flag generated by the local RX FIFO indicating that it is almost empty, i.e., it contains less than 256 bytes of data.

6 RX FIFO Empty (RE). Read-only flag generated by the local RX FIFO indicating that it is empty.

5 RX Start (RS). Read-only signal generated by the DMA FSM to indicate an ongoing reception.

4: 0 RX FSM State (RFS). Read-only field containing the current state of the RX FSM. Encoding: 0x01 Idle | 0x02 Select next Virtual Channel (descriptor prefetching) | 0x03 Prefetch descriptor | 0x04 Read descriptor | 0x05 Select next virtual channel (data storage) | 0x06 Store packet | 0x07 Write burst stage 1 | 0x08 Write burst stage 2 | 0x09 Write burst stage 3 | 0x0A Update RX descriptor | 0x0B End of RX | 0x0C Store broadcast message | 0x0D Broadcast message RX end | 0x0E Select next virtual channel (RMAP) | 0x0F Read RMAP header | 0x10 Decode RMAP header | 0x11 RMAP non-verified command | 0x12 RMAP single write access | 0x13 RMAP incremental write stage 0 | 0x14 RMAP incremental write stage 1 | 0x15 RMAP verified command | 0x16 RMAP verified read (RMW) | 0x17 RMAP verified write | 0x18 RMAP read | 0x19 Discard RMAP packet | 0x1A Waiting on RMAP reply.

74.13 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x0BC (GRSPFI). For description of vendor and device identifiers see GRLIB IP Library User's Manual.

74.14 Configuration options

Table 1231 shows the configuration options of the core (VHDL generics).

*Table 1231.*Configuration options

Generic name	Function	Allowed range	Default
tech	Technology for the internal memories.	0 - NTECH	inferred (0)
hmindex	AHB master index of the first DMA channel. The rest of the DMA channels are assigned an index depending on the settings of the generic *incr_hmindex*. Only present in the AHB wrapper.	0 - NAHBMST-1	0
hsindex	AHB slave index.	0 - NAHBSLV-1	0
haddr	Addr field of the AHB bar.	0 - 16#FFF#	0
hmask	Mask field of the AHB bar.	0 - 16#FFF#	16#FF0#
hirq	Interrupt line used by GRSPFI.	0 - NAHBIRQ-1	0
use_8b10b	If set, internal 8B10B encoding and decoding is activated.	0 - 1	1
use_sep_txclk	If set, the SerDes transmission clock is decoupled from the SpaceFibre port clock. An additional transmit buffer is instantiated for this reason.	0 - 1	0
sel_16_20_bit_mode	If set, the SerDes interface is 16+2 bit (without 8B10B) or 20 bit (with 8B10B) wide instead of 36/40 bit. If set, *use_sep_txclk* must also be set.	0 - 1	0
ticks_2us	Clock ticks corresponding to 2 us.	1 - 8192	125
tx_skip_freq	Frequency of SKIP word transmission in clock cycles.	1 - 8192	5000
prbs_init1	If set, the INIT1 sequence during lane initialization is embedded into a stream of pseudo-random numbers.	0 - 1	1
depth_rbuf_data	Log(Depth) of the data retry buffer.	1 - 32	8
depth_rbuf_fct	Log(Depth) of the FCT retry buffer.	1 - 32	4
depth_rbuf_bc	Log(Depth) of the broadcast retry buffer.	1 - 32	8
num_vc	Number of virtual channels.	1 - 32	4
fct_multiplier	Multiplier used in the internal FCT logic. The value of the FCT counter is compared with 64**fct_multiplier* to decide if the far end can receive new words.	1 - 8	1
depth_vc_rx_buf	Log(Depth) of virtual channel input buffer.	1 - 32	10
depth_vc_tx_buf	Log(Depth) of virtual channel output buffer.	1 - 32	10
remote_fct_cnt_max	Width of the remote FCT counter.	1 - 32	9
width_bw_credit	Width of the bandwidth credit counter.	0 - 16#FFFFFFFF#	20

*Table 1231.*Configuration options

Generic name	Function	Allowed range	Default
min_bw_credit	Minimum bandwidth credit threshold limit.	0 - 16#FFFFFFFF#	52428
idle_time_limit	Bandwidth idle time limit in clock cycles.	0 - 65535	62500
num_dmach	Number of DMA channels.	1 - 8	1
num_txdesc	Size of the TX descriptor table in number of descriptors.	64, 128, 256, 512	64
num_rxdesc	Size of the RX descriptor table in number of descriptors.	128, 256, 512, 1024	128
depth_dma_fifo	Depth of the DMA FIFOs. Sets the number of entries in the 128-bit transmitter and receiver FIFOs (2 FIFOs per DMA channel).	16 - 256	32
depth_bc_fifo	Depth of the BC FIFO. Sets the number of entries in the 82-bit receiver FIFO.	1 - 32	4
incr_hmindex	If set, the AHB master indices of the DMA channels are assigned incrementally, starting with hmindex. Otherwise, all AHB master indices are set to hmindex. Only present in the AHB wrapper.	0 - 1	1
use_async_rxrst	Force asynchronous reset in the recovered clock domain, regardless of the general GRLIB settings.	0 - 1	0
ft_core_vc	Enable fault-tolerance against SEU errors for the FIFOs in the Virtual Channel Layer. Possible options are byte parity protection (ft = 1), TMR registers (ft = 2), SECDED BCH (ft = 4) or technology specific protection (ft = 5). If set to 0, no protection is implemented.	0 - 2, 4 - 5	0
ft_core_rt1	Enable fault-tolerance against SEU errors for the large FIFOs in the Retry Layer (Data and Broadcast). The core supports the same configurations as above.	0 - 2, 4 - 5	0
ft_core_rt2	Enable fault-tolerance against SEU errors for the small FIFOs in the Retry Layer (FCT and pointers). The core supports the same configurations as above.	0 - 2, 4 - 5	0
ft_core_if	Enable fault-tolerance against SEU errors for the FIFOs in the Interface Layer. The core supports the same configurations as above.	0 - 2, 4 - 5	0
ft_dma_data	Enable fault-tolerance against SEU errors for the TX/RX FIFOs in the DMA layer. The core supports the same configurations as above.	0 - 2, 4 - 5	0
ft_dma_bc	Enable fault-tolerance against SEU errors for the Broadcast Channel FIFOs in the DMA layer. The core supports the same configurations as above.	0 - 2, 4 - 5	0
scantest	Enable Scan Test support.	0 - 1	0
ahbbits	Width of AHB read/write data buses and maximum access size. Only present in the AHB wrapper.	32, 64, 128	AHBDW
lendian	Select endianness of the system: 0 for big-endian, 1 for little-endian. Only present in the AXI wrapper.	0 - 1	1
rmap	If set, every DMA channel will feature a hardware RMAP target and RMAP CRC logic.	0 - 1	0
rmapcrc	Include RMAP CRC logic in every DMA channel without implementing a full RMAP target.	0 - 1	0
ccsdscrc	Enable CCSDS/CCITT CRC-16 and 16-bit ISO-checksum (J.G. Fletcher, ISO 8473-1:1998) logic in every DMA channel. It requires that either *rmap* or *rmapcrc* is also set to 1.	0 - 1	0

*Table 1231.*Configuration options

Generic name	Function	Allowed range	Default
nodeaddr	When set to 0 - 254: this generic specifies the reset value for the DEFADDR.DEFADDR field. When set to 255: Reset value for the node address is taken from the CFG.NODEADDR input signal.	0 - 255	254
destkey	Reset value for the core destination key.	0 - 255	0
numextvc	Number of external FIFO interfaces that can be used to bypass the DMA engine when transmitting and receiving packets over a specific virtual channel.	0 - 32	0
numextbc	If set, implement an external FIFO interface that can be used to bypass the DMA engine when transmitting and receiving broadcast messages.	0 - 1	0

74.15 Signal descriptions

Table 1232 shows the interface signals of the core when using the wrapper for AMBA AHB 2.0 (VHDL ports).

Table 1232. Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	System reset	Low
CLK	N/A	Input	AMBA clock	-
SPFI_RSTN	N/A	Input	SpaceFibre reset	Low
SPFI_CLK	N/A	Input	SpaceFibre clock	-
SPFI_TXCLK	N/A	Input	Optional SpaceFibre transmit clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	Array of AHB master output signals (one set of AHB outputs per DMA channel)	-
SPFII	RX_CLK	Input	SerDes receive clock	-
	RX_DATA	Input	Receive Data	-
	RX_KFLAGS	Input	Receive K-Flags	-
	RX_SERROR	Input	Receive Error Flags	High
	NO_SIGNAL	Input	SerDes No Signal Flag	High
SPFIO	TX_DATA	Output	Transmit Data	-
	TX_KFLAGS	Output	Transmit K-Flags	-
	TX_EN	Output	SerDes Transmitter Enable Flag	High
	RX_EN	Output	SerDes Receiver Enable Flag	High
	INV_POL	Output	SerDes Invert Polarity Flag	High
	TX_DATA_DBG	Output	Unencoded 32-bit transmit data	-
	TX_KFLAGS_DBG	Output	4-bit transmit K-Flags	-
	RX_DATA_DBG	Output	Unencoded 32-bit receive data	-
	RX_KFLAGS_DBG	Output	4-bit receive K-Flags	-

Table 1232. Signal descriptions

Signal name	Field	Type	Function	Active
CFG	RMAPEN	Input	Reset value for the VCCTRL0.RE bit. If set, the link auto-start bit is also automatically set, the virtual channel 0 is assigned 95% of the bandwidth and all time-slots, and is served by the DMA channel 0, so that the IP can process RMAP commands right after reset. Used only if the *rmap* generic is also set.	High
	NODEADDR	Input	Reset value for the node address (DEFADDR.DEFADDR field) when the *nodeaddr* VHDL generic = 255. Unused if *nodeaddr* /= 255.	-
EXTVCI[31:0]	TX_DATA	Input	Virtual channel transmit data when using the external FIFO interface.	-
	TX_KFLAGS	Input	Virtual channel transmit K-flags when using the external FIFO interface.	-
	TX_WEN	Input	Virtual channel transmit write enable when using the external FIFO interface.	High
	RX_REN	Input	Virtual channel receive read enable when using the external FIFO interface.	High
EXTVCO [31:0]	TX_FULL	Output	Virtual channel transmit full flag when using the external FIFO interface.	High
	RX_DATA	Output	Virtual channel receive data when using the external FIFO interface.	-
	RX_KFLAGS	Output	Virtual channel receive K-flags when using the external FIFO interface.	-
	RX_VALID	Output	Virtual channel receive valid flag when using the external FIFO interface.	High
EXTBCI	TX_DATA	Input	Broadcast transmit data when using the external FIFO interface.	-
	TX_CHANNEL	Input	Broadcast transmit channel when using the external FIFO interface.	-
	TX_BTYPE	Input	Broadcast transmit broadcast type when using the external FIFO interface.	-
	TX_DELAYED	Input	Broadcast transmit delayed flag when using the external FIFO interface.	High
	TX_LATE	Input	Broadcast transmit late flag when using the external FIFO interface.	High
	TX_WEN	Input	Broadcast transmit write enable when using the external FIFO interface.	High

Table 1232. Signal descriptions

Signal name	Field	Type	Function	Active
EXTBCO	TX_ACK	Output	Broadcast transmit acknowledge flag when using the external FIFO interface.	High
	RX_DATA	Output	Broadcast receive data when using the external FIFO interface.	-
	RX_CHANNEL	Output	Broadcast receive channel when using the external FIFO interface.	-
	RX_BTYPE	Output	Broadcast receive broadcast type when using the external FIFO interface.	-
	RX_VALID	Output	Broadcast receive valid flag when using the external FIFO interface.	High
	RX_DELAYED	Output	Broadcast receive delayed flag when using the external FIFO interface.	High
	RX_LATE	Output	Broadcast receive late flag when using the external FIFO interface.	High
MTESTI**	SPFI	Input	Memory BIST input signals to the FIFOs in the SpaceFibre codec. Please refer to the GRSPFI_CODEC section of this document for further details about the fields of this record	-
	DMA_TX	Input	Memory BIST input signal to the DMA TX FIFO	-
	DMA_RX	Input	Memory BIST input signal to the DMA RX FIFO	-
	DMA_BC	Input	Memory BIST input signal to the DMA Broadcast FIFO	-
MTESTO**	SPFI	Output	Memory BIST output signals from the FIFOs in the SpaceFibre codec. Please refer to the GRSPFI_CODEC section of this document for further details about the fields of this record	-
	DMA_TX	Output	Memory BIST output signal from the DMA TX FIFO	-
	DMA_RX	Output	Memory BIST output signal from the DMA RX FIFO	-
	DMA_BC	Output	Memory BIST output signal from the DMA Broadcast FIFO	-
MTESTCLK**	N/A	Input	Memory BIST clock	-

* see GRLIB IP Library User's Manual

** not available in FPGA releases

74.16 Library dependencies

Table 1233 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1233.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GRLIB	GENERIC BM PKG	Signals, components	Package including the generic bus master interface and the bridges for converting to AHB and AXI
GAISLER	HSSL	Signals, component	GRSPFI component and signal declarations

74.17 Instantiation

This example shows how the core can be instantiated using the AMBA 2.0 AHB wrapper.

```
library  ieee;
use      ieee.std_logic_1164.all;
library  gaisler;
use      gaisler.hssl.all;

entity example is
   generic (
      tech:       in    integer := 0);
   port (
   );

-- Signal declarations
signal clk       : std_ulogic;
signal rstn      : std_ulogic;
signal spfi_clk  : std_ulogic;
signal spfi_rstn : std_ulogic;

signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
signal ahbmi : ahb_mst_in_type;
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);

signal spfii : grhssl_in_type;
signal spfio : grhssl_out_type;

------------------------------------------------------------------------------
-- Component instantiation
------------------------------------------------------------------------------

  -- SpaceFibre IP core
  spfi_dut : grspfi_ahb
generic map (
      tech               => 0,
      hmindex            => 1,
      hsindex            => 0,
      haddr              => 16#A00#,
      hmask              => 16#FF0#,
      use_8b10b          => 1,
      use_sep_txclk      => 0,
      sel_16_20_bit_mode => 0,
      ticks_2us          => 125,
      tx_skip_freq       => 5000,
      prbs_init1         => 1,
      depth_rbuf_data    => 8,
      depth_rbuf_fct     => 4,
      depth_rbuf_bc      => 8,
      num_vc             => 4,
      fct_multiplier     => 1,
      depth_vc_rx_buf    => 8,
      depth_vc_tx_buf    => 8,
      remote_fct_cnt_max => 8,
      width_bw_credit    => 20,
      min_bw_credit      => 52428,
      idle_time_limit    => 62500,
      num_dmach          => 2,
      num_txdesc         => 256,
      num_rxdesc         => 512
      )
port map (
      clk        => clk,
      rstn       => rstn,
      spfi_clk   => spfi_clk,
      spfi_rstn  => spfi_rstn,
      spfi_txclk => '0',                -- Not using separate TX clock
      -- AHB interface
      ahbmi      => ahbmi,
      ahbmo      => ahbmo(2 downto 1),
```

```
      ahbsi      => ahbsi,
      ahbso      => ahbso(0),
      -- Serdes interface
      spfii      => spfii,
      spfio      => spfio);
```

75 GRSPFI_CODEC - SpaceFibre encoder/decoder

75.1 Overview

SpaceFibre is a high-speed serial link mainly designed for payload data processing applications on board spacecraft. Like many other modern network architectures, SpaceFibre utilises a Serialiser/ Deserialiser (SerDes) circuit at its physical layer, allowing data rates of 2 Gbit/s and more. The SerDes can either be part of the chip design or a standalone device can be used.

Interfacing a SpaceFibre port from the user application is simple as it closely follows the procedure known from SpaceWire. A SpaceFibre port has one or more pairs of transmit and receive buffers, referred to as virtual channels, and each virtual channel acts like a single SpaceWire interface, i.e. several SpaceWire network streams can be multiplexed into one SpaceFibre network stream. The multiplexer is called medium access controller and is choosing the active virtual channel according to a number of Quality-of-Service (QoS) rules.

Data is always transferred in frames with a size of 256 bytes or less. While such a data frame is passed to the physical link, it is also stored in an error recovery buffer. It remains in this buffer until the far end node acknowledges the correct reception of the frame, which is detected by checking a CRC checksum at the end of the frame. However, if the far end node sends a negative-acknowledgement (NACK) word instead, the frame is re-transmitted from the error-recovery buffer.

Aside from data frames, SpaceFibre also supports broadcast frames, which are multi-purpose high-priority messages. These messages are comparable to SpaceWire time-codes but in addition to a simple sequence number they also transmit a data payload of 8 bytes. Broadcast frames are stored in the replay buffer just like the data frames, i.e. they are automatically retransmitted after a link error.

On the receive side, incoming data from the physical link is processed continuously, i.e. one 32-bit word is processed every clock cycle. To avoid buffer overruns in the virtual channel receive buffers, the communication between a virtual channel transmit buffer in the local node and the virtual channel receive buffer in the far end node is flow-controlled by means of Flow Control Token (FCT) words. Just like the data and broadcast frames, the FCT words are stored in the error-recovery buffer and are therefore retransmitted in case of errors.

A simplified block diagram of the SpaceFibre IP core can be seen in Figure 182. The SpaceFibre IP port comprises a data link layer and lane layer. Internally, the data link layer is further divided into the so-called broadcast layer, virtual channel layer and retry layer, which are responsible for the transmission and reception of broadcast frames, for the transmission and reception of data frames and for the error recovery mechanism, respectively.

Signals related to the broadcast interface have the prefix 'bc', signals related to the virtual channel interface the prefix 'vc', and signals related to the SerDes the prefix 'se'. Furthermore, the prefix 'cf' describes a configuration parameter signal whereas the prefix 'sr' describes a status register signal.

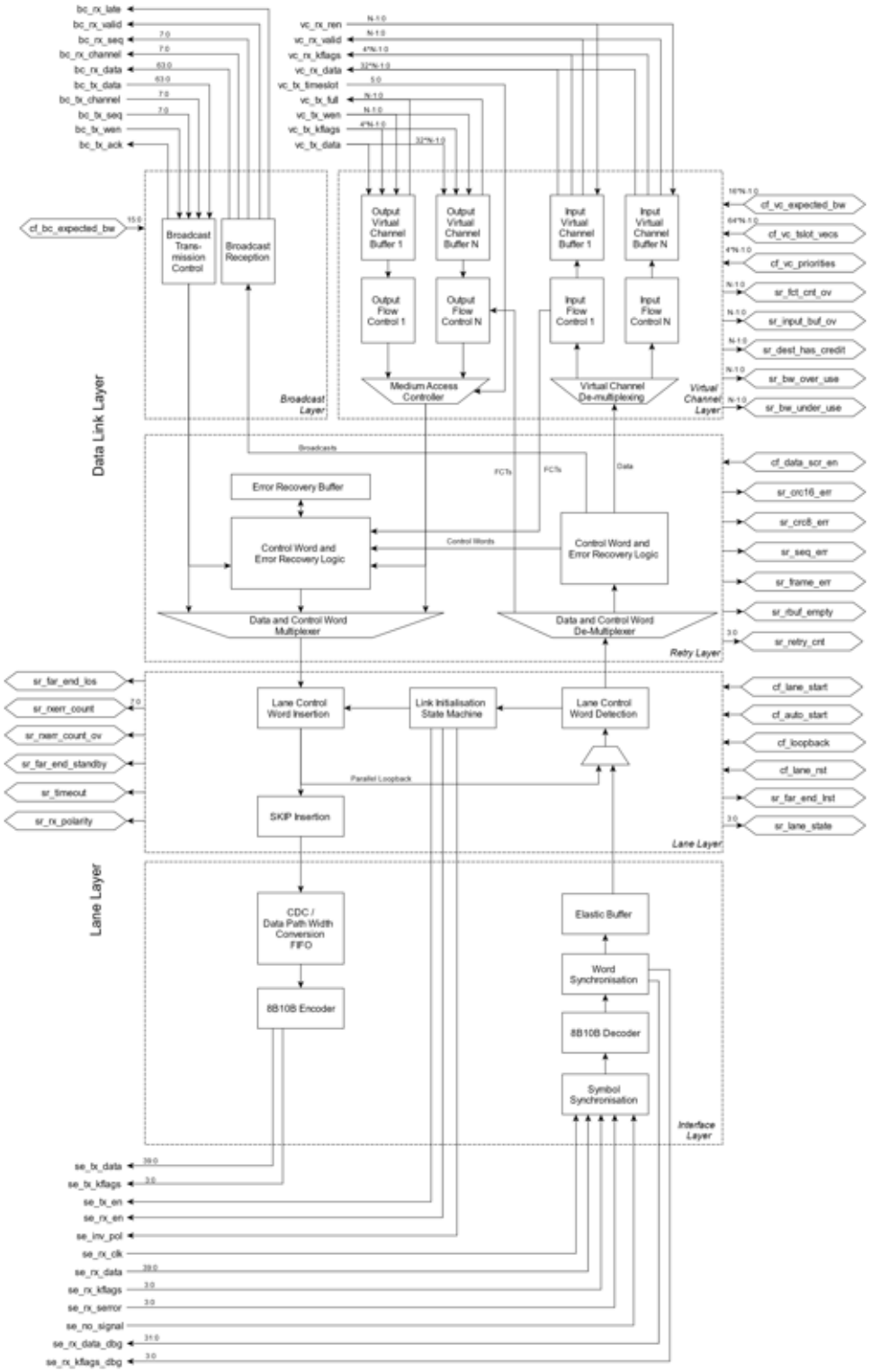


Figure 182. GRSPFI_CODEC block diagram

75.2 Operation

75.2.1 Configuration

All soft configuration parameters are synchronous to clock *clk* and should be stored in registers external to the SpaceFibre IP core.

Global configuration

The SpaceFibre link can be reset by asserting *cf_link_rst*. As a consequence, outbound packets are spilled up to and including the next EOP and inbound packets are terminated by an EEP.

Lane Layer

The lane initialization state machine is controlled with the signals *cf_lane_start* and *cf_auto_start*. When *cf_lane_start* is asserted, the lane layer will actively try to build up a connection with the far end by sending out INIT1 words. When *cf_auto_start* is asserted, the lane layer will passively listen to incoming words from the far end and start the handshake procedure once INIT1 words are received. *cf_lane_start* has higher precedence than *cf_lane_auto_start*. To shut down the lane, both *cf_lane_auto_start* and *cf_lane_start* must be de-asserted. When shut down, the transmitter and receiver parts of the SerDes are disabled as well.

The lane can be reset at any time by asserting *cf_lane_rst*. In contrast to link resets, a lane reset is only reinitialising the lane. Therefore, ongoing transmissions are probably delayed but never canceled.

The SpaceFibre IP core has an internal loop-back feature that can be enabled/ disabled with *cf_la_loopback*. The SpaceFibre IP core must be reset after asserting/de-asserting this flag.

Retry Layer

The retry layer comprises a data scrambler, which can be enabled/disabled with *cf_data_scr_en*. The data scrambler becomes directly active after asserting this flag. This will lead to continuous data retries if the de-scrambler of the remote node is not activated. Therefore, a lane reset must be triggered after asserting/de-asserting this flag. By doing so, the remote node is informed during the subsequent handshake if the data is scrambled or not and will then switch on or off the de-scrambler accordingly.

Broadcast Layer

The transmission of broadcasts is limited in bandwidth depending on an expected bandwidth value defined by the 16-bit wide port *cf_bc_expected_bw*. Typical values are between 1% and 95%. If the user application tries to send more broadcasts than allowed, the bandwidth limitation becomes automatically active. *cf_bc_expected_bw* is expressed in 0.16 fixed point format and is calculated as follows:

$$ExpectedBandwidth = \frac{Percentage}{400}$$

For example, the expected bandwidth value of 10% is expressed as: 10/400 = 0.025. In 0.16 fixed point notation: 0x0666. For convenience, these values are pre-calculated for 1% to 95% and stored in a constant array in the package file spfi_pkg.vhd. The transmission of broadcasts can also be completely switched off by setting the expected bandwidth value to 0x0000.

Virtual Channel Layer

The transmission of data frames is bandwidth-limited for each virtual channel. The expected bandwidth values for all virtual channels are concatenated in port vector *cf_vc_expected_bw*. Each expected bandwidth value is 16-bit wide. For instance, the value for virtual channel 0 is stored at bit positions 15:0 and the value for virtual channel 1 is stored at bit positions 31:16. The expected bandwidth values are expressed in 8.8 fixed point format and are calculated as follows:

$$ExpectedBandwidth = \left|1 - \frac{100}{Percentage}\right|$$

For example, the expected bandwidth value of 15% is expressed as: abs(1-(100/15)) = 5.6667. In 8.8 fixed point notation: 0x05ab. For convenience, these values are pre-calculated for 1% to 95% and stored in a constant array in the package file spfi_pkg.vhd. A virtual channel can also be completely switched off by setting its expected bandwidth value to 0x0000.

The timeslot vectors for all virtual channels are concatenated in port vector *cf_vc_tslot_vecs*. Since SpaceFibre offers 64 timeslots per virtual channel, the timeslot vector for virtual channel 0 is stored at bit positions 63:0, the timeslot vector for virtual channel 1 at bit positions 127:0 and so on. Each bit enables or disables the transmission of data frames in the timeslot with the number of the bit position. For instance, to allow virtual channel 0 to send data in timeslot 0, timeslot 3 and timeslot 7, one can set its timeslot vector to: 0b10001001 = 0x89.

The priority values for all virtual channels are concatenated in port vector *cf_vc_priorities*. Each priority value is 4-bit wide, i.e. the priority levels 0 to 15 are supported. For instance, the value for virtual channel 0 is stored at bit positions 3:0 and the value for virtual channel 1 is stored at bit positions 7:4. Priority level 15 has the lowest precedence and priority level 0 has the highest precedence.

75.2.2 Status signaling

Lane Layer

The status flag *sr_far_end_lrst* is pulsed high if a link reset was triggered due to a system reset in the far end node. *sr_far_end_los* is pulsed if a new handshake is started due to a loss of signal at the SerDes receiver. *sr_rxerr_count* is an 8-bit wide counter for received RXERR words. The counter is automatically decreased by one every time 16,384 more words have been received. *sr_rxerr_count_of* is pulsed when the RXERR word counter overflows. *sr_rx_polarity* is set when the lane layer detects inverted receive polarity during handshake. If internal 8B10B decoding is activated, the polarity of the incoming bitstream is automatically inverted.

sr_far_end_standby is pulsed when STANDBY words are received. This is always the case when the link initialization state machine in the far end node is deactivated. *sr_timeout* is set when a connection timeout occurs during lane initialization. *sr_lane_state* is a 4-bit wide unsigned value encoding the current lane initialization state as follows:

Table 1234. Lane initialization state encoding

1	Clear Line
2	Disabled
3	Wait
4	Started
5	Invert Rx Polarity
6	Connecting
7	Connected
8	Active
9	Loss Of Signal
10	Prepare Standby

Retry Layer

sr_crc16_err is pulsed when a CRC error has occurred during the reception of a data frame. *sr_crc8_err* is pulsed when a CRC error has occurred during the reception of a control word or broadcast frame. *sr_seq_err* is pulsed when a sequence number error occurred during the reception of data frames, broadcast frames, idle frames, FCT or FULL words. *sr_frame_err* is pulsed when an invalid data, broadcast, or idle frame is received.

sr_rbuf_empty is set when the error recovery buffer is empty, i.e. all data frames, broadcast frames and FCT words have been sent and acknowledged. *sr_retry_cnt* is an 4-bit wide counter that increments for every RETRY event initiated. It is automatically reset when an ACK word is received from the far end. The SpaceFibre link is automatically reset when the counter reaches 16.

Virtual Channel Layer

sr_fct_cnt_ov is a bit vector where each bit position corresponds to a virtual channel number. The bit is pulsed when the corresponding FCT credit counter has overflowed. This indicates that the width of the FCT counter, defined by *remote_fct_cnt_max*, is too small. *sr_input_buf_ov* is a bit vector where each bit position corresponds to a virtual channel number. The bit is set when a virtual channel input buffer is receiving data when it is full. This indicates a fatal protocol error. *sr_dest_has_credit* is a bit vector where each bit position corresponds to a virtual channel number. The bit is set when there is space in the virtual channel input buffer of the far end node. *sr_bw_over_use* is a bit vector where each bit position corresponds to a virtual channel number. The bit is set when the virtual channel uses more bandwidth than expected, i.e. when the bandwidth overuse mechanism is active. In contrast, *sr_bw_under_use* indicates that a virtual channel uses less bandwidth than expected. Note that this flag is also set when a virtual channel is deactivated due to an expected bandwidth value of 0, even if the user application tries to transmit data over this virtual channel.

75.2.3 Virtual channel interface

Transmit side

On port *vc_tx_data*, data is fed into the virtual channel transmit buffers. The input vectors are 32-bit wide and concatenated, i.e. the data word for virtual channel 0 is stored at bit positions 31:0, the data word for virtual channel 1 is stored at bit positions 63:32 and so on. The lowest byte is transmitted first. On port *vc_tx_kflags*, a 4-bit vector for each virtual channel defines which bytes of the data transmit word are k-codes. The vectors are concatenated as well, i.e. the k-flags vector for virtual channel 0 is stored at bit positions 3:0, the k-flags vector for virtual channel 1 is stored at bit positions 7:4 and so on. The lowest bit in the k-flags vector corresponds to the lowest byte of the data word. The following k-codes are allowed:

Table 1235. SpaceFibre k-codes

SpaceFibre Character	K-Code	Corresponding data byte
EOP – End of packet	K29.7	0xFD
EEP – Error end of packet	K30.7	0xFE
FILL – Fill character	K27.7	0xFB

An example of how to use the k-codes correctly is shown in figure 183. The first four lines are normal data words, which are part of the payload of a SpaceWire packet. The k-flags vector must be set to 0b0000. In line 5, the SpaceWire packet is terminated by an EOP. Due to the byte-order, the k-code for the EOP is transmitted as least significant byte. The EOP is followed by three FILL characters for achieving word alignment. The k-flags vector must be set to 0b1111 because all bytes of the word are k-codes. In line 6, a new SpaceWire packet begins, starting with physical address 0x08. The address is transmitted as most significant byte since three FILL characters must be transmitted first. Here, the k-flags vector must be set to 0b0111 because only the FILL characters are k-codes.

Usage of any other k-codes than the aforementioned ones for EOP, EEP an FILL characters must be avoided. SpaceFibre uses k-codes for control words on lower protocol layers. Injecting the wrong combination of k-codes through the virtual channel interface can compromise the correct functionality of the SpaceFibre protocol.

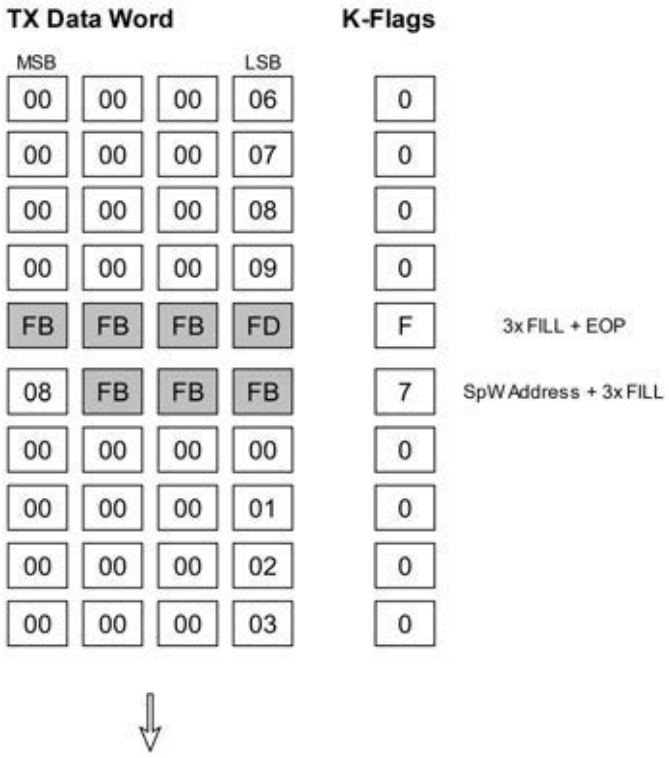


Figure 183. Example of k-code usage

To write into a virtual channel transmit buffer, the write enable flag for the virtual channel in bit vector *vc_tx_wen* must be asserted. The bit position corresponds to the virtual channel number. If the transmit buffer is full, the full flag for the virtual channel in bit vector *vc_tx_full* is asserted. Again, the bit position of the flag corresponds to the virtual channel number. If the transmit buffer is full, any further write attempts are ignored. An example timing diagram for virtual channel 0 is shown in figure 184.

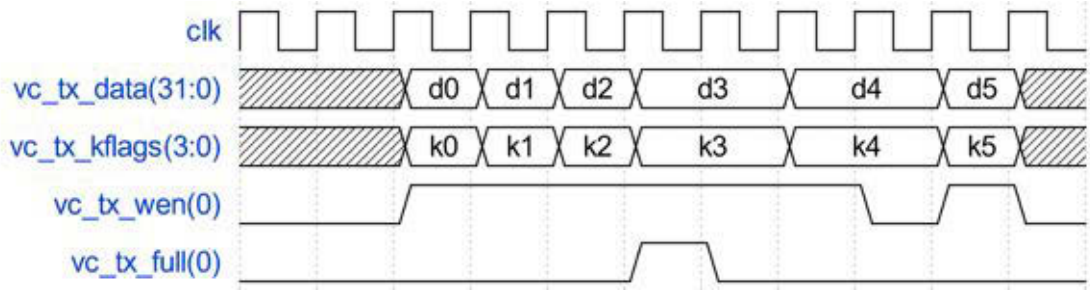


Figure 184. Example timing diagram - virtual channel 0 transmit side

The medium access controller allows the assignment of virtual channels to specific timeslots. *vc_tx_timeslot* is a 6-bit wide unsigned value defining the current timeslot number between 0 and 63.

Receive side

On port *vc_rx_data*, data is read out from the virtual channel input buffers. The output vectors are 32-bit wide and concatenated, i.e. the data word for virtual channel 0 is stored at bit positions 31:0, the

data word for virtual channel 1 is stored at bit positions 63:32 and so on. The lowest byte is the one received first. On port *vc_rx_kflags*, a 4-bit vector for each virtual channel defines which bytes of the received data word are k-codes. The vectors are concatenated as well, i.e. the k-flags vector for virtual channel 0 is stored at bit positions 3:0, the k-flags vector for virtual channel 1 is stored at bit positions 7:4 and so on. The lowest bit in the k-flags vector corresponds to the lowest byte of the data word. The same k-codes as described in table 1235 can occur, i.e. EOP, EEP and FILL characters.

To read from a virtual channel receive buffer, the read enable flag for the virtual channel in bit vector *vc_rx_ren* must be asserted. The bit position corresponds to the virtual channel number. Data on *vc_rx_data* is only valid if the corresponding valid flag in bit vector *vc_rx_valid* is asserted. Again, the bit position corresponds to the virtual channel number. This FIFO scheme makes it particularly easy to interface other FIFO-like interfaces in the user logic. For instance, one could loopback the received data to the transmit side by simply connecting *vc_tx_data* to *vc_rx_data*, *vc_tx_kflags* to *vc_rx_kflags*, *vc_tx_wen* to *vc_rx_valid* and *vc_rx_ren* to not(*vc_tx_full*). An example timing diagram is shown in figure 185.

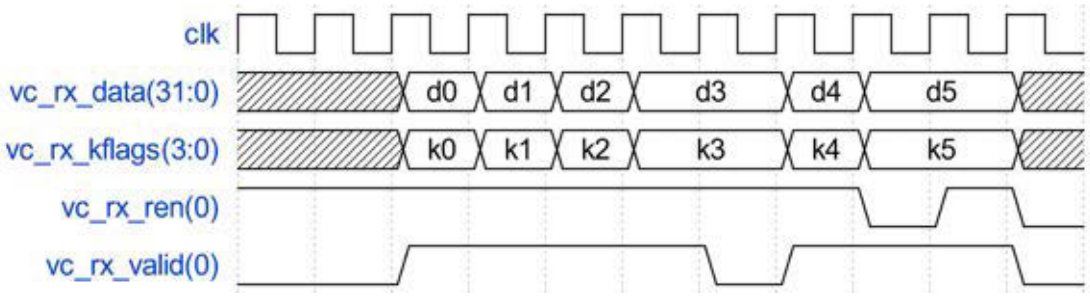


Figure 185. Example timing diagram - virtual channel 0 receive side

75.2.4 Broadcast interface

Transmit side

A broadcast delivers 8 bytes of payload, which are fed into the SpaceFibre IP core on port *bc_tx_data*. The lowest byte is transmitted first. Each broadcast has a channel and sequence number, which can be fed in at port *bc_tx_channel* and *bc_tx_seq* respectively. The write enable flag *bc_tx_wen* must be asserted to write the broadcast. Once the flag is set, an internal state machine prepares and executes the transmission of the broadcast frame. During this process the broadcast data, channel and sequence number must be kept stable. The SpaceFibre IP core signals the end of the transmission by pulsing the active-high flag bc_tx_ack for one clock cycle. An example timing diagram is shown in figure 186.

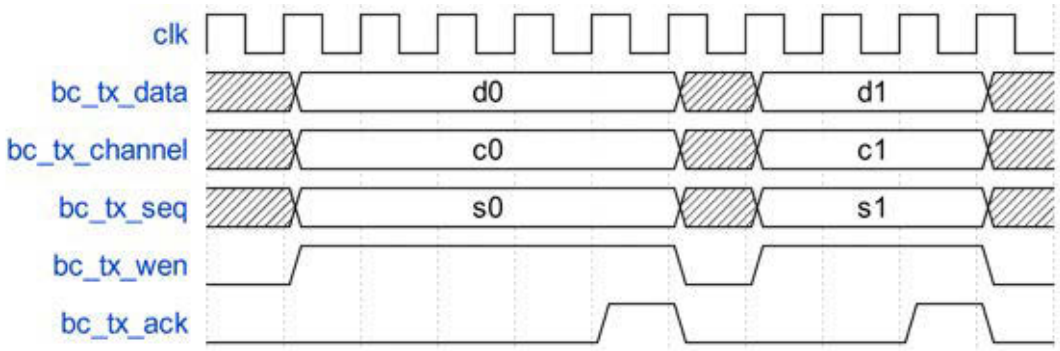


Figure 186. Example timing diagram - broadcast channel transmit side

Receive side

On the receive side, broadcast payload data is available on port *bc_rx_data*, the broadcast channel number on port *bc_rx_channel* and the broadcast sequence number on port *bc_rx_seq*. If the broadcast was sent during a retry, i.e. if the broadcast is unusually delayed, the flag *bc_rx_late* is high. Data

on the aforementioned ports is only valid when the active-high flag *bc_rx_valid* is pulsed for one clock-cycle. From a user's perspective, this flag can be used as a write enable signal for some sort of memory to store the broadcast data for further processing.

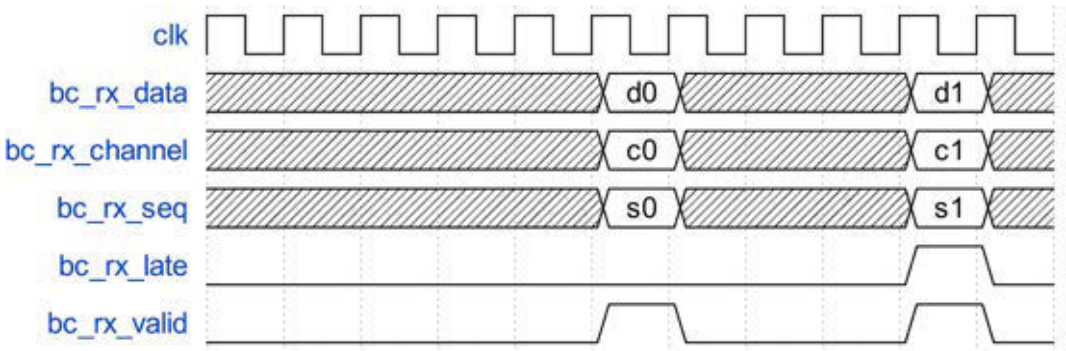


Figure 187. Example timing diagram - broadcast channel receive side

75.2.5 Serdes interface

Transmit

The internal data path width of the SpaceFibre IP core is 32 bits. The SpaceFibre IP core allows several different configurations:

- Large SerDes interface (*sel_16_20_bit_mode* is false), internal 8B10B encoding is activated (*use_8b10b* is true): In this configuration, 8B10B encoded 40-bit wide transmit data is outputted at *se_tx_data*(39:0). *se_tx_kflags*(3:0) is unused and can be left open.
- Large SerDes interface (*sel_16_20_bit_mode* is false), internal 8B10B encoding is deactivated (*use_8b10b* is false): In this configuration, unencoded 32-bit wide transmit data is outputted at *se_tx_data*(31:0). *se_tx_kflags*(3:0) is active and must be connected to the external 8B10B encoder, e.g. within the SerDes device.
- Small SerDes interface (*sel_16_20_bit_mode* is true), internal 8B10B encoding is activated (*use_8b10b* is true): In this configuration, 8B10B encoded 20-bit wide transmit data is outputted at *se_tx_data*(19:0). *se_tx_kflags*(3:0) is unused and can be left open.
- Small SerDes interface (*sel_16_20_bit_mode* is true), internal 8B10B encoding is deactivated (*use_8b10b* is false): In this configuration, unencoded 16-bit wide transmit data is outputted at *se_tx_data*(15:0). *se_tx_kflags*(1:0) is active and must be connected to the external 8B10B encoder, e.g. within the SerDes device.

If the lane layer of the SpaceFibre IP core is deactivated, the flag *se_tx_en* is low. It can be used to switch off the transmit side of the SerDes to save power during idle times.

For debug purposes, the transmit data and k-flags before any possible data path width conversion and 8B10B encoding can be monitored at *se_tx_data_dbg* and *se_tx_kflags_dbg*.

Receive

Similarly to the transmit side, the SerDes interface is used differently depending on the configuration:

- Large SerDes interface (*sel_16_20_bit_mode* is false), internal 8B10B decoding is activated (*use_8b10b* is true): In this configuration, 8B10B encoded 40-bit wide receive data is fed in on port *se_rx_data*(39:0). *se_rx_kflags*(3:0) is unused and can be tied to ground.
- Large SerDes interface (*sel_16_20_bit_mode* is false), internal 8B10B encoding is deactivated (*use_8b10b* is false): In this configuration, unencoded 32-bit wide receive data is fed in on port *se_rx_data*(31:0). *se_rx_kflags*(3:0) is active and must be connected to the external 8B10B decoder, e.g. within the SerDes device.

- Small SerDes interface (*sel_16_20_bit_mode* is true), internal 8B10B encoding is activated (*use_8b10b* is true): In this configuration, 8B10B encoded 20-bit wide receive data is fed in on port *se_rx_data*(19:0). *se_rx_data*(39:20) should be tied to ground. *se_rx_kflags*(3:0) is unused and can be tied to ground as well.
- Small SerDes interface (*sel_16_20_bit_mode* is true), internal 8B10B encoding is deactivated (*use_8b10b* is false): In this configuration, unencoded 16-bit wide receive data is fed in on port *se_rx_data*(15:0). *se_rx_kflags*(1:0) is active and must be connected to the external 8B10B encoder, e.g. within the SerDes device. *se_rx_data*(39:16) and *se_rx_kflags*(3:2) are unused and can be tied to ground.

If internal 8B10B decoding is not activated and the SerDes provides byte related error flags, for instance for disparity errors, these can be connected to the *se_rx_serror* port, with the lowest bit corresponding to the lowest byte of the receive data. If this is not the case, this port should be tied to ground. If the received data from the SerDes device is not synchronous to the SpaceFibre clock clk, the recovered SerDes receive clock must be connected to port *se_rx_clk*. If the data is already synchronous, for instance because the SerDes comprises its own elastic buffer, the SpaceFibre clock clk must be connected to *se_rx_clk*. If the lane layer of the SpaceFibre IP core is deactivated, the flag *se_rx_en* is low. It can be used to switch off the receive side of the SerDes to save power during idle times. If the SerDes provides a “No Signal” flag, it should be connected to port *se_no_signal*, otherwise this port must be tied to ground. If internal 8B10 decoding is not activated and the SerDes provides the capability to invert incoming bits, port *se_inv_pol* can be connected to the corresponding control port of the SerDes. If not, this port can be left unconnected.

For debug purposes, the receive data and k-flags after any possible data path width conversion and 8B10B decoding can be monitored at *se_rx_data_dbg* and *se_rx_kflags_dbg*.

75.3 Registers

There are no user accessible registers in the core. It is suggested to connect the configuration input signals (CF_*) and the status output signals (ST_*) to registers outside the core.

75.4 Vendor and device identifier

The vendor and device identifiers are only applicable for cores with AHB interfaces.

75.5 Implementation

75.5.1 Reset

The core changes reset behavior depending on settings in the GRLIB configuration package (see GRLIB User’s Manual). By default, the core makes use of synchronous reset and resets all its internal registers.

The core will use asynchronous reset for all registers if the GRLIB config package setting *grlib_async_reset_enable* is set.

75.6 Configuration options

Table 1236 shows the configuration options of the core (VHDL generics).

*Table 1236.*Configuration options

Generic name	Function	Allowed range	Default
tech	This generic can be used for technology-specific internal components such as memories.	0 - NTECH	inferred
use_8b10b	If set, internal 8B10B encoding and decoding is activated.	0 - 1	1
use_sep_txclk	If set, the SerDes transmission clock is decoupled from the SpaceFibre port clock. An additional transmit buffer is instantiated for this reason.	0 - 1	0
sel_16_20_bit_mode	If set, the SerDes interface is 16+2 bit (without 8B10B) or 20 bit (with 8B10B) wide instead of 36/40 bit. If set, use_sep_txclk must also be set.	0 - 1	0
ticks_2us	Clock ticks corresponding to 2 µs.	1 - 8192	125
tx_skip_freq	Frequency of SKIP word transmission in clock cycles.	1 - 8192	5000
prbs_init1	If set, the INIT1 sequence during lane initialisation is embedded into a stream of pseudo-random numbers.	0 - 1	1
depth_rbuf_data	Log(Depth) of the data retry buffer.	1 - 32	8
depth_rbuf_fct	Log(Depth) of the FCT retry buffer.	1 - 32	4
depth_rbuf_bc	Log(Depth) of the broadcast retry buffer.	1 - 32	8
num_vc	Number of virtual channels.	1 - 32	4
fct_multiplier	Multiplier used in the internal FCT logic. The value of the FCT counter is compared with 64**fct_multiplier* to decide if the far end can receive new words.	1 - 8	1
depth_vc_rx_buf	Log(Depth) of virtual channel input buffer.	1 - 32	10
depth_vc_tx_buf	Log(Depth) of virtual channel output buffer.	1 - 32	10
remote_fct_cnt_max	Width of the remote FCT counter.	0 - 32	9
width_bw_credit	Width of the bandwidth credit counter.	0 - 16#FFFFFFFF#	20
min_bw_credit	Minimum bandwidth credit threshold limit.	0 - 16#FFFFFFFF#	52428
idle_time_limit	Bandwidth idle time limit in clock cycles.	0 - 65535	62500
use_async_rxrst	Force asynchronous reset in the recovered clock domain, regardless of the general GRLIB settings	0 - 1	0
ft_vc	Enable fault-tolerance against SEU errors for the FIFOs in the Virtual Channel Layer. Possible options are byte parity protection (ft = 1), TMR registers (ft = 2), SECDED BCH (ft = 4) or technology specific protection (ft = 5). If set to 0, no protection is implemented.	0 - 2, 4 - 5	0
ft_rt1	Enable fault-tolerance against SEU errors for the large FIFOs in the Retry Layer (Data and Broadcast). The core supports the same configurations as above.	0 - 2, 4 - 5	0
ft_rt2	Enable fault-tolerance against SEU errors for the small FIFOs in the Retry Layer (FCT and pointers). The core supports the same configurations as above.	0 - 2, 4 - 5	0
ft_if	Enable fault-tolerance against SEU errors for the FIFOs in the Interface Layer. The core supports the same configurations as above.	0 - 2, 4 - 5	0
scantest	Enable Scan Test support	0 - 1	0

75.7 Signal Descriptions

Table 1237 shows the interface signals of the core (VHDL ports).

*Table 1237.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
TX_CLK	N/A	Input	Optional transmit clock	-
SE_RX_CLK	N/A	Input	SerDes receive clock	-
CF_RESET	N/A	Output	Active-high configuration register reset	High
VC_TX_DATA	N/A	Input	Virtual Channel Transmit Data	-
VC_TX_KFLAGS	N/A	Input	Virtual Channel Transmit K-Flags	-
VC_TX_WEN	N/A	Input	Virtual Channel Write Enable	High
VC_TX_FULL	N/A	Output	Virtual Channel Full Flags	High
VC_TX_TIMESLOT	N/A	Input	Current timeslot	-
VC_RX_DATA	N/A	Output	Virtual Channel Receive Data	-
VC_RX_KFLAGS	N/A	Output	Virtual Channel Receive K-Flags	-
VC_RX_VALID	N/A	Output	Virtual Channel Valid Flags	High
VC_RX_REN	N/A	Input	Virtual Channel Read Enable Flags	High
BC_TX_DATA	N/A	Input	Broadcast Transmit Data	-
BC_TX_CHANNEL	N/A	Input	Broadcast Transmit Channel	-
BC_TX_BTYPE	N/A	Input	Broadcast Transmit Type	-
BC_TX_LATE	N/A	Input	Broadcast Transmit Late Flag	High
BC_TX_DELAYED	N/A	Input	Broadcast Transmit Delayed Flag	High
BC_TX_WEN	N/A	Input	Broadcast Transmit Write Enable Flag	High
BC_TX_ACK	N/A	Output	Broadcast Transmit Acknowledgement Flag	High
BC_RX_DATA	N/A	Output	Broadcast Receive Data	-
BC_RX_CHANNEL	N/A	Output	Broadcast Receive Channel	-
BC_RX_BTYPE	N/A	Output	Broadcast Receive Type	-
BC_RX_VALID	N/A	Output	Broadcast Receive Valid Flag	High
BC_RX_LATE	N/A	Output	Broadcast Receive Late Flag	High
BC_RX_DELAYED	N/A	Output	Broadcast Receive Delayed Flag	High
SE_TX_DATA	N/A	Output	Transmit Data	-
SE_TX_KFLAGS	N/A	Output	Transmit K-Flags	-
SE_TX_EN	N/A	Output	SerDes Transmitter Enable Flag	High
SE_TX_DATA_DBG	N/A	Output	Unencoded 32-bit transmit data	-
SE_TX_KFLAGS_DBG	N/A	Output	4-bit K-Flags	-
SE_RX_DATA	N/A	Input	Receive Data	-
SE_RK_KFLAGS	N/A	Input	Receive K-Flags	-
SE_RX_SERROR	N/A	Input	Receive Error Flags	High
SE_RX_EN	N/A	Output	SerDes Receiver Enable Flag	High
SE_NO_SIGNAL	N/A	Input	SerDes No Signal Flag	High
SE_INV_POL	N/A	Output	SerDes Invert Polarity Flag	High
SE_RX_DATA_DBG	N/A	Output	Unencoded 32-bit receive data	-
SE_RX_KFLAGS_DBG	N/A	Output	4-bit K-Flags	-
CF_LINK_RST	N/A	Input	Active-high link reset	High

Table 1237. Signal descriptions

Signal name	Field	Type	Function	Active
CF_LANE_START	N/A	Input	Lane Start flag of the lane initialisation state machine	High
CF_AUTO_START	N/A	Input	Autostart flag of the lane initialisation state machine	High
CF_LANE_RST	N/A	Input	Active-high lane reset	High
CF_LOOPBACK	N/A	Input	Activates/deactivates internal loopback	
CF_STANDBY_REASON	N/A	Input	Standby Reason included in the STANDBY control words generated by the core	
CF_DATA_SCR_EN	N/A	Input	Activates/deactivates data scrambler on transmission side	
CF_BC_EXPECTED_BW	N/A	Input	Expected broadcast bandwidth value	-
CF_VC_PRIORITIES	N/A	Input	Priority values for the virtual channels	-
CF_VC_EXPECTED_BW	N/A	Input	Expected virtual channel bandwidth value	-
CF_VC_TSLOT_VECS	N/A	Input	Timeslot vectors for the virtual channels	-
SR_FAR_END_LRST	N/A	Output	Pulsed high when the far end triggered a link reset	High
SR_LANE_STATE	N/A	Output	Indicates the current state of the lane initialisation state machine	-
SR_RXERR_COUNT	N/A	Output	Counts the number of RXERR words	-
SR_RXERR_COUNT_OF	N/A	Output	Pulsed high when the RXERR counter overflows	High
SR_FAR_END_-STANDBY	N/A	Output	Pulsed high when STANDBY words are received	High
SR_TIMEOUT	N/A	Output	Pulsed high when a connection timeout during lane initialisation has occurred	High
SR_FAR_END_LOS	N/A	Output	Pulsed high when LOS words are received	High
SR_FAR_END_LOS_-CAUSE	N/A	Output	Loss of Signal Cause field received from the far end	-
SR_FAR_END_CAP	N/A	Output	Far-end capabilities field received in INIT3 words	
SR_RX_POLARITY	N/A	Output	Set when the SerDes receiver polarity is inverted	High
SR_CRC16_ERR	N/A	Output	Pulsed high when a CRC-16 error has occurred	High
SR_FRAME_ERR	N/A	Output	Pulsed high when a frame error has occurred	High
SR_CRC8_ERR	N/A	Output	Pulsed high when a CRC-8 error has occurred	High
SR_SEQ_ERR	N/A	Output	Pulsed high when a sequence error has occurred	High
SR_RBUF_EMPTY	N/A	Output	Set when the error recovery buffer is empty	High
SR_RETRY_CNT	N/A	Output	The number of error recovery attempts made by the SpaceFibre port	-
SR_TOO_MANY_ERR	N/A	Output	Set when the RETRY counter has overflown (i.e. the link is automatically reset).	High
SR_PROTOCOL_ERR	N/A	Output	Set when an ACK / NACK error has been detected.	High
SR_BW_OVER_USE	N/A	Output	Set when a virtual channel is using much more bandwidth than expected	High
SR_BW_UNDER_USE	N/A	Output	Set when a virtual channel is using much less bandwidth than expected	High
SR_DEST_HAS_CREDIT	N/A	Output	Set when there is space in the input buffer of the destination node	High
SR_INPUT_BUF_OV	N/A	Output	Set when an input buffer is receiving data when it is full	High

Table 1237. Signal descriptions

Signal name	Field	Type	Function	Active
SR_FCT_CNT_OV	N/A	Output	Pulsed high when the FCT credit counter of a virtual channel has overflowed	High
TESTEN	N/A	Input	Test enable. When GRLIB is configured to use asynchronous reset, TESTEN selects if the internal signal should come from RST or TESTRST	High
TESTRST	N/A	Input	Reset signal in test mode (TESTEN = High). Only used when GRLIB is configured to use asynchronous reset.	Low
TESTIN	N/A	Input	Test input vector for internal memories	-
MTESTI**	VC_TX	Input	Memory BIST input signal to the VC TX FIFOs	-
	VC_RX	Input	Memory BIST input signal to the VC RX FIFOs	-
	RETR_BC	Input	Memory BIST input signal to the Broadcast FIFO in the Retry Layer	-
	RETR_FCT	Input	Memory BIST input signal to the FCT FIFO in the Retry Layer	-
	RETR_DAT	Input	Memory BIST input signal to the Data FIFO in the Retry Layer	-
	RETR_PTR	Input	Memory BIST input signal to the Pointers FIFO in the Retry Layer	-
	IF_TX	Input	Memory BIST input signal to the TX FIFO in the Interface Layer	-
	IF_RX	Input	Memory BIST input signal to the RX FIFO in the Interface Layer	-
MTESTO**	VC_TX	Output	Memory BIST output signal from the VC TX FIFOs	-
	VC_RX	Output	Memory BIST output signal from the VC RX FIFOs	-
	RETR_BC	Output	Memory BIST output signal from the Broadcast FIFO in the Retry Layer	-
	RETR_FCT	Output	Memory BIST output signal from the FCT FIFO in the Retry Layer	-
	RETR_DAT	Output	Memory BIST output signal from the Data FIFO in the Retry Layer	-
	RETR_PTR	Output	Memory BIST output signal from the Pointers FIFO in the Retry Layer	-
	IF_TX	Output	Memory BIST output signal from the TX FIFO in the Interface Layer	-
	IF_RX	Output	Memory BIST output signal from the RX FIFO in the Interface Layer	-
MTESTCLK**	N/A	Input	Memory BIST clock	-

* see GRLIB IP Library User's Manual

** not available in FPGA releases

75.8 Library dependencies

Table 1238 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1238.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
SPFI	SPFICOMP	Signals, component	Component declaration

75.9 Instantiation

This example shows how the core can be instantiated.

```
grspfi_codec0: grspfi_codec
  generic map (
    tech                => inferred,
    use_8b10b           => 1,
    use_sep_txclk       => 0,
    sel_16_20_bit_mode  => 0,
    ticks_2us           => 125,
    tx_skip_freq        => 5000,
    prbs_init1          => 1,
    depth_rbuf_data     => 8,
    depth_rbuf_fct      => 4,
    depth_rbuf_bc       => 8,
    num_vc              => 4,
    fct_multiplier      => 1,
    depth_vc_rx_buf     => 8,
    depth_vc_tx_buf     => 8,
    remote_fct_cnt_max  => 8,
    width_bw_credit     => 20,
    min_bw_credit       => 52428,
    idle_time_limit     => 62500)
port map (
    clk                   => clk,
    tx_clk                => tx_clk,
    rst                   => rst,
    vc_tx_full            => vc_tx_full,
    vc_rx_data            => vc_rx_data,
    vc_rx_kflags          => vc_rx_kflags,
    vc_rx_valid           => vc_rx_valid,
    vc_rx_ren             => vc_rx_ren,
    vc_tx_data            => vc_tx_data,
    vc_tx_kflags          => vc_tx_kflags,
    vc_tx_wen             => vc_tx_wen,
    vc_tx_timeslot        => vc_tx_timeslot,
    bc_rx_data            => bc_rx_data,
    bc_rx_channel         => bc_rx_channel,
    bc_rx_btype           => bc_rx_btype,
    bc_rx_valid           => bc_rx_valid,
    bc_rx_late            => bc_rx_late,
    bc_rx_delayed         => bc_rx_delayed,
    bc_tx_ack             => bc_tx_ack,
    bc_tx_data            => bc_tx_data,
    bc_tx_channel         => bc_tx_channel,
    bc_tx_btype           => bc_tx_btype,
    bc_tx_late            => bc_tx_late,
    bc_tx_delayed         => bc_tx_delayed,
    bc_tx_wen             => bc_tx_wen,
    se_tx_data            => se_tx_data,
    se_tx_kflags          => se_tx_kflags,
    se_tx_en              => se_tx_en,
    se_rx_en              => se_rx_en,
    se_inv_pol            => se_inv_pol,
    se_rx_clk             => se_rx_clk,
```

```
se_rx_data           => se_rx_data,
se_rx_kflags         => se_rx_kflags,
se_rx_serror         => se_rx_serror,
se_no_signal         => se_no_signal,
se_tx_data_dbg       => se_tx_data_dbg,
se_tx_kflags_dbg     => se_tx_kflags_dbg,
se_rx_data_dbg       => se_rx_data_dbg,
se_rx_kflags_dbg     => se_rx_kflags_dbg,
cf_reset             => cf_reset,
cf_vc_priorities     => cf_vc_priorities,
cf_vc_expected_bw    => cf_vc_expected_bw,
cf_vc_tslot_vecs     => cf_vc_tslot_vecs,
cf_bc_expected_bw    => cf_bc_expected_bw,
cf_data_scr_en       => cf_data_scr_en,
cf_link_rst          => cf_link_rst,
cf_lane_start        => cf_lane_start,
cf_auto_start        => cf_auto_start,
cf_lane_rst          => cf_lane_rst,
cf_loopback          => cf_loopback,
cf_standby_reason    => cf_standby_reason,
sr_bw_over_use       => sr_bw_over_use,
sr_bw_under_use      => sr_bw_under_use,
sr_dest_has_credit   => sr_dest_has_credit,
sr_input_buf_ov      => sr_input_buf_ov,
sr_fct_cnt_ov        => sr_fct_cnt_ov,
sr_crc16_err         => sr_crc16_err,
sr_frame_err         => sr_frame_err,
sr_crc8_err          => sr_crc8_err,
sr_seq_err           => sr_seq_err,
sr_rbuf_empty        => sr_rbuf_empty,
sr_retry_cnt         => sr_retry_cnt,
sr_too_many_err      => sr_too_many_err,
sr_protocol_err      => sr_protocol_err,
sr_far_end_lrst      => sr_far_end_lrst,
sr_lane_state        => sr_lane_state,
sr_rxerr_count       => sr_rxerr_count,
sr_rxerr_count_of    => sr_rxerr_count_of,
sr_far_end_standby   => sr_far_end_standby,
sr_timeout           => sr_timeout,
sr_far_end_los       => sr_far_end_los,
sr_far_end_los_cause => sr_far_end_los_cause,
sr_far_end_cap       => sr_far_end_cap,
sr_rx_polarity       => sr_rx_polarity);
```

76 GRSPW - SpaceWire codec with AHB host Interface and RMAP target

76.1 Overview

The SpaceWire core provides an interface between the AHB bus and a SpaceWire network. It implements the SpaceWire standard (ECSS-E-ST-50-12C) with the protocol identification extension (ECSS-E-ST-50-51C). The optional Remote Memory Access Protocol (RMAP) target implements the ECSS standard (ECSS-E-ST-50-52C).

The core is configured through a set of registers accessed through an APB interface. Data is transferred through DMA channels using an AHB master interface.

Currently, there is one DMA channel but the core can easily be extended to use separate DMA channels for specific protocols. The core can also be configured to have either one or two ports.

There can be up to four clock domains: one for the AHB interface (system clock), one for the transmitter and one or two for the receiver depending on the number of configured ports. The receiver clock can be twice as fast and the transmitter clock four times as fast as the system clock whose frequency should be at least 10 MHz.

The core only supports byte addressed 32-bit big-endian host systems.

Note: This IP core is deprecated and should not be used for new designs. Please see the GRSPW2 IP core instead.

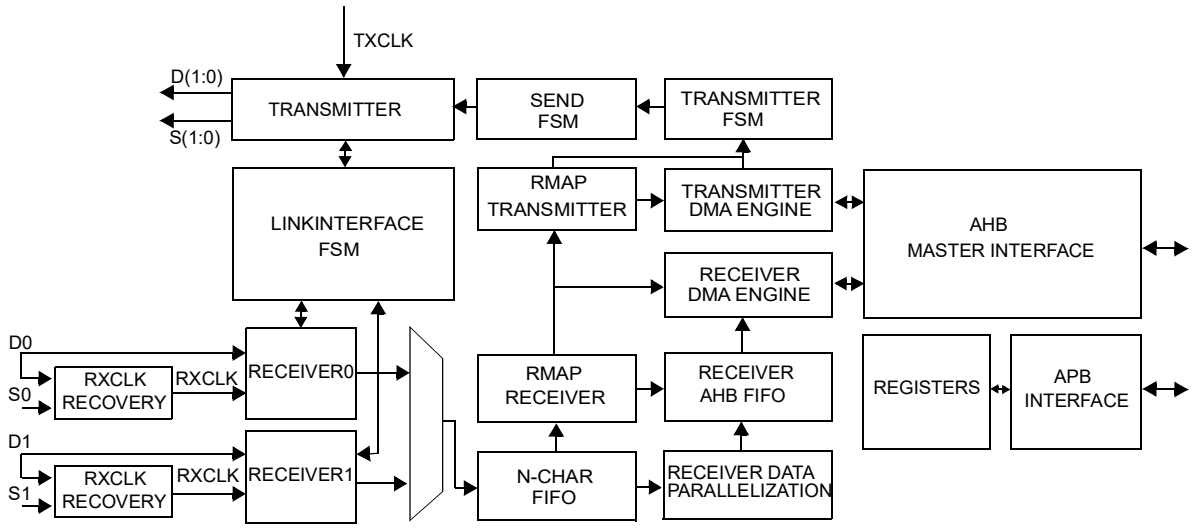


Figure 188. Block diagram

76.2 Operation

76.2.1 Overview

The main sub-blocks of the core are the link-interface, the RMAP target and the AMBA interface. A block diagram of the internal structure can be found in figure 188.

The link interface consists of the receiver, transmitter and the link interface FSM. They handle communication on the SpaceWire network. The AMBA interface consists of the DMA engines, the AHB master interface and the APB interface. The link interface provides FIFO interfaces to the DMA engines. These FIFOs are used to transfer N-Chars between the AMBA and SpaceWire domains during reception and transmission.

The RMAP target is an optional part of the core which can be enabled with a VHDL generic. The RMAP target handles incoming packets which are determined to be RMAP commands instead of the

receiver DMA engine. The RMAP command is decoded and if it is valid, the operation is performed on the AHB bus. If a reply was requested it is automatically transmitted back to the source by the RMAP transmitter.

The core is controlled by writing to a set of user registers through the APB interface and three signals: tick-in, rmapen and clkdiv10. The controlled parts are clock-generation, DMA engines, RMAP target and the link interface.

The link interface, DMA engines, RMAP target and AMBA interface are described in section 76.3, 76.4, 76.5 and 76.7 respectively.

76.2.2 Protocol support

The core only accepts packets with a destination address corresponding to the one set in the node address register. Packets with address mismatch will be silently discarded (except in promiscuous mode which is covered in section 76.4.10). The node address register is initialized to the default address 254 during reset. It can then be changed to other values by writing to the register.

The core has support for the protocol ID specified in ECSS-E-ST-50-51C. It is used for identifying RMAP commands that should be received by the RMAP target. Other packets are stored to the DMA channel. This is only applicable if the RMAP target is present and enabled. When the RMAP target is not present or disabled all the bytes after the address are treated as normal cargo.

RMAP commands are identified using the protocol ID (0x01) and the instruction field. They are handled separately from other packets if the hardware RMAP target is enabled. When enabled, all RMAP commands are processed, executed and replied in hardware. RMAP replies received are always stored to the DMA channel. If the RMAP target is disabled, all packets are stored to the DMA channel. More information on the RMAP protocol support is found in section 76.7.

RMAP packets arriving with the extended protocol ID (0x000001) are stored to the DMA channel which means that the hardware RMAP target will not work if the incoming RMAP packets use the extended protocol ID. Note also that packets with the reserved extended protocol identifier (ID = 0x000000) are not ignored by the core. It is up to the client receiving the packets to ignore them.

When transmitting packets, the address and protocol-ID fields must be included in the buffers from where data is fetched. They are *not* automatically added by the core.

Figure 189 shows the packet types supported by the core. The core also allows reception and transmission with extended protocol identifiers but without support for RMAP CRC calculations and the RMAP target.

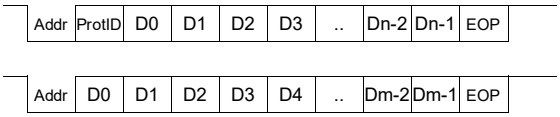


Figure 189. The SpaceWire packet types supported by the GRSPW.

76.3 Link interface

The link interface handles the communication on the SpaceWire network and consists of a transmitter, receiver, a FSM and FIFO interfaces. An overview of the architecture is found in figure 188.

76.3.1 Link interface FSM

The FSM controls the link interface (a more detailed description is found in the SpaceWire standard). The low-level protocol handling (the signal and character level of the SpaceWire standard) is handled by the transmitter and receiver while the FSM in the host domain handles the exchange level.

The link interface FSM is controlled through the control register. The link can be disabled through the link disable bit, which depending on the current state, either prevents the link interface from reaching the started state or forces it to the error-reset state. When the link is not disabled, the link interface FSM is allowed to enter the started state when either the link start bit is set or when a NULL character has been received and the autostart bit is set.

The current state of the link interface determines which type of characters are allowed to be transmitted which together with the requests made from the host interfaces determine what character will be sent.

Time-codes are sent when the FSM is in the run-state and a request is made through the time-interface (described in section 76.3.5).

When the link interface is in the connecting- or run-state it is allowed to send FCTs. FCTs are sent automatically by the link interface when possible. This is done based on the maximum value of 56 for the outstanding credit counter and the currently free space in the receiver N-Char FIFO. FCTs are sent as long as the outstanding counter is less than or equal to 48 and there are at least 8 more empty FIFO entries than the counter value.

N-Chars are sent in the run-state when they are available from the transmitter FIFO and there are credits available. NULLs are sent when no other character transmission is requested or the FSM is in a state where no other transmissions are allowed.

The credit counter (incoming credits) is automatically increased when FCTs are received and decreased when N-Chars are transmitted. Received N-Chars are stored to the receiver N-Char FIFO for further handling by the DMA interface. Received Time-codes are handled by the time-interface.

76.3.2 Transmitter

The state of the FSM, credit counters, requests from the time-interface and requests from the DMA-interface are used to decide the next character to be transmitted. The type of character and the character itself (for N-Chars and Time-codes) to be transmitted are presented to the low-level transmitter which is located in a separate clock-domain.

This is done because one usually wants to run the SpaceWire link on a different frequency than the host system clock. The core has a separate clock input which is used to generate the transmitter clock. More information on transmitter clock generation is found in section 76.8.2. Since the transmitter often runs on high frequency clocks (> 100 MHz) as much logic as possible has been placed in the system clock domain to minimize power consumption and timing issues.

The transmitter logic in the host clock domain decides what character to send next and sets the proper control signal and presents any needed character to the low-level transmitter as shown in figure 190. The transmitter sends the requested characters and generates parity and control bits as needed. If no requests are made from the host domain, NULLs are sent as long as the transmitter is enabled. Most of the signal and character levels of the SpaceWire standard is handled in the transmitter. External LVDS drivers are needed for the data and strobe signals.

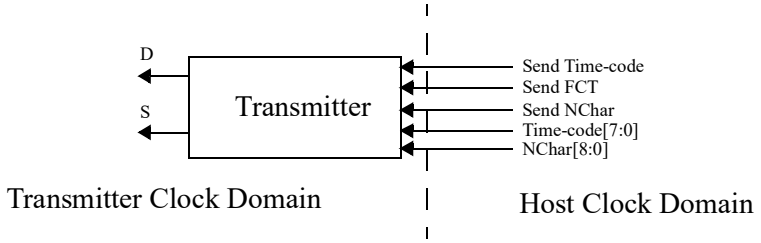


Figure 190. Schematic of the link interface transmitter.

A transmission FSM reads N-Chars for transmission from the transmitter FIFO. It is given packet lengths from the DMA interface and appends EOPs/EEPs and RMAP CRC values if requested. When it is finished with a packet the DMA interface is notified and a new packet length value is given.

76.3.3 Receiver

The receiver detects connections from other nodes and receives characters as a bit stream on the data and strobe signals. It is also located in a separate clock domain which runs on a clock generated from the received data and strobe signals. More information on the clock-generation can be found in section 76.8.2.

The receiver is activated as soon as the link interface leaves the error reset state. Then after a NULL is received it can start receiving any characters. It detects parity, escape and credit errors which causes the link interface to enter the error reset state. Disconnections are handled in the link interface part in the system clock domain because no receiver clock is available when disconnected.

Received Characters are flagged to the host domain and the data is presented in parallel form. The interface to the host domain is shown in figure 191. L-Chars are the handled automatically by the host domain link interface part while all N-Chars are stored in the receiver FIFO for further handling. If two or more consecutive EOPs/EEPs are received all but the first are discarded.

There are no signals going directly from the transmitter clock domain to the receiver clock domain and vice versa. All the synchronization is done to the system clock.

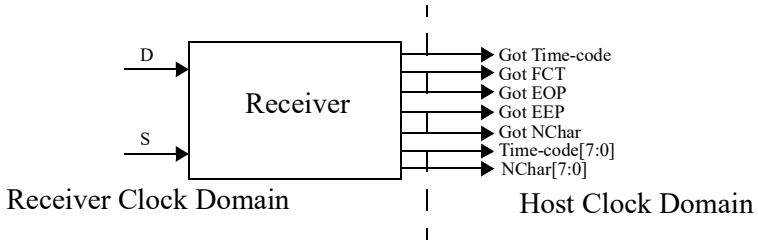


Figure 191. Schematic of the link interface receiver.

76.3.4 Dual port support

The core can be configured to include an additional SpaceWire port. With dual ports the transmitter drives an additional pair of data/strobe output signals and one extra receiver is added to handle a second pair of data/strobe input signals.

One of the ports is set as active (how the active port is selected is explained below) and the transmitter drives the data/strobe signals of the active port with the actual output values as explained in section 76.3.2. The inactive port is driven with zero on both data and strobe.

Both receivers will always be active but only the active port's interface signals (see figure 191) will be propagated to the link interface FSM. Each time the active port is changed, the link will be reset so that the new link is started in a controlled manner.

When the noportforce register is zero the portsel register bit selects the active link and when set to one it is determined by the current link activity. In the latter mode the port is changed when no activity is seen on the currently active link while there is activity on the deselected receive port. Activity is defined as a detected null. This definition is selected so that glitches (e.g. port unconnected) do not cause unwanted port switches.

76.3.5 Time interface

The time interface is used for sending Time-codes over the SpaceWire network and consists of a time-counter register, time-ctrl register, tick-in signal, tick-out signal, tick-in register field and a tick-out

register field. There are also two control register bits which enable the time receiver and transmitter respectively.

Each Time-code sent from the core is a concatenation of the time-ctrl and the time-counter register. There is a timetxen bit which is used to enable Time-code transmissions. It is not possible to send time-codes if this bit is zero.

Received Time-codes are stored to the same time-ctrl and time-counter registers which are used for transmission. The timerxen bit in the control register is used for enabling time-code reception. No time-codes will be received if this bit is zero.

The two enable bits are used for ensuring that a node will not (accidentally) both transmit and receive time-codes which violates the SpaceWire standard. It also ensures that a the master sending time-codes on a network will not have its time-counter overwritten if another (faulty) node starts sending time-codes.

The time-counter register is set to 0 after reset and is incremented each time the tick-in signal is asserted for one clock-period and the timetxen bit is set. This also causes the link interface to send the new value on the network. Tick-in can be generated either by writing a one to the register field or by asserting the tick-in signal. A Tick-in should not be generated too often since if the time-code after the previous Tick-in has not been sent the register will not be incremented and no new value will be sent. The tick-in field is automatically cleared when the value has been sent and thus no new ticks should be generated until this field is zero. If the tick-in signal is used there should be at least 4 system-clock and 25 transmit-clock cycles between each assertion.

A tick-out is generated each time a valid time-code is received and the timerxen bit is set. When the tick-out is generated the tick-out signal will be asserted one clock-cycle and the tick-out register field is asserted until it is cleared by writing a one to it.

The current time counter value can be read from the time register. It is updated each time a Time-code is received and the timerxen bit is set. The same register is used for transmissions and can also be written directly from the APB interface.

The control bits of the Time-code are always stored to the time-ctrl register when a Time-code is received whose time-count is one more than the nodes current time-counter register. The time-ctrl register can be read through the APB interface. The same register is used during time-code transmissions.

It is possible to have both the time-transmission and reception functions enabled at the same time.

76.4 Receiver DMA engine

The receiver DMA engine handles reception of data from the SpaceWire network to different DMA channels. Currently there is only one receive DMA channel available but the core has been written so that additional channels can be easily added if needed.

76.4.1 Basic functionality

The receiver DMA engine reads N-Chars from the N-Char FIFO and stores them to a DMA channel. Reception is based on descriptors located in a consecutive area in memory that hold pointers to buffers where packets should be stored. When a packet arrives at the core it reads a descriptor from memory and stores the packet to the memory area pointed to by the descriptor. Then it stores status to the same descriptor and increments the descriptor pointer to the next one.

76.4.2 Setting up the core for reception

A few registers need to be initialized before reception can take place. First the link interface need to be put in the run state before any data can be sent. The DMA channel has a maximum length register which sets the maximum size of packet that can be received to this channel. Larger packets are truncated and the excessive part is spilled. If this happens an indication will be given in the status field of the descriptor. The minimum value for the receiver maximum length field is 4 and the value can only

be incremented in steps of four bytes. If the maximum length is set to zero the receiver will *not* function correctly.

The node address register needs to be set to hold the address of this SpaceWire node. Packets received with the incorrect address are discarded. Finally, the descriptor table and control register must be initialized. This will be described in the two following sections.

76.4.3 Setting up the descriptor table address

The core reads descriptors from an area in memory pointed to by the receiver descriptor table address register. The register consists of a base address and a descriptor selector. The base address points to the beginning of the area and must start on a 1 kbytes aligned address. It is also limited to be 1 kbytes in size which means the maximum number of descriptors is 128.

The descriptor selector points to individual descriptors and is increased by 1 when a descriptor has been used. When the selector reaches the upper limit of the area it wraps to the beginning automatically. It can also be set to wrap automatically by setting a bit in the descriptors. The idea is that the selector should be initialized to 0 (start of the descriptor area) but it can also be written with another 8 bytes aligned value to start somewhere in the middle of the area. It will still wrap to the beginning of the area.

If one wants to use a new descriptor table the receiver enable bit has to be cleared first. When the rxactive bit for the channel is cleared it is safe to update the descriptor table register. When this is finished and descriptors are enabled the receiver enable bit can be set again.

76.4.4 Enabling descriptors

As mentioned earlier one or more descriptors must be enabled before reception can take place. Each descriptor is 8 byte in size and the layout can be found in the tables below. The descriptors should be written to the memory area pointed to by the receiver descriptor table address register. When new descriptors are added they must always be placed after the previous one written to the area. Otherwise they will not be noticed.

A descriptor is enabled by setting the address pointer to point at a location where data can be stored and then setting the enable bit. The WR bit can be set to cause the selector to be set to zero when reception has finished to this descriptor. IE should be set if an interrupt is wanted when the reception has finished. The DMA control register interrupt enable bit must also be set for this to happen.

The descriptor packet address should be word aligned. All accesses on the bus are word accesses so complete words will always be overwritten regardless of whether all 32-bit contain received data. Also if the packet does not end on a word boundary the complete word containing the last data byte will be overwritten. If the *rxunaligned* or *rmap* VHDL generic is set to 1 this restriction is removed

and any number of bytes can be received to any packet address without excessive bytes being overwritten.

Table 1239. GRSPW receive descriptor word 0 (address offset 0x0)

31	30	29	28	27	26	25	24 ... 0
TR	DC	HC	EP	IE	WR	EN	PACKETLENGTH

31	Truncated (TR) - Packet was truncated due to maximum length violation.
30	Data CRC (DC) - 1 if a CRC error was detected for the data and 0 otherwise.
29	Header CRC (HC) - 1 if a CRC error was detected for the header and 0 otherwise.
28	EEP termination (EP) - This packet ended with an Error End of Packet character.
27	Interrupt enable (IE) - If set, an interrupt will be generated when a packet has been received if the receive interrupt enable bit in the DMA channel control register is set.
26	Wrap (WR) - If set, the next descriptor used by the GRSPW will be the first one in the descriptor table (at the base address). Otherwise the descriptor pointer will be increased with 0x8 to use the descriptor at the next higher memory location. The descriptor table is limited to 1 kbytes in size and the pointer will be automatically wrap back to the base address when it reaches the 1 kbytes boundary.
25	Enable (EN) - Set to one to activate this descriptor. This means that the descriptor contains valid control values and the memory area pointed to by the packet address field can be used to store a packet.
24: 0	Packet length (PACKETLENGTH) - The number of bytes received to this buffer. Only valid after EN has been set to 0 by the GRSPW.

Table 1240. GRSPW receive descriptor word 1 (address offset 0x4)

31 ... 0
PACKETADDRESS

31: 0	Packet address (PACKETADDRESS) - The address pointing at the buffer which will be used to store the received packet. If the rxunaligned and rmap VHDL generics are both set to zero only bit 31 to 2 are used.

76.4.5 Setting up the DMA control register

The final step to receive packets is to set the control register in the following steps: The receiver must be enabled by setting the rxen bit in the DMA control register (see section 76.9). This can be done anytime and before this bit is set nothing will happen. The rxdescav bit in the DMA control register is then set to indicate that there are new active descriptors. This must always be done after the descriptors have been enabled or the core might not notice the new descriptors. More descriptors can be activated when reception has already started by enabling the descriptors and writing the rxdescav bit. When these bits are set reception will start immediately when data is arriving.

76.4.6 The effect to the control bits during reception

When the receiver is disabled all packets going to the DMA-channel are discarded. If the receiver is enabled the next state is entered where the rxdescav bit is checked. This bit indicates whether there are active descriptors or not and should be set by the external application using the DMA channel each time descriptors are enabled as mentioned above. If the rxdescav bit is ‘0’ and the nospill bit is ‘0’ the packets will be discarded. If nospill is one the core waits until rxdescav is set.

When rxdescav is set the next descriptor is read and if enabled the packet is received to the buffer. If the read descriptor is not enabled, rxdescav is set to ‘0’ and the packet is spilled depending on the value of nospill.

The receiver can be disabled at any time and will cause all packets received afterwards to be discarded. If a packet is currently received when the receiver is disabled the reception will still be finished. The rxdescav bit can also be cleared at any time. It will not affect any ongoing receptions but

no more descriptors will be read until it is set again. Rxdescav is also cleared by the core when it reads a disabled descriptor.

76.4.7 Address recognition and packet handling

When the receiver N-Char FIFO is not empty, N-Chars are read by the receiver DMA engine. The first character is interpreted as the logical address which is compared to the node address register. If it does not match, the complete packet is discarded (up to and including the next EOP/EEP).

If the address matches the next action taken depends on whether RMAP is enabled or not. If RMAP is disabled all packets are stored to the DMA channel and depending on the conditions mentioned in the previous section, the packet will be received or not. If the packet is received the complete packet including address and protocol ID but excluding EOP/EEP is stored to the address indicated in the descriptor, otherwise the complete packet is discarded.

If RMAP is enabled the protocol ID and 3rd byte in the packet is first checked before any decisions are made. If incoming packet is an RMAP packet (ID = 0x01) and the command type field is 01b the packet is processed by the RMAP command handler which is described in section 76.6. Otherwise the packet is processed by the DMA engine as when RMAP is disabled.

At least 2 non EOP/EEP N-Chars need to be received for a packet to be stored to the DMA channel. If it is an RMAP packet 3 N-Chars are needed since the command byte determines where the packet is processed. Packets smaller than the minimum size are discarded.

76.4.8 Status bits

When the reception of a packet is finished the enable bit in the current descriptor is set to zero. When enable is zero, the status bits are also valid and the number of received bytes is indicated in the length field. The DMA control register contains a status bit which is set each time a packet has been received. The core can also be made to generate an interrupt for this event as mentioned in section 76.4.4.

RMAP CRC logic is included in the implementation if the *rmapcrc* or *rmap* VHDL generic set to 1. The RMAP CRC calculation is always active for all received packets and all bytes except the EOP/EEP are included. The packet is always assumed to be a RMAP packet and the length of the header is determined by checking byte 3 which should be the command field. The calculated CRC value is then checked when the header has been received (according to the calculated number of bytes) and if it is non-zero the HC bit is set indicating a header CRC error.

The CRC value is not set to zero after the header has been received, instead the calculation continues in the same way until the complete packet has been received. Then if the CRC value is non-zero the DC bit is set indicating a data CRC error. This means that the core can indicate a data CRC error even if the data field was correct when the header CRC was incorrect. However, the data should not be used when the header is corrupt and therefore the DC bit is unimportant in this case. When the header is not corrupted the CRC value will always be zero when the calculation continues with the data field and the behaviour will be as if the CRC calculation was restarted

If the received packet is not of RMAP type the header CRC error indication bit cannot be used. It is still possible to use the DC bit if the complete packet is covered by a CRC calculated using the RMAP CRC definition. This is because the core does not restart the calculation after the header has been received but instead calculates a complete CRC over the packet. Thus any packet format with one CRC at the end of the packet calculated according to RMAP standard can be checked using the DC bit.

If the packet is neither of RMAP type nor of the type above with RMAP CRC at the end, then both the HC and DC bits should be ignored.

76.4.9 Error handling

If a packet reception needs to be aborted because of congestion on the network, the suggested solution is to set link disable to '1'. Unfortunately, this will also cause the packet currently being transmitted to be truncated but this is the only safe solution since packet reception is a passive operation depending on the transmitter at the other end. A channel reset bit could be provided but is not a satisfactory solution since the untransmitted characters would still be in the transmitter node. The next character (somewhere in the middle of the packet) would be interpreted as the node address which would probably cause the packet to be discarded but not with 100% certainty. Usually this action is performed when a reception has stuck because of the transmitter not providing more data. The channel reset would not resolve this congestion.

If an AHB error occurs during reception the current packet is spilled up to and including the next EEP/EOP and then the currently active channel is disabled and the receiver enters the idle state. A bit in the channels control/status register is set to indicate this condition.

76.4.10 Promiscuous mode

The core supports a promiscuous mode where all the data received is stored to the DMA channel regardless of the node address and possible early EOPs/EEPs. This means that all non-eop/eep N-Chars received will be stored to the DMA channel. The rxmaxlength register is still checked and packets exceeding this size will be truncated.

RMAP commands will still be handled by the RMAP target when promiscuous mode is enabled if the rmapen bit is set. If it is cleared, RMAP commands will also be stored to the DMA channel.

76.5 Transmitter DMA engine

The transmitter DMA engine handles transmission of data from the DMA channel to the SpaceWire network. There is one DMA channel available but the core has been written so that additional DMA channels can be easily added if needed.

76.5.1 Basic functionality

The transmit DMA engine reads data from the AHB bus and stores them in the transmitter FIFO for transmission on the SpaceWire network. Transmission is based on the same type of descriptors as for the receiver and the descriptor table has the same alignment and size restrictions. When there are new descriptors enabled the core reads them and transfer the amount data indicated.

76.5.2 Setting up the core for transmission

Four steps need to be performed before transmissions can be done with the core. First the link interface must be enabled and started by writing the appropriate value to the ctrl register. Then the address to the descriptor table needs to be written to the transmitter descriptor table address register and one or more descriptors must also be enabled in the table. Finally, the txen bit in the DMA control register should be written with a one which triggers the transmission. These steps will be covered in more detail in the next sections.

76.5.3 Enabling descriptors

The descriptor table address register works in the same way as the receiver's corresponding register which was covered in section 76.4.

To transmit packets one or more descriptors have to be initialized in memory which is done in the following way: The number of bytes to be transmitted and a pointer to the data has to be set. There are two different length and address fields in the transmit descriptors because there are separate pointers for header and data. If a length field is zero the corresponding part of a packet is skipped and if both are zero no packet is sent. The maximum header length is 255 bytes and the maximum data length is

16 Mbyte - 1. When the pointer and length fields have been set the enable bit should be set to enable the descriptor. This must always be done last. The other control bits must also be set before enabling the descriptor.

The transmit descriptors are 16 bytes in size so the maximum number in a single table is 64. The different fields of the descriptor together with the memory offsets are shown in the tables below.

The HC bit should be set if RMAP CRC should be calculated and inserted for the header field and correspondingly the DC bit should be set for the data field. This field is only used by the core when the CRC logic is available (*rmap* or *rmapcrc* VHDL generic set to 1). The header CRC will be calculated from the data fetched from the header pointer and the data CRC is generated from data fetched from the data pointer. The CRCs are appended after the corresponding fields. The NON-CRC bytes field is set to the number of bytes in the beginning of the header field that should not be included in the CRC calculation. The CRCs are sent even if the corresponding length is zero.

When both header and data length are zero no packet is sent not even an EOP.

76.5.4 Starting transmissions

When the descriptors have been initialized, the transmit enable bit in the DMA control register has to be set to tell the core to start transmitting. New descriptors can be activated in the table on the fly (while transmission is active). Each time a set of descriptors is added the transmit enable register bit should be set. This has to be done because each time the core encounters a disabled descriptor this register bit is set to 0.

*Table 1241.*GRSPW transmit descriptor word 0 (address offset 0x0)

31 18	17	16	15	14	13	12	11 8	7 0
RESERVED	DC	HC	LE	IE	WR	EN	NONCRCLEN	HEADERLEN

31: 18	RESERVED
17	Append data CRC (DC) - Append CRC calculated according to the RMAP specification after the data sent from the data pointer. The CRC covers all the bytes from this pointer. A null CRC will be sent if the length of the data field is zero.
16	Append header CRC (HC) - Append CRC calculated according to the RMAP specification after the data sent from the header pointer. The CRC covers all bytes from this pointer except a number of bytes in the beginning specified by the non-crc bytes field. The CRC will not be sent if the header length field is zero.
15	Link error (LE) - A Link error occurred during the transmission of this packet.
14	Interrupt enable (IE) - If set, an interrupt will be generated when the packet has been transmitted and the transmitter interrupt enable bit in the DMA control register is set.
13	Wrap (WR) - If set, the descriptor pointer will wrap and the next descriptor read will be the first one in the table (at the base address). Otherwise the pointer is increased with 0x10 to use the descriptor at the next higher memory location.
12	Enable (EN) - Enable transmitter descriptor. When all control fields (address, length, wrap and crc) are set, this bit should be set. While the bit is set the descriptor should not be touched since this might corrupt the transmission. The GRSPW clears this bit when the transmission has finished.
11: 8	Non-CRC bytes (NONCRCLEN)- Sets the number of bytes in the beginning of the header which should not be included in the CRC calculation. This is necessary when using path addressing since one or more bytes in the beginning of the packet might be discarded before the packet reaches its destination.
7: 0	Header length (HEADERLEN) - Header Length in bytes. If set to zero, the header is skipped.

*Table 1242.*GRSPW transmit descriptor word 1 (address offset 0x4)

31 0
HEADERADDRESS

31: 0	Header address (HEADERADDRESS) - Address from where the packet header is fetched. Does not need to be word aligned.

Table 1243. GRSPW transmit descriptor word 2 (address offset 0x8)

31 24	23 0
RESERVED	DATALEN

31: 24	RESERVED
23: 0	Data length (DATALEN) - Length of data part of packet. If set to zero, no data will be sent. If both data- and header-lengths are set to zero no packet will be sent.

Table 1244. GRSPW transmit descriptor word 3(address offset 0xC)

31 0
DATAADDRESS

31: 0	Data address (DATAADDRESS) - Address from where data is read. Does not need to be word aligned.

76.5.5 The transmission process

When the txen bit is set the core starts reading descriptors immediately. The number of bytes indicated are read and transmitted. When a transmission has finished, status will be written to the first field of the descriptor and a packet sent bit is set in the DMA control register. If an interrupt was requested it will also be generated. Then a new descriptor is read and if enabled a new transmission starts, otherwise the transmit enable bit is cleared and nothing will happen until it is enabled again.

76.5.6 The descriptor table address register

The internal pointer which is used to keep the current position in the descriptor table can be read and written through the APB interface. This pointer is set to zero during reset and is incremented each time a descriptor is used. It wraps automatically when the 1 kbytes limit for the descriptor table is reached or it can be set to wrap earlier by setting a bit in the current descriptor.

The descriptor table register can be updated with a new table anytime when no transmission is active. No transmission is active if the transmit enable bit is zero and the complete table has been sent or if the table is aborted (explained below). If the table is aborted one has to wait until the transmit enable bit is zero before updating the table pointer.

76.5.7 Error handling

Abort Tx

The DMA control register contains a bit called Abort TX which if set causes the current transmission to be aborted, the packet is truncated and an EEP is inserted. This is only useful if the packet needs to be aborted because of congestion on the SpaceWire network. If the congestion is on the AHB bus this will not help (This should not be a problem since AHB slaves should have a maximum of 16 wait-states). The aborted packet will have its LE bit set in the descriptor. The transmit enable register bit is also cleared and no new transmissions will be done until the transmitter is enabled again.

AHB error

When an AHB error is encountered during transmission the currently active DMA channel is disabled and the transmitter goes to the idle mode. A bit in the DMA channel's control/status register is set to indicate this error condition and, if enabled, an interrupt will also be generated. Further error handling depends on what state the transmitter DMA engine was in when the AHB error occurred. If the descriptor was being read the packet transmission had not been started yet and no more actions need to be taken.

If the AHB error occurs during packet transmission the packet is truncated and an EEP is inserted. Lastly, if it occurs when status is written to the descriptor the packet has been successfully transmitted but the descriptor is not written and will continue to be enabled (this also means that no error bits are set in the descriptor for AHB errors).

The client using the channel has to correct the AHB error condition and enable the channel again. No more AHB transfers are done again from the same unit (receiver or transmitter) which was active during the AHB error until the error state is cleared and the unit is enabled again.

Link error

When a link error occurs during the transmission the remaining part of the packet is discarded up to and including the next EOP/EEP. When this is done status is immediately written (with the LE bit set) and the descriptor pointer is incremented. The link will be disconnected when the link error occurs but the core will automatically try to connect again provided that the link-start bit is asserted and the link-disabled bit is deasserted. If the LE bit in the DMA channel's control register is not set the transmitter DMA engine will wait for the link to enter run-state and start a new transmission immediately when possible if packets are pending. Otherwise the transmitter will be disabled when a link error occurs during the transmission of the current packet and no more packets will be transmitted until it is enabled again.

76.6 RMAP

The Remote Memory Access Protocol (RMAP) is used to implement access to resources in the node via the SpaceWire Link. Some common operations are reading and writing to memory, registers and FIFOs. The core has an optional hardware RMAP target which is enabled with a VHDL generic. This section describes the basics of the RMAP protocol and the target implementation.

76.6.1 Fundamentals of the protocol

RMAP is a protocol which is designed to provide remote access via a SpaceWire network to memory mapped resources on a SpaceWire node. It has been assigned protocol ID 0x01. It provides three operations write, read and read-modify-write. These operations are posted operations which means that a source does not wait for an acknowledge or reply. It also implies that any number of operations can be outstanding at any time and that no timeout mechanism is implemented in the protocol. Time-outs must be implemented in the user application which sends the commands. Data payloads of up to 16 Mb - 1 is supported in the protocol. A destination can be requested to send replies and to verify data before executing an operation. A complete description of the protocol is found in the RMAP standard.

76.6.2 Implementation

The core includes a taget for RMAP commands which processes all incoming packets with protocol ID = 0x01 and type field (bit 7 and 6 of the 3rd byte in the packet) equal to 01b. When such a packet is detected it is not stored to the DMA channel, instead it is passed to the RMAP receiver.

The core implements all three commands defined in the standard with some restrictions. First of all the optional error code 12 is not implemented and support is only provided for 32-bit big-endian systems. This means that the first byte received is the msb in a word. The command handler will not receive RMAP packets using the extended protocol ID which are always dumped to the DMA channel.

The RMAP receiver processes commands. If they are correct and accepted the operation is performed on the AHB bus and a reply is formatted. If an acknowledge is requested the RMAP transmitter automatically send the reply. RMAP transmissions have priority over DMA channel transmissions.

Packets with a mismatching destination logical address are never passed to the RMAP target. There is a user accessible destination key register which is compared to destination key field in incoming packets. If there is a mismatch and a reply has been requested the error code in the reply is set to 3. Replies are sent if and only if the ack field is set to '1'.

Detection of all error codes except code 12 is supported. When a failure occurs during a bus access the error code is set to 1 (General Error). There is predetermined order in which error-codes are set in the case of multiple errors in the core. It is shown in table 1245.

Table 1245. The order of error detection in case of multiple errors in the GRSPW. The error detected first has number 1.

Detection Order	Error Code	Error
1	2	Unused RMAP packet type or command code
2	3	Invalid destination key
3	9	Verify buffer overrun
4	11	RMW data length error
5	10	Authorization failure
6*	1	General Error (AHB errors during non-verified writes)
7	5/7	Early EOP / EEP (if early)
8	4	Invalid Data CRC
9	1	General Error (AHB errors during verified writes or RMW)
10	7	EEP
11	6	Cargo Too Large
*The AHB error is not guaranteed to be detected before Early EOP/EEP or Invalid Data CRC. For very long accesses the AHB error detection might be delayed causing the other two errors to appear first.		

Read accesses are performed on the fly, that is they are not stored in a temporary buffer before transmission. This means that the error code 1 will never be seen in a read reply since the header has already been sent when the data is read. If the AHB error occurs the packet will be truncated and ended with an EEP.

Errors up to and including Invalid Data CRC (number 8) are checked before verified commands. The other errors do not prevent verified operations from being performed.

The details of the support for the different commands are now presented. All defined commands which are received but have an option set which is not supported in this specific implementation will not be executed and a possible reply is sent with error code 10.

76.6.3 Write commands

The write commands are divided into two subcategories when examining their capabilities: verified writes and non-verified writes. Verified writes have a length restriction of 4 B and the address must be aligned to the size. That is 1 B writes can be done to any address, 2 B must be halfword aligned, 3 B are not allowed and 4 B writes must be word aligned. Since there will always be only one AHB operation performed for each RMAP verified write command the incrementing address bit can be set to any value.

Non-verified writes have no restrictions when the incrementing bit is set to 1. If it is set to 0 the number of bytes must be a multiple of 4 and the address word aligned. There is no guarantee how many words will be written when early EOP/EEP is detected for non-verified writes.

76.6.4 Read commands

Read commands are performed on the fly when the reply is sent. Thus if an AHB error occurs the packet will be truncated and ended with an EEP. There are no restrictions for incrementing reads but non-incrementing reads have the same alignment restrictions as non-verified writes. Note that the “Authorization failure” error code will be sent in the reply if a violation was detected even if the length field was zero. Also note that no data is sent in the reply if an error was detected i.e. if the status field is non-zero.

76.6.5 RMW commands

All read-modify-write sizes are supported except 6 which would have caused 3 B being read and written on the bus. The RMW bus accesses have the same restrictions as the verified writes. As in the verified write case, the incrementing bit can be set to any value since only one AHB bus operation will be performed for each RMW command. Cargo too large is detected after the bus accesses so this error will not prevent the operation from being performed. No data is sent in a reply if an error is detected i.e. the status field is non-zero.

76.6.6 Control

The RMAP command handler mostly runs in the background without any external intervention, but there are a few control possibilities.

There is an enable bit in the control register of the core which can be used to completely disable the RMAP command handler. When it is set to '0' no RMAP packets will be handled in hardware, instead they are all stored to the DMA channel.

There is a possibility that RMAP commands will not be performed in the order they arrive. This can happen if a read arrives before one or more writes. Since the command handler stores replies in a buffer with more than one entry several commands can be processed even if no replies are sent. Data for read replies is read when the reply is sent and thus writes coming after the read might have been performed already if there was congestion in the transmitter. To avoid this the RMAP buffer disable bit can be set to force the command handler to only use one buffer which prevents this situation.

The last control option for the command handler is the possibility to set the destination key which is found in a separate register.

Table 1246. GRSPW hardware RMAP handling of different packet type and command fields.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Command	Action
Reserved	Command / Response	Write / Read	Verify data before write	Acknow-ledge	Increment Address		
0	0	-	-	-	-	Response	Stored to DMA-channel.
0	1	0	0	0	0	Not used	Does nothing. No reply is sent.
0	1	0	0	0	1	Not used	Does nothing. No reply is sent.
0	1	0	0	1	0	Read single address	Executed normally. Address has to be word aligned and data size a multiple of four. Reply is sent. If alignment restrictions are violated error code is set to 10.
0	1	0	0	1	1	Read incre-menting address.	Executed normally. No restric-tions. Reply is sent.
0	1	0	1	0	0	Not used	Does nothing. No reply is sent.
0	1	0	1	0	1	Not used	Does nothing. No reply is sent.
0	1	0	1	1	0	Not used	Does nothing. Reply is sent with error code 2.
0	1	0	1	1	1	Read-Mod-ify-Write increment-ing address	Executed normally. If length is not one of the allowed rmw val-ues nothing is done and error code is set to 11. If the length was correct, alignment restric-tions are checked next. 1 byte can be rmw to any address. 2 bytes must be halfword aligned. 3 bytes are not allowed. 4 bytes must be word aligned. If these restrictions are violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	0	0	0	Write, sin-gle-address, do not verify before writ-ing, no acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done. No reply is sent.
0	1	1	0	0	1	Write, incre-menting address, do not verify before writ-ing, no acknowledge	Executed normally. No restric-tions. No reply is sent.
0	1	1	0	1	0	Write, sin-gle-address, do not verify before writ-ing, send acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.

*Table 1246.*GRSPW hardware RMAP handling of different packet type and command fields.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Command	Action
Reserved	Command / Response	Write / Read	Verify data before write	Acknow-ledge	Increment Address		
0	1	1	0	1	1	Write, incre-menting address, do not verify before writ-ing, send acknowledge	Executed normally. No restric-tions. If AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	0	0	Write, single address, ver-ify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restric-tions apply as for rmw. No reply is sent.
0	1	1	1	0	1	Write, incre-menting address, ver-ify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restric-tions apply as for rmw. If they are violated nothing is done. No reply is sent.
0	1	1	1	1	0	Write, single address, ver-ify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are vio-lated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	1	1	Write, incre-menting address, ver-ify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are vio-lated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
1	0	-	-	-	-	Unused	Stored to DMA-channel.
1	1	-	-	-	-	Unused	Stored to DMA-channel.

76.7 AMBA interface

The AMBA interface consists of an APB interface, an AHB master interface and DMA FIFOs. The APB interface provides access to the user registers which are described in section 76.9. The DMA engines have 32-bit wide FIFOs to the AHB master interface which are used when reading and writing to the bus.

The transmitter DMA engine reads data from the bus in bursts which are half the FIFO size in length. A burst is always started when the FIFO is half-empty or if it can hold the last data for the packet. The burst containing the last data might have shorter length if the packet is not an even number of bursts in size.

The receiver DMA works in the same way except that it checks if the FIFO is half-full and then performs a burst write to the bus which is half the fifo size in length. The last burst might be shorter. If the *rmap* or *rxunaligned* VHDL generics are set to 1 the interface also handles byte accesses. Byte accesses are used for non word-aligned buffers and/or packet lengths that are not a multiple of four bytes. There might be 1 to 3 single byte writes when writing the beginning and end of the received packets.

76.7.1 APB slave interface

As mentioned above, the APB interface provides access to the user registers which are 32-bits in width. The accesses to this interface are required to be aligned word accesses. The result is undefined if this restriction is violated.

76.7.2 AHB master interface

The core contains a single master interface which is used by both the transmitter and receiver DMA engines. The arbitration algorithm between the channels is done so that if the current owner requests the interface again it will always acquire it. This will not lead to starvation problems since the DMA engines always deassert their requests between accesses.

The AHB accesses are always word accesses (HSIZE = 0x010) of type incremental burst with unspecified length (HBURST = 0x001) if VHDL generics rmap and rxunaligned are disabled. The AHB accesses can be of size byte, halfword and word (HSIZE = 0x000, 0x001, 0x010) otherwise. Byte and halfword accesses are always NONSEQ. Note that read accesses are always word accesses (HSIZE = 0x010), which can result in destructive read.

The burst length will be half the AHB FIFO size except for the last transfer for a packet which might be smaller. Shorter accesses are also done during descriptor reads and status writes.

The AHB master also supports non-incrementing accesses where the address will be constant for several consecutive accesses. HTRANS will always be NONSEQ in this case while for incrementing accesses it is set to SEQ after the first access. This feature is included to support non-incrementing reads and writes for RMAP.

If the core does not need the bus after a burst has finished there will be one wasted cycle (HTRANS = IDLE).

BUSY transfer types are never requested and the core provides full support for ERROR, RETRY and SPLIT responses.

76.8 Implementation

76.8.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

76.8.2 Clock-generation

Figure 192 shows the clock recovery scheme for the receiver. Data and strobe are coupled directly from their pads to an xor gate which generates the clock. The output from the xor is then connected to a clock network. The specific type of clock network depends on the technology used. The xor gate is actually all that logically belongs to the Rx clock recovery module in figure 192.

The clock output drives all flip-flops in the receiver module found in figure 188. The data signal which is used for generating the clock is also coupled to the data inputs of several flip-flops clocked

by the Rx clock as seen in figure 192. Care must be taken so that the delay from the data and strobe signals through the clock network are longer than the delay to the data input + setup time.

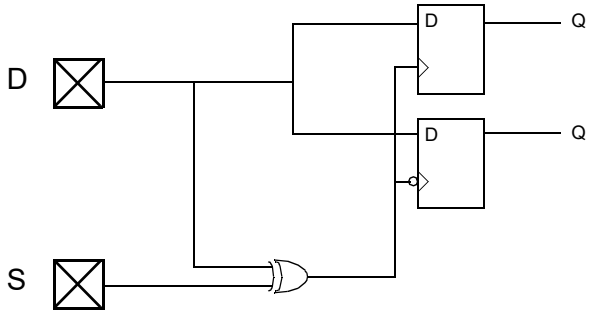


Figure 192. The clocking scheme for the receiver. The clock is

The transmitter clock is generated from the txclk input. A separate clock input is used to allow the transmitter to be run at much higher frequencies than the system clock. The SpaceWire node contains a clock-divider which divides the txclk signal to the wanted frequency. The transmitter clock should be 10 MHz during initialization and any frequency above 2 MHz in the run-state.

There is an input signal called clkdiv10 which sets the clock divisor value during initialization and the reset value for the user accessible clock divisor register. The user register value will be used in run-state. The resulting tx clock frequency will be txclk/(clock divisor value+1). So if no clock division is wanted, the clock divisor should be set to 0.

Since only integer values are allowed for the clock division and the required init-frequency is 10 Mhz the frequency of the txclk input must be a multiple of 10 MHz. The clock divisor value is 8-bits wide so the maximum txclk frequency supported is 2.56 GHz (note that there is also a restriction on the relation between the system and transmit clock frequencies).

76.8.3 Timers

There are two timers in the core: one for generating the 6.4/12.8 us periods and one for disconnect timing. They run on the system (AMBA) clock and the frequency must be at least 10 MHz to guarantee disconnect timing limits.

There are two user accessible registers which are used to the set the number of clock cycles used for the timeout periods. These registers are described in section 76.9.

The reset value for the timer registers can be set in two different ways selected by the usegen VHDL generic. If usegen is set to 1, the sysfreq VHDL generic is used to generate reset values for the disconnect, 6.4 us and 12.8 us timers. Otherwise, the input signals dcrstval and timerrstval will be used as reset values. If the system clock frequency is 10 MHz or above the disconnect time will be within the limits specified in the SpaceWire standard.

76.8.4 Synchronization

The VHDL generic nsync selects how many synchronization registers are used between clock domains. The default is one and should be used when maximum performance is needed. It allows the transmitter to be clocked 4 times faster than the system clock and the receiver 2 times faster. These are theoretical values without consideration for clock skew and jitter. Note also that the receiver clocks data at both negative and positive edges. Thus, the bitrate is twice as high as the clock-rate.

The synchronization limits the Tx and Rx clocks to be at most 4 and 2 times faster than the system clock. But it might not be possible to achieve such high clock rates for the Tx and Rx clocks for all technologies.

The asynchronous reset to the receiver clock domain has to have a maximum delay of one receiver clock cycle to ensure correct operation. This is needed because the receiver uses has a completely

asynchronous reset. To make sure that nothing bad happens the is a synchronous reset guard which prevents any signals from being assigned before all registers have their reset signals released.

76.8.5 Fault-tolerance

The core can optionally be implemented with fault-tolerance against SEU errors in the FIFO memories. The fault-tolerance is enabled through the *ft* VHDL generic. Possible options are byte parity protection (*ft = 1*) or TMR registers (*ft = 2*). Note: the GPL version of GRLIB does not include fault-tolerance, and the core will not work unless the *ft* VHDL generic is 0.

76.8.6 Synthesis

Since the receiver and transmitter may run on very high frequency clocks their clock signals have been coupled through a clock buffer with a technology wrapper. This clock buffer will utilize a low skew net available in the selected technology for the clock.

The clock buffer will also enable most synthesis tools to recognize the clocks and it is thus easier to find them and place constraints on them. The fact there are three clock domains in the GRSPW of which all are possibly high frequency clocks makes it necessary to declare all paths between the clock domains as false paths.

In Synplify this is most easily done by declaring all the clocks to be in different clockgroups in the sdc file (if Synplify does not automatically put them in different groups). This will disable any timing considerations between the clock domains and these constraints will also propagate to the place and route tool.

The type of clock buffer is selectable with a VHDL generic and the value zero provides a normal feed through which lets the synthesis tool infer the type of net used.

76.8.7 Technology mapping

The core has three generics for technology mapping: *tech*, *techfifo* and *memtech*. *Tech* selects the technology used for the clock buffers and also adds reset to some registers for technologies where they would otherwise cause problems with gate-level simulations. *Techfifo* selects whether *memtech* should be used to select the technology for the FIFO memories (the RMAP buffer is not affected by the this generic) or if they should be inferred. *Tech* and *memtech* can be set to any value from 0 to NTECH as defined in the GRLIB.TECH package.

76.8.8 RAM usage

The core maps all RAM memories on the syncram_2p component if the *ft* generic is 0 and to the syncram_2pft component for other values. The syncrams are located in the technology mapping library (TECHMAP). The organization of the different memories are described below. If techfifo and/or memtech is set to 0 the synthesis tool will infer the memories. Either RAM blocks or flip-flops will be used depending on the tool and technology. The number of flip-flops used is *syncram depth x syncram width* for all the different memories. The receiver AHB FIFO with fifosize 32 will for example use 1024 flips-flops.

Receiver ahb FIFO

The receiver AHB fifo consists of one syncram_2p block with a width of 32-bits. The depth is determined by the configured FIFO depth. Table 1247 shows the syncram organization for the allowed configurations.

*Table 1247.*syncram_2p sizes for GRSPW receiver AHB FIFO.

Fifosize	Syncram_2p organization
4	4x32
8	8x32
16	16x32
32	32x32

Transmitter ahb FIFO

The transmitter AHB fifo consists of one syncram_2p block with a width of 32-bits. The depth is determined by the configured FIFO depth. Table 1248 shows the syncram organization for the allowed configurations.

*Table 1248.*syncram_2p sizes for transmitter AHB FIFO.

Fifosize	Syncram_2p organization
4	4x32
8	8x32
16	16x32
32	32x32

Receiver N-Char FIFO

The receiver N-Char fifo consists of one syncram_2p block with a width of 10-bits. The depth is determined by the configured FIFO depth. Table 1249 shows the syncram organization for the allowed configurations.

*Table 1249.*syncram_2p sizes for the receiver N-Char FIFO.

Fifosize	Syncram_2p organization
16	16x10
32	32x10
64	64x10

RMAP buffer

The RMAP buffer consists of one syncram_2p block with a width of 8-bits. The depth is determined by the number of configured RMAP buffers. Table 1250 shows the syncram organization for the allowed configurations.

*Table 1250.*syncram_2p sizes for RMAP buffer memory.

RMAP buffers	Syncram_2p organization
2	64x8
4	128x8
8	256x8

76.9 AXI support

The core is designed for an AMBA system but can be adapted for AXI using the AHBM2AXI adapter.

76.10 Registers

The core is programmed through registers mapped into APB address space.

*Table 1251.*GRSPW registers

APB address offset	Register
0x0	Control
0x4	Status/Interrupt-source
0x8	Node address
0xC	Clock divisor
0x10	Destination key
0x14	Time
0x18	Timer and Disconnect
0x20	DMA channel 1 control/status
0x24	DMA channel 1 rx maximum length
0x28	DMA channel 1 transmit descriptor table address.
0x2C	DMA channel 1 receive descriptor table address.

76.10.1 Control Register

*Table 1252.*0x00 - CTRL - control register

31	30	29	28 27	26	25 22	21	20	19 18	17	16	15 12	11	10	9	8	7	6	5	4	3	2	1	0
RA	RX	RC	R	PO	RESERVED	PS	NP	R	RD	RE	RESERVED	TR	TT	LI	TQ	R	RS	PM	TI	IE	AS	LS	LD
*	*	*	0	*	0	0	*	0	0	*	0	0	0	NR	NR	0	0	0	0	0	*	0	0
r	r	r	r	r	r	rw*	rw*	r	rw*	rw*	r	rw	rw	rw	rw	rw	rw*	rw	rw	rw	rw	rw	rw

31 RMAP available (RA) - Set to one if the RMAP command handler is available. Only readable.

30 RX unaligned access (RX) - Set to one if unaligned writes are available for the receiver. Only readable.

29 RMAP CRC available (RC) - Set to one if RMAP CRC is enabled in the core. Only readable.

28: 27 RESERVED

26 Number of ports (PO) - The number of available SpaceWire ports minus one. Only readable.

25: 22 RESERVED

21 Port select (PS) - Selects the active port when the no port force bit is zero. ‘0’ selects the port connected to data and strobe on index 0 while ‘1’ selects index 1. Only available if the ports VHDL generic is set to 2. Reset value: ‘0’.

20 No port force (NP) - Disable port force. When disabled the port select bit cannot be used to select the active port. Instead, it is automatically selected by checking the activity on the respective receive links. Only available if the ports VHDL generic is set to 2. Reset value: ‘0’ if the RMAP command handler is not available. If available the reset value is set to the value of the rmapen input signal.

19: 18 RESERVED

17 RMAP buffer disable (RD) - If set only one RMAP buffer is used. This ensures that all RMAP commands will be executed consecutively. Only available if the rmap VHDL generic is set to 1. Reset value: ‘0’.

16 RMAP Enable (RE) - Enable RMAP command handler. Only available if rmap VHDL generic is set to 1. Reset value: ‘1’.

15: 12 RESERVED

11 Time Rx Enable (TR) - Enable time-code receptions. Reset value: ‘0’.

10 Time Tx Enable (TT) - Enable time-code transmissions. Reset value: ‘0’.

9 Link error IRQ (LI) - Generate interrupt when a link error occurs. Not reset.

8 Tick-out IRQ (TQ) - Generate interrupt when a valid time-code is received. Not reset.

7 RESERVED

6 Reset (RS) - Make complete reset of the SpaceWire node. Self clearing. Reset value: ‘0’.

5 Promiscuous Mode (PM) - Enable Promiscuous mode. Reset value: ‘0’.

4 Tick In (TI) - The host can generate a tick by writing a one to this field. This will increment the timer counter and the new value is transmitted after the current character is transferred. A tick can also be generated by asserting the tick_in signal. Reset value: ‘0’.

3 Interrupt Enable (IE) - If set, an interrupt is generated when one or both of bit 8 to 9 is set and its corresponding event occurs. Reset value: ‘0’.

2 Autostart (AS) - Automatically start the link when a NULL has been received. Reset value: ‘0’ if the RMAP command handler is not available. If available the reset value is set to the value of the rmapen input signal.

1 Link Start (LS) - Start the link, i.e. allow a transition from ready to started state. Reset value: ‘0’.

0 Link Disable (LD) - Disable the SpaceWire codec. Reset value: ‘0’.

76.10.2 Status Register

*Table 1253.*0x04 - STS - status register

31 24	23 21	20 10	9	8	7	6	5	4	3	2	1	0
RESERVED	LS	RESERVED	AP	EE	IA	WE	R	PE	DE	ER	CE	TO
0	0	0	0	0	0	0	0	0	0	0	0	0
r	r	r	rw*	wc	wc	wc	r	wc	wc	wc	wc	wc

31: 24 RESERVED

23: 21 Link State (LS) - The current state of the start-up sequence. 0 = Error-reset, 1 = Error-wait, 2 = Ready, 3 = Started, 4 = Connecting, 5 = Run. Reset value: 0.

20: 10 RESERVED

9 Active port (AP) - Shows the currently active port. ‘0’ = Port 0 and ‘1’ = Port 1 where the port numbers refer to the index number of the data and strobe signals. Only available if the ports generic is set to 2.

8 Early EOP/EEP (EE) - Set to one when a packet is received with an EOP after the first byte for a non-rmap packet and after the second byte for a RMAP packet. Cleared when written with a one. Reset value: ‘0’.

7 Invalid Address (IA) - Set to one when a packet is received with an invalid destination address field, i.e it does not match the nodeaddr register. Cleared when written with a one. Reset value: ‘0’.

6 Write synchronization Error (WE) - A synchronization problem has occurred when receiving N-Chars. Cleared when written with a one. Reset value: ‘0’.

5 RESERVED

4 Parity Error (PE) - A parity error has occurred. Cleared when written with a one. Reset value: ‘0’.

3 Disconnect Error (DE) - A disconnection error has occurred. Cleared when written with a one. Reset value: ‘0’.

2 Escape Error (ER) - An escape error has occurred. Cleared when written with a one. Reset value: ‘0’.

1 Credit Error (CE) - A credit has occurred. Cleared when written with a one. Reset value: ‘0’.

0 Tick Out (TO) - A new time count value was received and is stored in the time counter field. Cleared when written with a one. Reset value: ‘0’.

76.10.3 Node Address Register

*Table 1254.*0x08 - NODEADDR - node address register

31 8	7 0
RESERVED	NODEADDR
0	*
r	rw

31: 8 RESERVED

7: 0 Node address (NODEADDR) - 8-bit node address used for node identification on the SpaceWire network. Reset value: 254 (taken from the nodeaddr VHDL generic when /= 255, else from the rmapnodeaddr input signal)

76.10.4 Clock Divisor Register

*Table 1255.*0x0C - CLKDIV - clock divisor register

31 16	15 8	7 0
RESERVED	CLKDIVSTART	CLKDIVRUN
0	*	*
r	rw	rw

31: 16 RESERVED

15: 8 Clock divisor startup (CLKDIVSTART) - 8-bit Clock divisor value used for the clock-divider during startup (link-interface is in other states than run). The actual divisor value is Clock Divisor register + 1. Reset value: clkdiv10 input signal.

7: 0 Clock divisor run (CLKDIVRUN) - 8-bit Clock divisor value used for the clock-divider when the link-interface is in the run-state. The actual divisor value is Clock Divisor register + 1. Reset value: clkdiv10 input signal.

76.10.5 Destination Key

*Table 1256.*0x10 - DKEY - destination key

31 8	7 0
RESERVED	DESTKEY
0	*
r	rw

31: 8 RESERVED

7: 0 Destination key (DESTKEY) - RMAP destination key. Only available if the rmap VHDL generic is set to 1. Reset value: 0 (taken from the deskey VHDL generic)

76.10.6 Time Register

*Table 1257.*0x14 - TIME - time register

31 8	7 6	5 0
RESERVED	TCTRL	TIMECNT
0	0	0
r	rw	rw

31: 8 RESERVED

7: 6 Time control flags (TCTRL) - The current value of the time control flags. Sent with time-code resulting from a tick-in. Received control flags are also stored in this register. Reset value: ‘0’.

5: 0 Time counter (TIMECNT) - The current value of the system time counter. It is incremented for each tick-in and the incremented value is transmitted. The register can also be written directly but the written value will not be transmitted. Received time-counter values are also stored in this register. Reset value: ‘0’.

76.10.7 Timer and Disconnect Register

*Table 1258.*0x18 - TDR - timer and disconnect register.

31 22	21 12	11 0
RESERVED	DISCONNECT	TIMER64

31: 22 RESERVED

21: 12 Disconnect (DISCONNECT) - Used to generate the 850 ns disconnect time period. The disconnect period is the number is the number of clock cycles in the disconnect register + 3. So to get a 850 ns period, the smallest number of clock cycles that is greater than or equal to 850 ns should be calculated and this values - 3 should be stored in the register. Reset value is set with VHDL generics or with input signals depending on the value of the usegen VHDL generic.

11: 0 6.4 us timer (TIMER64) - Used to generate the 6.4 and 12.8 us time periods. Should be set to the smallest number of clock cycles that is greater than or equal to 6.4 us. Reset value is set with VHDL generics or with input signals depending on the value of the usegen VHDL generic.

76.10.8 DMA Control Register

*Table 1259.*0x20 - DMACTRL - dma control register

31 17	16	15 13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED	LE	RESERVED	NS	RD	RX	AT	RA	TA	PR	PS	AI	RI	TI	RE	TE
0	0	0	0	0	0	0	0	0	0	0	NR	NR	NR	0	0
r	rw	r	rw	rw	r	r	wc	wc	wc	wc	rw	rw	rw	rw	rw

31: 17 RESERVED

16 Link error disable (LE) - Disable transmitter when a link error occurs. No more packets will be transmitted until the transmitter is enabled again. Reset value: ‘0’.

15: 13 RESERVED

12 No spill (NS) - If cleared, packets will be discarded when a packet is arriving and there are no active descriptors. If set, the GRSPW will wait for a descriptor to be activated.

11 Rx descriptors available (RD) - Set to one, to indicate to the GRSPW that there are enabled descriptors in the descriptor table. Cleared by the GRSPW when it encounters a disabled descriptor: Reset value: ‘0’.

10 RX active (RX) - Is set to ‘1’ if a reception to the DMA channel is currently active otherwise it is ‘0’. Only readable.

9 Abort TX (AT) - Set to one to abort the currently transmitting packet and disable transmissions. If no transmission is active the only effect is to disable transmissions. Self clearing. Reset value: ‘0’.

8 RX AHB error (RA) - An error response was detected on the AHB bus while this receive DMA channel was accessing the bus. Cleared when written with a one. Reset value: ‘0’.

7 TX AHB error (TA) - An error response was detected on the AHB bus while this transmit DMA channel was accessing the bus. Cleared when written with a one. Reset value: ‘0’.

6 Packet received (PR) - This bit is set each time a packet has been received. never cleared by the SW-node. Cleared when written with a one. Reset value: ‘0’.

5 Packet sent (PS) - This bit is set each time a packet has been sent. Never cleared by the SW-node. Cleared when written with a one. Reset value: ‘0’.

4 AHB error interrupt (AI) - If set, an interrupt will be generated each time an AHB error occurs when this DMA channel is accessing the bus. Not reset.

3 Receive interrupt (RI) - If set, an interrupt will be generated each time a packet has been received. This happens both if the packet is terminated by an EEP or EOP. Not reset.

2 Transmit interrupt (TI) - If set, an interrupt will be generated each time a packet is transmitted. The interrupt is generated regardless of whether the transmission was successful or not. Not reset.

*Table 1259.*0x20 - DMACTRL - dma control register

1 Receiver enable (RE) - Set to one when packets are allowed to be received to this channel. Reset value: ‘0’.

0 Transmitter enable (TE) - Write a one to this bit each time new descriptors are activated in the table. Writing a one will cause the SW-node to read a new descriptor and try to transmit the packet it points to. This bit is automatically cleared when the SW-node encounters a descriptor which is disabled. Reset value: ‘0’.

76.10.9 RX Maximum Length Register

*Table 1260.*0x24 - DMAMAXLEN - RX maximum length register.

31 25	24 2	1 0
RESERVED	RXMAXLEN	R
0	NR	0
r	rw	r

31: 25 RESERVED

24: 2 RX maximum length (RXMAXLEN) - Receiver packet maximum length in bytes. Only bits 24 - 2 are writable. Bits 1 - 0 are always 0. Not reset.

1: 0 RESERVED

76.10.10Transmitter Descriptor Table Address Register

*Table 1261.*0x28 - DMATYDESC - transmitter descriptor table address register.

31 10	9 4	3 0
DESCBASEADDR	DESCSEL	RESERVED
NR	0	0
rw	rw	r

31: 10 Descriptor table base address (DESCBASEADDR) - Sets the base address of the descriptor table. Not reset.

9: 4 Descriptor selector (DESCSEL) - Offset into the descriptor table. Shows which descriptor is currently used by the GRSPW. For each new descriptor read, the selector will increase with 16 and eventually wrap to zero again. Reset value: 0.

3: 0 RESERVED

76.10.11Receiver Descriptor Table Address Register

*Table 1262.*0x2C - DMARXDESC - receiver descriptor table address register.

31 10	9 3	2 0
DESCBASEADDR	DESCSEL	RESERVED
NR	0	0
rw	rw	r

31: 10 Descriptor table base address (DESCBASEADDR) - Sets the base address of the descriptor table. Not reset.

9: 3 Descriptor selector (DESCSEL) - Offset into the descriptor table. Shows which descriptor is currently used by the GRSPW. For each new descriptor read, the selector will increase with 8 and eventually wrap to zero again. Reset value: 0.

2: 0 RESERVED

76.11 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x1F. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

76.12 Implementation

76.12.1 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

76.13 Configuration options

Table 1263 shows the configuration options of the core (VHDL generics).

Table 1263. Configuration options

Generic	Function	Allowed range	Default
tech	Technology for clock buffers	0 - NTECH	inferred
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by GRSPW.	0 - NAHBIRQ-1	0
sysfreq	Frequency of clock input "clk" in kHz.	-	10000
usegen	Use values calculated from sysfreq generic as reset values for 6.4 us timer and disconnect timer.	0 - 1	1
nsync	Number of synchronization registers. Warning: Value 2 only to be used when bit rate is equal or less than the system clock frequency.	1 - 2	1
rmap	Include hardware RMAP target. RMAP CRC logic will also be added. If set to 2 the core will only implement the RMAP target, provide a limited APB interface, enable time code reception and its interrupt.	0 - 2	0
rmapcrc	Enable RMAP CRC logic.	0 - 1	0
fifosize1	Sets the number of entries in the 32-bit receiver and transmitter AHB fifos.	4 - 32	32
fifosize2	Sets the number of entries in the 9-bit receiver fifo (N-Char fifo).	16 - 64	64
rxclkbuftype	Select clock buffer type for receiver clock. 0 does not select a buffer, instead i connects the input directly to the output (synthesis tools may still infer a buffer). 1 selects hardwired clock while 2 selects routed clock.	0 - 2	0
rxunaligned	Receiver unaligned write support. If set, the receiver can write any number of bytes to any start address without writing any excessive bytes.	0 - 1	0
rmapbufs	Sets the number of buffers to hold RMAP replies.	2 - 8	4
ft	Enable fault-tolerance against SEU errors	0 - 2	0
scantest	Enable support for scan test	0 - 1	0
techfifo	Implement FIFO with RAM cells (1) or flip-flops (0)	0 - 1	1
netlist	Use netlist rather then RTL code	0 - 1	0
ports	Sets the number of ports	1 - 2	1
memtech	Technology for RAM blocks	0 - NTECH	inferred
nodeaddr	Sets the reset value for the core's node address. Value 255 enables rmapnodeaddr input instead.	0 - 254 255	254
destkey	Sets the reset value for the core's destination key.	0 - 255	0

76.14 Signal descriptions

Table 1264 shows the interface signals of the core (VHDL ports). As indicated in the table the core consists of two different entities, called GRSPW and GRSPW_PHY. The GRSPW entity is the main part and includes most core's functionality, while the GRSPW_PHY only handles the receiver clock generation and the lower parts of the PHY layer. One GRSPW_PHY entity is used per port. See section 76.18 for information on how to interface GRSPW with GRSPW_PHY.

Table 1264. Signal descriptions

Entity	Signal name	Field	Type	Function	Active
GRSPW	RST	N/A	Input	Reset	Low
	CLK	N/A	Input	Clock	-
	RXCLK[1:0]	N/A	Input	Receiver clock. One clock per port.	-
	TXCLK	N/A	Input	Transmitter default run-state clock	-
	AHBMI	*	Input	AHB master input signals	-
	AHBMO	*	Output	AHB master output signals	-
	APBI	*	Input	APB slave input signals	-
	APBO	*	Output	APB slave output signals	-
	SWNI	D[1:0]	Input	Data input synchronous to RXCLK (rising edge). One bit per port.	-
		ND[9:0]	Input	Data input synchronous to RXCLK (falling edge). Five bits per port.	
		TICKIN	Input	Time counter tick input	High
		CLKDIV10	Input	Clock divisor value used during initialization and as reset value for the clock divisor register	-
		RMAPEN	Input	Reset value for the rmapen control register bit	-
		RMAPNODEADDR	Input	Reset value for nodeaddr register bits when nodeaddr VHDL generic /= 255	-
		DCRSTVAL	Input	Reset value for disconnect timer. Used if usegen VHDL generic is set to 0.	-
		TIMERRSTVAL	Input	Reset value for 6.4 us timer. Used if usegen VHDL generic is set to 0.	-
		DCONNECT[3:0]	Input	Disconnect strobes. Two bits per port.	
	SWNO	D[1:0]	Output	SpaceWire data output. One bit per port.	-
		S[1:0]	Output	SpaceWire strobe output. One bit per port.	-
		TICKOUT	Output	Time counter tick output	High
		LINKDIS	Output	Linkdisabled status	High
		RMAPACT	Output	RMAP command processing active	High
		RXRST	Output	Receiver reset.	Low

Table 1264. Signal descriptions

Entity	Signal name	Field	Type	Function	Active
GRSPW_PHY	RXRST	N/A	Input	Receiver reset.	Low
	DI	N/A	Input	SpaceWire data input.	-
	SI	N/A	Input	SpaceWire strobe input.	-
	RXCLKO	N/A	Output	Receiver clock recovered from data and strobe input.	-
	DO	N/A	Ouput	Recovered data, synchronous to RXCLKO (rising edge).	-
	NDO[4:0]	N/A	Ouput	Recovered data, synchronous to RXCLKO (falling edge)	-
	DCON-NECT[1:0]	N/A	Ouput	Disconnect strobe signals.	-
	TESTEN	N/A	Input	Scan test enable	High
	TESTCLK	N/A	Input	Scan test clock. Used inside the GRSP-W_PHY entity instead of recovered RXCLK when TESTEN is active.	-
	* see GRLIB IP Library User's Manual				

76.15 Signal definitions and reset values

The signals and their reset values are described in table 1265.

*Table 1265.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
spw_clk	Input	Transmitter default run-state clock	Rising edge	-
spw_rxd	Input, LVDS	Data input, positive	High	-
spw_rxdn	Input, LVDS	Data input, negative	Low	-
spw_rxs	Input, LVDS	Strobe input, positive	High	-
spw_rxsn	Input, LVDS	Strobe input, negative	Low	-
spw_txd	Output, LVDS	Data output, positive	High	Logical 0
spw_txdn	Output, LVDS	Data output, negative	Low	Logical 1
spw_txs	Output, LVDS	Strobe output, positive	High	Logical 0
spw_txsn	Output, LVDS	Strobe output, negative	Low	Logical 1

76.16 Timing

The timing waveforms and timing parameters are shown in figure 193 and are defined in table 1266.

The SpaceWire jitter and skew timing waveforms and timing parameters are shown in figure 194 and are defined in table 1267.

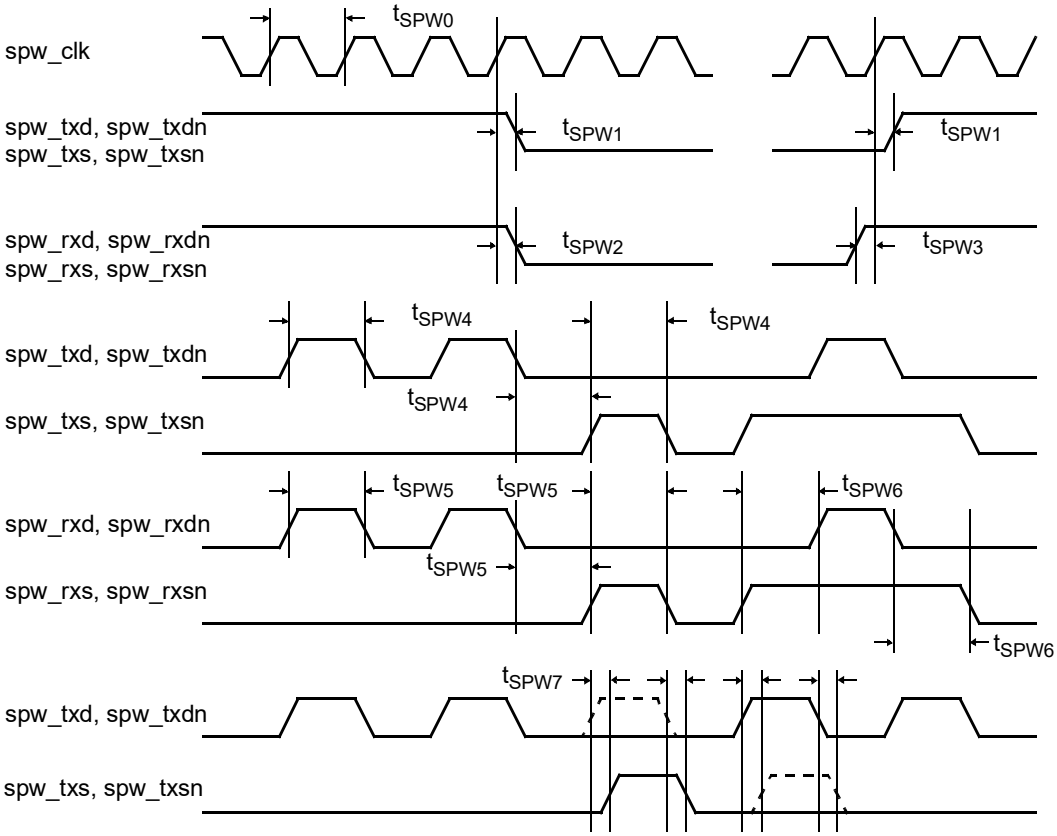


Figure 193. Timing waveforms

Table 1266. Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{SPW0}	transmit clock period	-	TBD	-	ns
t_{SPW1}	clock to output delay	rising *spw_clk* edge	TBD	TBD	ns
t_{SPW2}	input to clock hold	-	-	-	not applicable
t_{SPW3}	input to clock setup	-	-	-	not applicable
t_{SPW4}	output data bit period	-	-		*clk* periods
		-	t_{SPW0} - TBD	t_{SPW0} +TBD	ns
t_{SPW5}	input data bit period	-	TBD	-	ns
t_{SPW6}	data & strobe edge separation	-	TBD	-	ns
t_{SPW7}	data & strobe output skew	-	-	TBD	ns

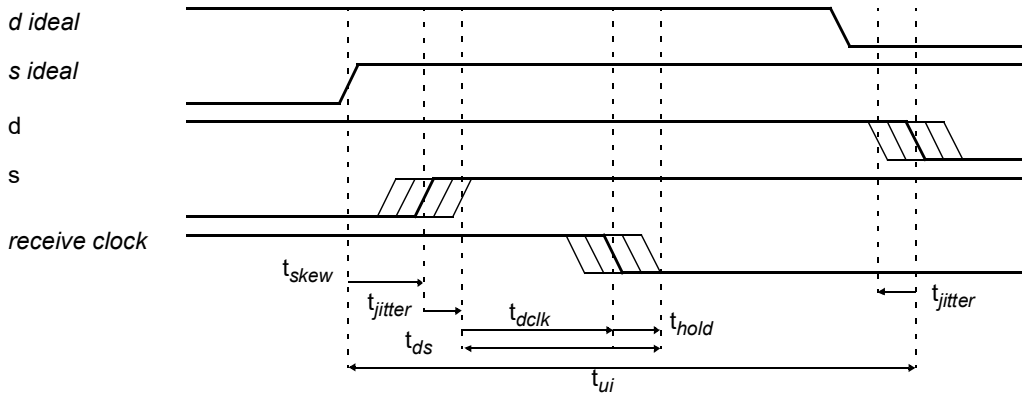


Figure 194. Skew and jitter timing waveforms

*Table 1267.*Skew and jitter timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{skew}	skew between data and strobe	-	-	TBD	ns
t_{jitter}	jitter on data or strobe	-	-	TBD	ns
t_{ds}	minimum separation between data and strobe edges	-	TBD	-	ns
t_{dclk}	delay from edge of data or strobe to the receiver flip-flop	-	-	TBD	ns
t_{hold}	hold timer on receiver flip-flop	-	TBD	-	ns
t_{ui}	unit interval (bit period)	-	TBD	-	ns

76.17 Library dependencies

Table 1268 shows libraries used when instantiating the core (VHDL libraries).

*Table 1268.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	SPACEWIRE	Signals, component	Component and record declarations.

76.18 Instantiation

This example shows how the core can be instantiated.

Normally di, si, do and so should be connected to input and output pads configured with LVDS drivers. How this is done is technology dependent.

The GRSPW in the example is a 2-port core configured with non-ft memories of size 4, 64 and 8 entries for AHB FIFOs, N-Char FIFO and RMAP buffers respectively. The system frequency (clk) is 40 MHz and the transmitter frequency (txclk) is 20 MHz.

The memory technology is inferred which means that the synthesis tool will select the appropriate components. The rx clk buffer uses a hardwired clock.

The hardware RMAP command handler is enabled which also automatically enables rxunaligned and rmapcrc. Finally, the DMA channel interrupt line is 2 and the number of synchronization registers is 1.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.spacewire.all;

entity spacewire_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- spacewire signals
    di :  in  std_logic_vector(1 downto 0);
    si :  in  std_logic_vector(1 downto 0);
    do :  out std_logic_vector(1 downto 0);
    so :  out std_logic_vector(1 downto 0)
    );
end;

architecture rtl of spacewire_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- Spacewire signals
  signal swni : grspw_in_type;
  signal swno : grspw_out_type;
  signal rxclk : std_logic_vector(1 downto 0);

begin

  -- AMBA Components are instantiated here
  ...

  -- GRSPW
  sw0 : grspw
  generic map (tech => inferred, hindex => 5, pindex => 7, paddr => 7, nsync => 1,
    rmap => 1, rxunaligned => 0, rmapcrc => 0, rxclkbuftype => 0, sysfreq => 40000,
    pirq => 2, fifosize1 => 4, fifosize2 => 64, rmapbufs => 8, ft => 0, ports => 2)
  port map (rstn, clk, rxclk, apbi, apbo(7), ahbmi, ahbmo(5), swni, swno);

  phy0 : grspw_phy
  generic map (tech => inferred, rxclkbuftype => 0, scantest => 0)
  port map (rxrst => swno.rxrst, di => di(0), si => si(0),
    rxclko => rxclk(0), do => swni.d(0), ndo => swni.nd(4 downto 0),
    dconnect => swni.dconnect(1 downto 0));

  phy1 : grspw_phy
  generic map (tech => inferred, rxclkbuftype => 0)
  port map (rxrst => swno.rxrst, di => di(1), si => si(1),
    rxclko => rxclk(1), do => swni.d(1), ndo => swni.nd(9 downto 5),
    dconnect => swni.dconnect(3 downto 2));

  swni.rmapen   <= '1';
  swni.clkdiv10 <= "00000001";
  swni.tickin   <= '0';
  do(0)         <= swno.d(0);
  so(0)         <= swno.s(0);
  do(1)         <= swno.d(1);
  so(1)         <= swno.s(1);
end;
```

76.19 API

A simple Application Programming Interface (API) is provided together with the GRSPW. The API is located in $(GRLIB)/software/spw. The files are rmapapi.c, spwapi.c, rmapapi.h, spwapi.h. The spwapi.h file contains the declarations of the functions used for configuring the GRSPW and transferring data. The corresponding definitions are located in spwapi.c. The rmapapi is structured in the same manner and contains a function for building RMAP packets.

These functions could be used as a simple starting point for developing drivers for the GRSPW. The different functions are described in this section.

76.19.1 GRSPW Basic API

The basic GRSPW API is based on a struct spwvars which stores all the information for a single GRSPW core. The information includes its address on the AMBA bus as well as SpaceWire parameters such as node address and clock divisor. A pointer to this struct is used as a input parameter to all the functions. If several cores are used, a separate struct for each core is created and used when the specific core is accessed.

Table 1269. The spwvars struct

Field	Description	Allowed range
regs	Pointer to the GRSPW	-
nospill	The nospill value used for the core.	0 - 1
rmap	Indicates whether the core is configured with RMAP. Set by spw_init.	0 - 1
rxunaligned	Indicates whether the core is configured with rxunaligned support. Set by spw_init.	0 - 1
rmapcrc	Indicates whether the core is configured with RMAPCRC support. Set by spw_init.	0 - 1
clkdiv	The clock divisor value used for the core.	0 - 255
nodeaddr	The node address value used for the core.	0 - 255
destkey	The destination key value used for the core.	0 - 255
rxmaxlen	The Receiver maximum length value used for the core.	0 - 33554431
rxpnt	Pointer to the next receiver descriptor.	0 - 127
rxchkpnt	Pointer to the next receiver descriptor that will be polled.	0 - 127
txpnt	Pointer to the next transmitter descriptor.	0 - 63
txchkpnt	Pointer to the next transmitter descriptor that will be polled.	0 - 63
timetxen	The timetxen value used for this core.	0 - 1
timerxen	The timerxen value used for this core.	0 - 1
txd	Pointer to the transmitter descriptor table.	-
rxd	Pointer to the receiver descriptor table	-

The following functions are available in the basic API:

```
int spw_setparam(int nodeaddr, int clkdiv, int destkey, int nospill, int timetxen, int
timerxen, int rxmaxlen, int spwadr, struct spwvars *spw);
```

Used for setting the different parameters in the spwvars struct. Should always be run first after creating a spwvars struct. This function only initializes the struct. Does not write anything to the SpaceWire core.

*Table 1270.*Return values for spw_setparam

Value	Description
0	The function completed successfully
1	One or more of the parameters had an illegal value

*Table 1271.*Parameters for spw_setparam

Parameter	Description	Allowed range
nodeaddr	Sets the node address value of the struct spw passed to the function.	0-255
clkdiv	Sets the clock divisor value of the struct spw passed to the function.	0-255
destkey	Sets the destination key of the struct spw passed to the function.	0-255
nospill	Sets the nospill value of the struct spw passed to the function.	0 - 1
timetxen	Sets the timetxen value of the struct spw passed to the function.	0 - 1
timerxen	Sets the timerxen value of the struct spw passed to the function.	0 - 1
rxmaxlen	Sets the receiver maximum length field of the struct spw passed to the function.	0 - 2^{25}-1
spwadr	Sets the address to the GRSPW core which will be associated with the struct passed to the function.	0 - 2^{32}-1

```
int spw_init(struct spwvars *spw);
```

Initializes the GRSPW core located at the address set in the struct spw. Sets the following registers: node address, destination key, clock divisor, receiver maximum length, transmitter descriptor table address, receiver descriptor table address, ctrl and dmactrl. All bits are set to the values found in the spwvars struct. If a register bit is not present in the struct it will be set to zero. The descriptor tables are allocated to an aligned area using malloc. The status register is cleared and lastly the link interface is enabled. The run state frequency will be set according to the value in clkdiv.

*Table 1272.*Return values for spw_init

Value	Description
0	The function completed successfully
1	One or more of the parameters could not be set correctly or the link failed to initialize.

*Table 1273.*Parameters for spw_init

Parameter	Description	Allowed range
spw	The spwvars struct associated with the GRSPW core that should be initialized.	-

```
int set_txdesc(int pnt, struct spwvars *spw);
```

Sets a new address to the transmitter descriptor table address register. Should only be used when no transmission is active. Also resets the pointers for spw_tx and spw_checktx (Explained in the section for those functions).

*Table 1274.*Return values for spw_txdesc

Value	Description
0	The function completed successfully
1	The new address could not be written correctly

*Table 1275.*Parameters for spw_txdesc

Parameter	Description	Allowed range
pnt	The new address to the descriptor table area	0 - 2^{32}-1
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int set_rxdesc(int pnt, struct spwvars *spw);
```

Sets a new address to the Receiver descriptor table address register. Should only be used when no transmission is active. Also resets the pointers for spw_rx and spw_checkrx (Explained in the section for those functions).

*Table 1276.*Return values for spw_rxdesc

Value	Description
0	The function completed successfully
1	The new address could not be written correctly

*Table 1277.*Parameters for spw_rxdesc

Parameter	Description	Allowed range
pnt	The new address to the descriptor table area	0 - 2^{32}-1
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_disable(struct spwvars *spw);
```

Disables the GRSPW core (the link disable bit is set to ‘1’).

*Table 1278.*Parameters for spw_disable

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_enable(struct spwvars *spw);
```

Enables the GRSPW core (the link disable bit is set to ‘0’).

*Table 1279.*Parameters for spw_enable

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_start(struct spwvars *spw);
```

Starts the GRSPW core (the link start bit is set to ‘1’).

*Table 1280.*Parameters for spw_start

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_stop(struct spwvars *spw);
```

Stops the GRSPW core (the link start bit is set to ‘0’).

*Table 1281.*Parameters for spw_start

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_setclockdiv(struct spwvars *spw);
```

Sets the clock divisor register with the clock divisor value stored in the spwvars struct.

*Table 1282.*Return values for spw_setclockdiv

Value	Description
0	The function completed successfully
1	The new clock divisor value is illegal.

*Table 1283.*Parameters for spw_setclockdiv

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_set_nodeadr(struct spwvars *spw);
```

Sets the node address register with the node address value stored in the spwvars struct.

*Table 1284.*Return values for spw_set_nodeadr

Value	Description
0	The function completed successfully
1	The new node address value is illegal.

*Table 1285.*Parameters for spw_set_nodeadr

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_set_rxmaxlength(struct spwvars *spw);
```

Sets the Receiver maximum length register with the rxmaxlen value stored in the spwvars struct.

*Table 1286.*Return values for spw_set_rxmaxlength

Value	Description
0	The function completed successfully
1	The new node address value is illegal.

*Table 1287.*Parameters for spw_set_rxmaxlength

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_tx(int crc, int skipcrcsize, int hsize, char *hbuf, int dsize, char *dbuf, struct
spwvars *spw);
```

Transmits a packet. Separate header and data buffers can be used. If CRC logic is available the GSPW inserts RMAP CRC values after the header and data fields if crc is set to one. This function only sets a descriptor and initiates the transmission. Spw_checktx must be used to check if the packet has been transmitted. A pointer into the descriptor table is stored in the spwvars struct to keep track of the next location to use. It is incremented each time the function returns 0.

*Table 1288.*Return values for spw_tx

Value	Description
0	The function completed successfully
1	There are no free transmit descriptors currently available
2	There was illegal parameters passed to the function

*Table 1289.*Parameters for spw_tx

Parameter	Description	Allowed range
crc	Set to one to append RMAP CRC after the header and data fields. Only available if hardware CRC is available in the core.	0 - 1
skipcrcsize	The number of bytes in the beginning of a packet that should not be included in the CRC calculation	0 - 15
hsize	The size of the header in bytes	0 - 255
hbuf	Pointer to the header data	-
dsize	The size of the data field in bytes	0 - 2^{24}-1
dbuf	Pointer to the data area.	-
spw	Pointer to the spwvars struct associated with GRSPW core that should transmit the packet	-

```
int spw_rx(char *buf, struct spwvars *spw);
```

Enables a descriptor for reception. The packet will be stored to buf. Spw_checkrx must be used to check if a packet has been received. A pointer in the spwvars struct is used to keep track of the next location to use in the descriptor table. It is incremented each time the function returns 0.

*Table 1290.*Return values for spw_rx

Value	Description
0	The function completed successfully
1	There are no free receive descriptors currently available

*Table 1291.*Parameters for spw_rx

Parameter	Description	Allowed range
buf	Pointer to the data area.	-
spw	Pointer to the spwvars struct associated with GRSPW core that should receive the packet	-

```
int spw_checkrx(int *size, struct rxstatus *rxs, struct spwvars *spw);
```

Checks if a packet has been received. When a packet has been received the size in bytes will be stored in the size parameter and status is found in the rxs struct. A pointer in the spwvars struct is used to keep track of the location in the descriptor table to poll. It is incremented each time the function returns nonzero.

*Table 1292.*Return values for spw_checkrx

Value	Description
0	No packet has been received
1	A packet has been received

*Table 1293.*Parameters for spw_checkrx

Parameter	Description	Allowed range
size	When the function returns 1 this variable holds the number of bytes received	-
rxs	When the function returns 1 this variable holds status information	-
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

Table 1294. The rxstatus struct

Field	Description	Allowed range
truncated	Packet was truncated	0 - 1
dcrcerr	Data CRC error bit was set. Only indicates an error if the packet received was an RMAP packet.	0 - 1
hcrcerr	Header CRC error bit was se.t. Only indicates an error if the packet received was an RMAP packet.	0 - 1
eep	Packet was terminated with EEP	0 - 1

```
int spw_checktx(struct spwvars *spw);
```

Checks if a packet has been transmitted. A pointer is used to keep track of the location in the descriptor table to poll. It is incremented each time the function returns nonzero.

Table 1295. Return values for spw_checktx

Value	Description
0	No packet has been transmitted
1	A packet has been correctly transmitted
2	A packet has been incorrectly transmitted

Table 1296. Parameters for spw_checktx

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
void send_time(struct spwvars *spw);
```

Sends a new time-code. Increments the time-counter in the GRSPW and transmits the value.

Table 1297. Parameters for send time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
int check_time(struct spwvars *spw);
```

Check if a new time-code has been received.

Table 1298. Return values for check_time

Value	Description
0	No time-code has been received
1	A new time-code has been received

Table 1299. Parameters for check_time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
int get_time(struct spwvars *spw);
```

Get the current time counter value.

*Table 1300.*Return values for get_time

Value	Description
0 - 63	Returns the current time counter value

*Table 1301.*Parameters for get_time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
void spw_reset(struct spwvars *spw);
```

Resets the GRSPW.

*Table 1302.*Parameters for spw_reset

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be reset	-

```
void spw_rmapen(struct spwvars *spw);
```

Enables hardware RMAP. Has no effect if the RMAP command handler is not available in GRSPW.

*Table 1303.*Parameters for spw_rmapen

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

```
void spw_rmapdis(struct spwvars *spw);
```

Disables hardware RMAP. Has no effect if the RMAP command handler is not available in GRSPW

*Table 1304.*Parameters for spw_rmapdis

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

```
int spw_setdestkey(struct spwvars *spw);
```

Set the destination key of the GRSPW. Has no effect if the RMAP command handler is not available. The value from the spwvars struct is used.

*Table 1305.*Return values for spw_setdestkey

Value	Description
0	The function completed successfully
1	The destination key parameter in the spwvars struct contains an illegal value

*Table 1306.*Parameters for spw_setdestkey

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set.	-

76.19.2 GRSPW RMAP API

The RMAP API contains only one function which is used for building RMAP headers.

```
int build_rmap_hdr(struct rmap_pkt *pkt, char *hdr, int *size);
```

Builds a RMAP header to the buffer pointed to by hdr. The header data is taken from the rmap_pkt struct.

*Table 1307.*Return values for build_rmap_hdr

Value	Description
0	The function completed successfully
1	One or more of the parameters contained illegal values

*Table 1308.*Parameters for build_rmap_hdr

Parameter	Description	Allowed range
pkt	Pointer to a rmap_pkt struct which contains the data from which the header should be built	
hdr	Pointer to the buffer where the header will be built	
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

*Table 1309.*rmap_pkt struct fields

Field	Description	Allowed Range
type	Selects the type of packet to build.	writecmd, readcmd, rmwcmd, writerep, readrep, rmwrep
verify	Selects whether the data should be verified before writing	yes, no
ack	Selects whether an acknowledge should be sent	yes, no
incr	Selects whether the address should be incremented or not	yes, no
destaddr	Sets the destination address	0 - 255
destkey	Sets the destination key	0 - 255
srcaddr	Sets the source address	0 - 255
tid	Sets the transaction identifier field	0 - 65535
addr	Sets the address of the operation to be performed. The extended address field is currently always set to 0.	0 - 2^{32}-1
len	The number of bytes to be writte, read or read-modify-written	0 - 2^{24}-1
status	Sets the status field	0 - 11
dstspalen	Number of source path address bytes to insert before the destination address	0 - 228
dstspa	Pointer to memory holding the destination path address bytes	-
srcspalen	Number of source path address bytes to insert in a command. For a reply these bytes are placed before the return address	0 - 12
srcspa	Pointer to memory holding the source path address bytes	-

76.20 Appendix A Clarifications of the GRSPW implementation of the standard

6.3.1 page 9 RMAP draft F

"The user application at destination will be informed that there was an error in the data transferred. The source will be informed of the data error if the acknowledge bit in the command has been set."

We view the RMAP command handler as both protocol parser and user application. All commands are parsed in the first stage and various (internal) status bits are set. The next step (which can be viewed as the user application) will make the decision of how to act upon the received command from these bits. Therefore, the various errors that can occur are not externally observable.

If an error occurs when the command handler is accessing the AHB bus through the DMA interface errors will be externally observable using the AHB status register in GRLIB.

6.3.6 page 13 RMAP draft F

“The Write Command packet arrives at the destination and its header is found to be in error. This fact is added to the error statistics in the destination node.”

This text does not state how and if these statistics should be observable. At the moment the error handling is internal to the RMAP command handler and therefore no statistics are internally observable. A counter for this particular error might be added in the future.

6.3.6 page 15 RMAP draft F

"These various errors will be reported to the user application running on the destination node (Write Data Error Indication)."

Again the RMAP command handler is the user application and all these errors are handled internally.

6.5.6 page 31 RMAP draft F

"The source user application, in fact immediately rejects this as an authorisation failure as the command is trying to RMW an area of protected memory."

It should probably be destination user application instead of source. It is unclear what immediately means. Should it be rejected before any accesses are done on the bus and thus requiring the RMAP command handler to include a complete bus decoding. The GRSPW does (probably) not comply to this paragraph at the moment. If a bus error occurs a general error code will be returned.

6.5.6 page 32

"If the header of the RMW reply packet is received intact but the data field is corrupted as indicated by an incorrect data field length (too long or too short) or by a CRC error, then an error can be flagged to the application immediately (RMW Data Failure) without having to wait for an application timeout."

This is not applicable to the GRSPW since it does not handle replies. However this is practically an unnecessary comment since it is not specified in the standard in which manner received replies are indicated to the higher layers.

77 GRSPW2 - SpaceWire codec with AHB host Interface and RMAP target

77.1 Overview

The SpaceWire core provides an interface between the AHB bus and a SpaceWire network. It implements the SpaceWire standard (ECSS-E-ST-50-12C) with the protocol identification extension (ECSS-E-ST-50-51C). The optional Remote Memory Access Protocol (RMAP) target implements the ECSS standard (ECSS-E-ST-50-52C).

The SpaceWire interface is configured through a set of registers accessed through an APB interface. Data is transferred through DMA channels using an AHB master interface. The number of DMA channels is configurable from one to four.

The core can also be configured with two SpaceWire ports with manual or automatic switching between them.

There can be up to four clock domains: one for the AHB interface (system clock), one for the transmitter and one or two for the receiver depending on the number of configured ports.

The core supports byte addressed 32-bit big-endian and little-endian host systems. Transmitter outputs can be either Single Data Rate (SDR) or Double Data Rate (DDR). The receiver can be connected either to an Frontgrade SpaceWire transceiver or recover the data itself using a self-clocking scheme or sampling (SDR or DDR).

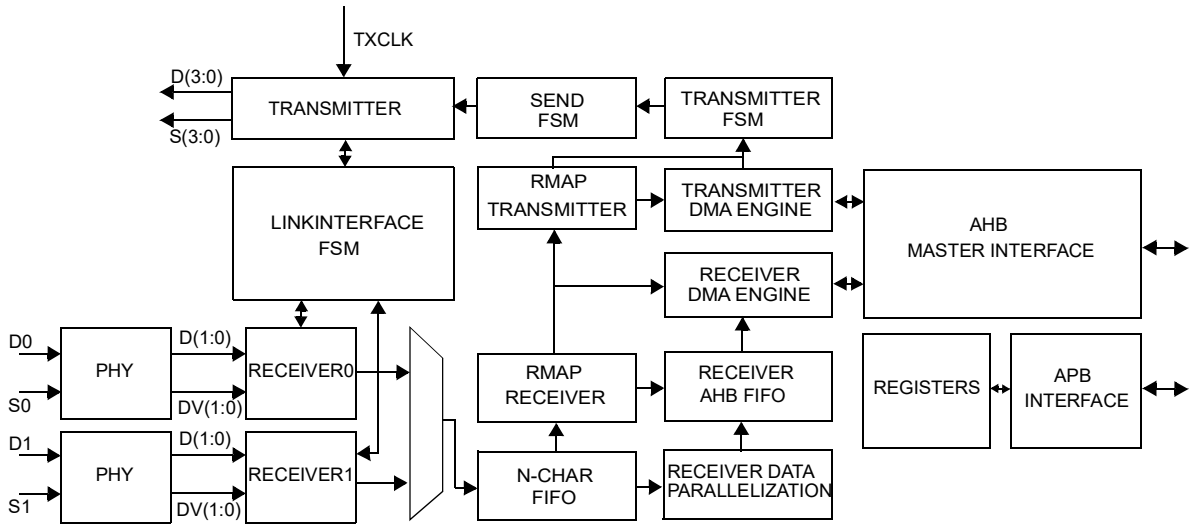


Figure 195. Block diagram

77.2 Operation

77.2.1 Overview

The main sub-blocks of the core are the link interface, the RMAP target and the AMBA interface. A block diagram of the internal structure can be found in figure 195.

The link interface consists of the receiver, transmitter, and the link interface FSM. They handle communication on the SpaceWire network. The PHY block provides a common interface for the receiver to the four different data recovery schemes and is external to this core. A short description is found in section 77.3.5. The complete documentation is found in the GRSPW2_PHY section of GRLIB IP Core User's Manual. The AMBA interface consists of the DMA engines, the AHB master interface and the APB interface. The link interface provides FIFO interfaces to the DMA engines. These FIFOs

are used to transfer N-Chars between the AMBA and SpaceWire domains during reception and transmission.

The RMAP target is an optional part of the core which can be enabled with a VHDL generic. The RMAP target handles incoming packets which are determined to be RMAP commands instead of the receiver DMA engine. The RMAP command is decoded and if it is valid, the operation is performed on the AHB bus. If a reply was requested it is automatically transmitted back to the source by the RMAP transmitter.

The link interface, DMA engines, RMAP target and AMBA interface are described in section 77.3, 77.6, 77.8 and 77.9 respectively.

77.2.2 Protocol support

The core only accepts packets with a valid destination address in the first received byte. Packets with address mismatch will be silently discarded (except in promiscuous mode, which is covered in section 77.6.10).

The second byte is sometimes interpreted as a protocol ID a described hereafter. The RMAP protocol (ID=0x1) is the only protocol handled separately in hardware while other packets are stored to a DMA channel. If the RMAP target is present and enabled all RMAP commands will be processed, executed and replied automatically in hardware. Otherwise RMAP commands are stored to a DMA channel in the same way as other packets. RMAP replies are always stored to a DMA channel. More information on the RMAP protocol support is found in section 77.8. When the RMAP target is not present or disabled, there is no need to include a protocol ID in the packets and the data can start immediately after the address.

All packets arriving with the extended protocol ID (0x00) are stored to a DMA channel. This means that the hardware RMAP target will not work if the incoming RMAP packets use the extended protocol ID. Note also that packets with the reserved extended protocol identifier (ID = 0x000000) are not ignored by the core. It is up to the client receiving the packets to ignore them.

When transmitting packets, the address and protocol-ID fields must be included in the buffers from where data is fetched. They are *not* automatically added by the core.

Figure 196 shows the packet types accepted by the core. The core also allows reception and transmission with extended protocol identifiers but without support for RMAP CRC calculations and the RMAP target.

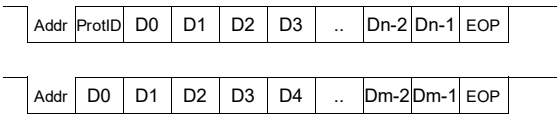


Figure 196. The SpaceWire packet types supported by the core.

When a package is received with a protocol ID = 0x02 (CCSDS) the core can, if enabled, automatically calculate and check the CCSDS/CCITT CRC-16 and 16-bit ISO-checksum (J.G. Fletcher, ISO 8473-1:1998) assuming this is included in the package.

77.2.3 Endianness

The core is designed for big-endian and little-endian systems.

GRSPW2 automatically detects the endianness of the system by reading a dedicated sideband signal included in the AMBA records. When accessing the AHB bus, the data may be swapped, depending on the selected endianness and the type of data involved in the transfer:

- Words belonging to TX and RX descriptors are never swapped and shall be read and written using 32-bit accesses. These words shall be located at the same address offsets (0x0, 0x4, 0x8 and 0xC) regardless of the endianness.
- Data in standard SpaceWire packets is treated as a stream of bytes, i.e. the bytes will be located in the same memory positions in both big- and little-endian systems. If *addr* is the address configured in the corresponding field of a descriptor, the first byte shall be located at *addr*, the second byte at *addr+1*, and so on. This implies that GRSPW2 will internally swap the AHB words in little-endian configurations when accessing the AMBA bus.
- Data in RMAP packets is also treated as a stream of bytes for all commands (Read, Write and Read-Modify-Write). The first data byte of an RMAP packet is associated with the address in the packet. GRSPW2 will swap the data bytes in little-endian configurations. Note: the user shall take the endianness of the host system into account. This means that, when accessing APB registers and 32-bit descriptors via RMAP, the user shall swap the bytes of the command in little-endian host systems.

77.3 Link interface

The link interface handles the communication on the SpaceWire network and consists of a transmitter, receiver, a FSM and FIFO interfaces. An overview of the architecture is found in figure 195.

77.3.1 Link interface FSM

The FSM controls the link interface (a more detailed description is found in the SpaceWire standard). The low-level protocol handling (the signal and character level of the SpaceWire standard) is handled by the transmitter and receiver while the FSM handles the exchange level.

The link interface FSM is controlled through the Control register (CTRL). The link can be disabled through the CTRL.LD bit, which depending on the current state, either prevents the link interface from reaching the started state or forces it to the error-reset state. When the link is not disabled, the link interface FSM is allowed to enter the started-state when either the CTRL.LS bit is set or when a NULL character has been received and the CTRL.AS bit is set.

The state of the link interface determines which type of characters that are allowed to be transmitted, which together with the requests made from the host interfaces determine what character will be sent.

Time-codes are sent when the FSM is in the run-state and a request is made through the time-interface (described in section 77.4).

When the link interface is in the connecting- or run-state it is allowed to send FCTs. FCTs are sent automatically by the link interface when possible. This is done based on the maximum value of 56 for the outstanding credit counter and the currently free space in the receiver N-Char FIFO. FCTs are sent as long as the outstanding counter is less than or equal to 48 and there are at least 8 more empty FIFO entries than the counter value.

N-Chars are sent in the run-state when they are available from the transmitter FIFO and there are credits available. NULLs are sent when no other character transmission is requested, or when the FSM is in a state where no other transmissions are allowed.

The credit counter (incoming credits) is automatically increased when a FCTs is received, and decreased when N-Chars are transmitted. Received N-Chars are stored to the receiver N-Char FIFO for further handling by the DMA interface. Received Time-codes are handled by the time-interface.

77.3.2 Transmitter

The state of the FSM, credit counters, requests from the time-interface and requests from the DMA-interface are used to decide the next character to be transmitted. The type of character and the character itself (for N-Chars and Time-codes) to be transmitted are presented to the low-level transmitter which is located in a separate clock-domain.

This is done because one usually wants to run the SpaceWire link on a different frequency than the host system clock. The core has a separate clock input which is used to generate the transmitter clock. More information on transmitter clock generation is found in section 77.11.2. Since the transmitter often runs on high frequency clocks (> 100 MHz) as much logic as possible has been placed in the system clock domain to minimize power consumption and timing issues.

The transmitter logic in the host clock domain decides what character to send next and sets the proper control signal and presents any needed character to the low-level transmitter as shown in figure 197. The transmitter sends the requested characters and generates parity and control bits as needed. If no requests are made from the host domain, NULLs are sent as long as the transmitter is enabled. Most of the signal and character levels of the SpaceWire standard is handled in the transmitter. External LVDS drivers are needed for the data and strobe signals. The outputs can be configured as either single- or double data rate. The latter increases maximum bitrate significantly but is not available for all technologies.

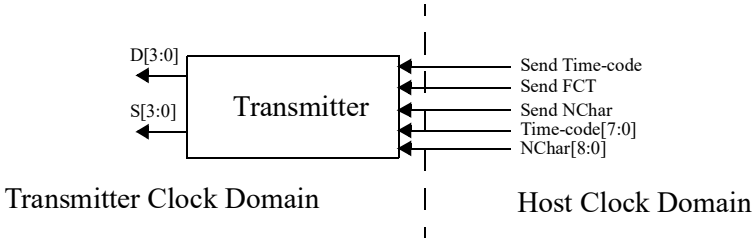


Figure 197. Schematic of the link interface transmitter. Some bits in D[3:0] and S[3:0] may be unused, depending on the values of the *output_type*, and *ports* VHDL generics (see section 70.15).

A transmission FSM reads N-Chars for transmission from the transmitter FIFO. It is given packet lengths from the DMA interface and appends EOPs/EEPs and RMAP/CCSDS-CRC (or ISO-checksum) values if requested. When it is finished with a packet the DMA interface is notified and a new packet length value is given.

77.3.3 Receiver

The receiver detects connections from other nodes and receives characters as a bit stream recovered from the data and strobe signals by the GRSPW2_PHY module, which presents it as a data and data-valid signal. The receiver and GRSPW2_PHY are located in a separate clock domain which runs on a clock driven by the GRSPW2_PHY. More information on the clock-generation can be found in section 77.11.2.

The receiver is activated as soon as the link interface leaves the error reset state. Then after a NULL is received it can start receiving any characters. It detects parity, escape and credit errors which causes the link interface to enter the error reset state. Disconnections are handled in the link interface part in the tx clock domain because no receiver clock is available when disconnected.

Received Characters are flagged to the host domain and the data is presented in parallel form. The interface to the host domain is shown in figure 198. L-Chars are the handled automatically by the host domain link interface part while all N-Chars are stored in the receiver FIFO for further handling. If two or more consecutive EOPs/EEPs are received all but the first one are discarded.

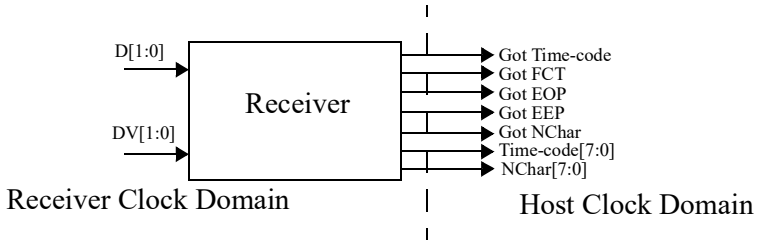


Figure 198. Schematic of the link interface receiver.

77.3.4 Dual port support

The core can be configured to include an additional SpaceWire port. With dual ports the transmitter drives an additional pair of data/strobe output signals and one extra receiver is added to handle a second pair of data/strobe input signals.

One of the ports is set as active (how the active port is selected is explained below) and the transmitter drives the data/strobe signals of the active port with the actual output values as explained in section 77.3.2. The inactive port is driven with zero on both data and strobe.

Both receivers will always be active but only the active port's interface signals (see figure 198) will be propagated to the link interface FSM. Each time the active port is changed, the link will be reset so that the new link is started in a controlled manner.

When the CTRL.NP bit is zero, the CTRL.PS bit selects the active port. When the CTRL.NP bit is set to one, the active port is automatically selected during initialization. For the latter mode, the port on which the first bit is received will be selected as the active port. If the initialization attempt fails on that port the link is reset and the active port is again sected based on which port the first bit is received.

77.3.5 Receiver PHY

The receiver supports four different input data recovery schemes: self-clocking (xor), sampling SDR, sampling DDR and the Frontgrade SpaceWire transceiver. These four recovery types are handled in the PHY module and data is presented to the receiver as a data (D) and data-valid (DV) signal. This part of the receiver must often be constrained and placing it in a separate module makes this process easier with the most common synthesis tools. The input type is selected using the VHDL generic *input_type*. More information about the PHY can be found in the GRSPW2_PHY section of the GRLIP IP Core User's Manual.

77.3.6 Setting link-rate

The register field CLKDIV.CLKDIVSTART determines the link-rate during initialization (all states up to and including the connecting-state). The register is also used to calculate the link interface FSM timeouts (6.4 µs and 12.8 µs, as defined in the SpaceWire standard). The CLKDIV.CLKDIVSTART field should always be set so that a 10 Mbit/s link-rate is achieved during initialization. In that case the timeout values will also be calculated correctly.

To achieve a 10 Mbit/s link-rate, the CLKDIV.CLKDIVSTART field should be set according to the following formulas:

With single data rate (SDR) outputs:

CLKDIV.CLKDIVSTART = (<frequency in MHz of TXCLK> / 10) - 1

With double data rate (DDR) outputs, or when connected to Frontgrade SpaceWire transceiver:

CLKDIV.CLKDIVSTART = (2 x <frequency in MHz of TXCLK> / 10) - 1

The link-rate in run-state is controlled with the run-state divisor, the CLKDIV.CLKDIVRUN register field. The link-rate in run-state is calculated according to the following formulas:

With SDR outputs:

<link-rate in Mbits/s> = <frequency in MHz of TXCLK> / (CLKDIV.CLKDIVRUN+1)

With DDR outputs / Frontgrade SpaceWire transceiver:

<link-rate in Mbits/s> = 2 x <frequency in MHz of TXCLK> / (CLKDIV.CLKDIVRUN+1)

The value of CLKDIV.CLKDIVRUN only affects the link-rate in run-state, and does not affect the 6.4 µs or 12.8 µs timeouts values.

Note that when using DDR ouputs, or when connected to Frontgrade SpaceWire transceiver, there is a limitation in the usable clock divisor values. All even values (except 0) will result in the same bitrate as the one higher odd number.

An example of clock divisor and resulting link-rate, with a TXCLK frequency of 50 MHz, is shown in the table 1310. Also see 77.11.2 for information on clock requirements.

Table 1310. SpaceWire link-rate example with 50 MHz TXCLK

	Link-rate in Mbit/s	
Clock divisor value	**SDR outputs**	**DDR outputs / Frontgrade SpaceWire transceiver**
0	50	100
1	25	50
2	16.67	25
3	12.5	25
4	10	16.67
5	8.33	16.67
6	7.14	12.5
7	6.25	12.5
8	5.56	10
9	5	10

77.4 Time-code distribution

Time-codes are control codes that consists of two control flags (bits 7:6) and a time value (bits 5:0), and they are used to distribute time over the SpaceWire network. The current time value (value of latest received or transmitted time-code), and control flags, can be read from the Time-code register (TC).

77.4.1 Receiving time-codes

When a control-code is received, and either the control flags (bits 7:6) have value “00”, or both control flag filtering and interrupt receive is disabled (CTRL.TF bit, and INTCFG.IR bit both set to 0), then the received control code is considered to be a Time-Code. If Time-Code reception is enabled (CTRL.TR bit set to 1) then the received time value is stored in the TC.TIMECNT field. If the received time value equals TC.TIMECNT+1 (modulo 64), then the Time-Code is considered valid.

When a valid Time-Code is received, in addition to the time value being updated, the received control flags are stored to the TC.TCTRL field. Also, when a valid Time-Code is received, the TICKOUT output signal is asserted for one system clock cycle, the STS.TO bit is set to 1, and an AMBA interrupt is generated if the CTRL.IE bit and CTRL.TQ bit are both set to 1.

For all received control codes, Time-Codes or not, the control flags together with the time value are outputted on the TIMEOUT[7:0] signals, and the TICKOUTRAW signal is asserted for one system clock cycle.

77.4.2 Transmitting time-codes

Time-codes can be transmitted either through the AMBA APB registers or through the signals TICKIN, TICKINRAW, TIMEIN, and TICKINDONE.

In order to send a Time-code, Time-Code transmission must be enabled by setting the CTRL.TT bit to 1. To transmit a time-code through the register interface the CTRL.TI bit should be written to 1. When the bit is written the current time value (TC.TIMECNT field) is incremented, and a Time-Code consisting of the new time value together with the current control flags (TC.TCTRL field) is sent. The CTRL.TI bit will stay high until the Time-Code has been transmitted. If time-code transmission is disabled, writing the CTRL.TI bit has no effect.

To transmit a time-code using the TICKIN signal the sender must wait until the TICKINDONE output is low, then assert TICKIN. When TICKINDONE is asserted again, the TICKIN signal should be de-asserted the same cycle. Following this procedure will make the core transmit a Time-Code consisting of the current control flags and the current time value + 1 (modulo 64). This also requires that time-code transmission is enabled through the CTRL.TT bit.

To transmit a Time-Code using the TICKINRAW signal the sender must wait until TICKINDONE is low, then assert TICKINRAW and place the value of the Time-Code to be sent on the TIMEIN[7:0] signals. When TICKINDONE is asserted again, the TICKINRAW signal should be de-asserted the same cycle. Note that sending Time-Codes by using TICKINRAW does not require that Time-Code transmission is enabled from the Control register. However, in order to send Time-Codes with control flags different than “00”, interrupt transmit must be disabled (INTCTRL.IT bit set to 0). If interrupt transmit is enabled then control codes “10” are interpreted as interrupt-codes, while control codes “01” and “11” are discarded.

Note that the link interface must be in run-state in order to be able to send a Time-Code.

77.5 Interrupt distribution

The core supports interrupt distribution functionality. Whether or not this functionality is implemented is indicated by the CTRL.ID bit, and the number of supported interrupt numbers is indicated by the INTCFG.NUMINT field. Either 1, 2, 4, or 32 interrupt numbers (in the range 0-31) can be supported. When less than 32 interrupt numbers are supported it is programmable through the INTCFG register which interrupt numbers in the range 0-31 that are allowed to be sent and received. When extended interrupt mode is enabled (INTCFG.EE bit set to 1), the supported interrupt number in “interrupt mode” is extended to 0-63).

The interrupts are distributed as control codes with the control flags (bits 7:6) set to “10”. Bit 5 of the control code specifies if the code is an interrupt-code (bit 5 = ‘0’) or an interrupt-acknowledge-code (bit 5 = ‘1’). An interrupt-code is generated by the source of the interrupt event, while the interrupt-acknowledge-code is sent by the interrupt handler for the corresponding interrupt number. When extended interrupt mode is enabled (INTCFG.EE bit set to 1), then interrupt-acknowledge-code is interpret as interrupt-codes in the range 32-63.

An Interrupt distribution ISR register holds the current state of all the interrupt numbers in the SpaceWire network. A bit in the ISR register is set to 1 when an interrupt-code with the corresponding interrupt number is received / transmitted, and the bit is set to 0 when an interrupt-acknowledge-code with the corresponding interrupt number is received / transmitted.

Each interrupt number also has its own timer that is used to clear the ISR bit if an interrupt-acknowledge-code is not received before the timer expires. There is also a timer for each interrupt-number that controls the minimum time between and interrupt-code and interrupt-acknowledge-code (and vice

versa), in order to allow propagation of the codes through the whole network before a new code with the same interrupt number is sent.

77.5.1 Interrupt distribution timers

Each interrupt number has three corresponding timers, called the ISR timer, INT/ACK-timer, and ISR change timer. Whether or not these timers are implemented in hardware, and how large they are, can be detected by probing the ISRTIMER, IATIMER, and ICTIMER registers respectively.

If the ISR timers are enabled (ISRTIMER.EN bit set to 1), the ISR timer is started and reloaded with the value from the ISRTIMER.RL field each time an interrupt-code is received such that the corresponding ISR bit is set to 1. If a matching interrupt-acknowledge-code is received, the corresponding ISR timer is stopped. If the ISR timer expires before an interrupt-acknowledge-code is received, the corresponding ISR bit is cleared. The purpose of the ISR timer is to recover from situations where an interrupt-acknowledge-code is lost. If an interrupt-acknowledge-code is lost and there were no ISR timer, then the corresponding ISR bit would stay set forever, and prevent future interrupt-codes with that interrupt number to be distributed. It is important to configure the reload value for the ISR timer correctly. The reload value shall not be less than the worst network propagation delay for the interrupt-code, plus the maximum delay in the interrupt handler, plus the worst network propagation delay for the interrupt-acknowledge-code. Note that use of the ISR timer is mandatory, so if the hardware timers are either disabled or not implemented, software must handle the timers.

The INT/ACK-timer is used to control the minimum time between an interrupt-code and interrupt-acknowledge-code with the same interrupt number, and vice versa. The purpose of the INT/ACK-timer is to make sure that each interrupt- / interrupt-acknowledge-code gets enough time to propagate through the complete network before the next interrupt- / interrupt-acknowledge-code is sent, ensuring that no interrupt- / interrupt-acknowledge-code is received out of order. If the INT/ACK-timers are enabled (IATIMER.EN bit set to 1), then each time an interrupt- / interrupt-acknowledge-code is received the corresponding INT/ACK-timer is started and reloaded with the value from the IATIMER.RL field. As long as the timer is running, an interrupt- / interrupt-acknowledge-code with that interrupt number will not be sent.

The ISR change timer is used to control the minimum time between two consecutive changes to the same ISR bit. The purpose of the timer is to protect against unexpected occurrences of interrupt- / interrupt-acknowledge-codes that could occur, for example, due to a network malfunction or a babbling idiot. If the ISR change timers are enabled (ICTIMER.EN bit set to 1), then the timer for an ISR bit is started and reloaded with the value from the ICTIMER.RL field each time a received interrupt- / interrupt-acknowledge-code makes the ISR bit change value. Until the timer has expired, the corresponding ISR bit is not allowed to change value, and any received interrupt- / interrupt-acknowledge-codes with that interrupt number are discarded.

77.5.2 Receiving interrupt- / interrupt-acknowledge-codes

When a control code with control flags set to “10” is received, and interrupt receive is enable (IR bit in Interrupt distribution control register set to 1), the control code is considered an interrupt-code if bit 5 is 0, and an interrupt-acknowledge-code if bit 5 is 1. If an interrupt-code is received and the interrupt number’s corresponding ISR bit is already set to 1, or an interrupt-acknowledge-code when the ISR bit is 0, then the received interrupt- / interrupt-acknowledge-code is discarded without any further action.

When an interrupt-code is received, and the corresponding ISR bit is 0, the ISR bit is set to 1. If the interrupt number’s corresponding bit in the Interrupt tick-out mask register is set to 1 then the corresponding bit in the Interrupt-code receive register is set to 1, the TICKOUT signal is asserted for one clock cycle (if the INTCTRL.IT bit is 1), and an AMBA interrupt is generated (if the IE bit in the Control register, and IQ bit in the Interrupt distribution control register are both set to 1). If the interrupt number’s corresponding bit in the Interrupt-code auto acknowledge mask register is set to 1, then

an interrupt-acknowledge-code will be automatically sent once the INT/ACK-timer has expired, and the ISR bit will be cleared again.

When an interrupt-acknowledge-code is received, and the corresponding ISR bit is 1, the ISR bit is set to 0. If the interrupt number's corresponding bit in the Interrupt tick-out mask register is set to 1, and the interrupt-code that made the ISR bit get set to 1 in the first place was sent by software (through register access), then the corresponding bit in the Interrupt-acknowledge-code receive register is set to 1. The TICKOUT signal is asserted for one clock cycle as well (if the INTCTRL.AT bit is 1), and an AMBA interrupt is generated (if the IE bit in the Control register, and IQ bit in the Interrupt distribution control register are both set to 1).

Note that all received control codes, interrupt- / interrupt-acknowledge-codes or not, are outputted on the TIMEOUT[7:0] signals, and the TICKOUTRAW signal is asserted for one clock cycle.

For more details regarding interrupt- / interrupt-acknowledge-code reception, please see the description of the interrupt distribution registers in section 77.13.

77.5.3 Transmitting interrupt- / interrupt-acknowledge-codes

Interrupt- / interrupt-acknowledge-codes can be transmitted either through the AMBA APB registers or through the signals TICKINRAW, TIMEIN, and TICKINDONE.

To transmit an interrupt- / interrupt-acknowledge-code through the register interface the II bit in the Interrupt distribution control register should be written to 1. When the bit is written the value of the TXINT field determine which interrupt- / interrupt-acknowledge-code that will be sent.

To transmit an interrupt- / interrupt-acknowledge-code using the TICKINRAW signal the sender must wait until TICKINDONE is low, then assert TICKINRAW and place the value of the interrupt- / interrupt-acknowledge-code to be sent on the TIMEIN[7:0] signals. When TICKINDONE is asserted again, the TICKINRAW signal should be de-asserted the same cycle.

Both methods of sending an interrupt- / interrupt-acknowledge-code requires that interrupt transmission is enabled (IT bit in Interrupt distribution control register set to 1). The actual sending of the interrupt- / interrupt-acknowledge-code is delayed until the corresponding INT/ACK-timer has expired.

For more details regarding interrupt- / interrupt-acknowledge-code transmission, please see the description of the interrupt distribution registers in section 77.13.

77.5.4 Interrupt-code generation

Interrupt-codes can be generated automatically due to a number of internal events. Which events that should force an interrupt-code to be sent, and what interrupt-number to use, is controlled from the Interrupt distribution control register, and the DMA control/status register. Interrupt transmission must also be enabled (IT bit in Interrupt distribution control register) for interrupt-codes to be generated. Internally generated interrupt-codes are sent in the same manner as interrupt-codes transmitted through the register interface and the TICKINRAW signal, as described in section 77.5.3. For more details regarding interrupt-code generation please see the description of the Interrupt distribution control register and DMA control/status register in section 77.13.

77.6 Receiver DMA channels

The receiver DMA engine handles reception of data from the SpaceWire network to different DMA channels.

77.6.1 Address comparison and channel selection

Packets are received to different channels based on the address and whether a channel is enabled or not. When the receiver N-Char FIFO contains one or more characters, N-Chars are read by the receiver DMA engine. The first character is interpreted as the logical address and is compared with

the addresses of each channel starting from 0. The packet will be stored to the first channel with an matching address. The complete packet including address and protocol ID but excluding EOP/EEP is stored to the memory address pointed to by the descriptors (explained later in this section) of the channel.

Each SpaceWire address register has a corresponding mask register. Only bits at an index containing a zero in the corresponding mask register are compared. This way a DMA channel can accept a range of addresses. There is a Default address register which is used for address checking in all implemented DMA channels that do not have separate addressing enabled and for RMAP commands in the RMAP target. With separate addressing enabled the DMA channels' own address/mask register pair is used instead.

If an RMAP command is received it is only handled by the target if the Default address register (including mask) matches the received address. Otherwise the packet will be stored to a DMA channel if one or more of them has a matching address. If the address does not match neither the default address nor one of the DMA channels' separate register, the packet is still handled by the RMAP target if enabled since it has to return the invalid address error code. The packet is only discarded (up to and including the next EOP/EEP) if an address match cannot be found and the RMAP target is disabled.

Packets, other than RMAP commands, that do not match neither the default address register nor the DMA channels' address register will be discarded. Figure 199 shows a flowchart of packet reception.

At least 2 non EOP/EEP N-Chars needs to be received for a packet to be stored to the DMA channel unless the promiscuous mode is enabled in which case 1 N-Char is enough. If it is an RMAP packet with hardware RMAP enabled 3 N-Chars are needed since the command byte determines where the packet is processed. Packets smaller than these sizes are discarded.

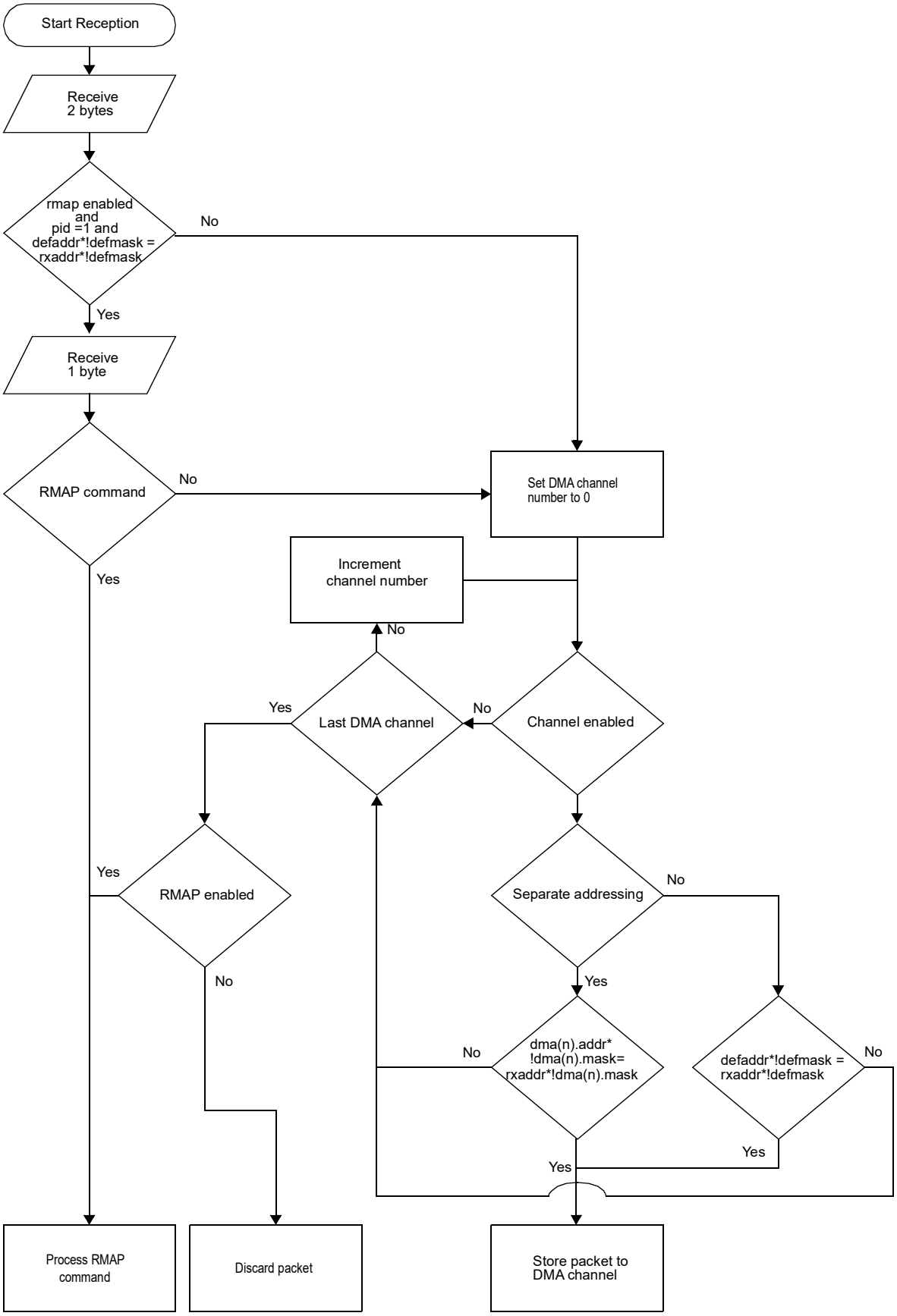


Figure 199. Flow chart of packet reception.

77.6.2 Basic functionality of a channel

Reception is based on descriptors located in a consecutive area in memory that hold pointers to buffers where packets should be stored. When a packet arrives at the core the channel which should receive it is first determined as described in the previous section. A descriptor is then read from the channels' descriptor area and the packet is stored to the memory area pointed to by the descriptor. Lastly, status is stored to the same descriptor and increments the descriptor pointer to the next one. The following sections will describe DMA channel reception in more detail.

77.6.3 Setting up the core for reception

A few registers need to be initialized before reception to a channel can take place. First the link interface need to be put in the run state before any data can be sent. The DMA channel has a maximum length register which sets the maximum packet size in bytes that can be received to this channel. Larger packets are truncated and the excessive part is spilled. If this happens an indication will be given in the status field of the descriptor. The minimum value for the receiver maximum length field is 4 and the value can only be incremented in steps of four bytes up to the maximum value 33554428. If the maximum length is set to zero the receiver will *not* function correctly.

Either the Default address register or the channel specific address register (the accompanying mask register must also be set) needs to be set to hold the address used by the channel. A control bit in the DMA channel control register determines whether the channel should use default address and mask registers for address comparison or the channel's own registers. Using the default register the same address range is accepted as for other channels with default addressing and the RMAP target while the separate address provides the channel its own range. If all channels use the default registers they will accept the same address range and the enabled channel with the lowest number will receive the packet.

Finally, the descriptor table and Control register must be initialized. This will be described in the two following sections.

77.6.4 Setting up the descriptor table address

The core reads descriptors from an area in memory pointed to by the receiver descriptor table address register. The register consists of a base address and a descriptor selector. The base address points to the beginning of the area and must start on an address that is aligned to the size of the descriptor table. The size of the descriptor table can be determined from the formula: STS.NRXD*8. The STS.NRXD field shows the number of entries in the descriptor table, and each descriptor size is 8 bytes.

The descriptor selector points to individual descriptors and is increased by 1 when a descriptor has been used. When the selector reaches the upper limit of the area it wraps to the beginning automatically. It can also be set to wrap at a specific descriptor before the upper limit by setting the wrap bit in the descriptor. The idea is that the selector should be initialized to 0 (start of the descriptor area) but it can also be written with another 8 bytes aligned value to start somewhere in the middle of the area. It will still wrap to the beginning of the area.

If one wants to use a new descriptor table the receiver enable bit has to be cleared first. When the rxactive bit for the channel is cleared it is safe to update the descriptor table register. When this is finished and descriptors are enabled the receiver enable bit can be set again.

77.6.5 Enabling descriptors

As mentioned earlier one or more descriptors must be enabled before reception can take place. Each descriptor is 8 byte in size and the layout can be found in the tables below. The descriptors should be written to the memory area pointed to by the receiver descriptor table address register. When new descriptors are added they must always be placed after the previous one written to the area. Otherwise they will not be noticed.

A descriptor is enabled by setting the address pointer to point at a location where data can be stored and then setting the enable bit. The WR bit can be set to cause the selector to be set to zero when reception has finished to this descriptor. IE should be set if an interrupt is wanted when the reception has finished. The DMA control register interrupt enable bit must also be set for an interrupt to be generated.

The descriptor packet address should be word aligned. All accesses on the bus are word accesses so complete words will always be overwritten regardless of whether all 32-bit contain received data. Also if the packet does not end on a word boundary the complete word containing the last data byte will be overwritten. If the *rxunaligned* or *rmap* VHDL generics are set to 1, this restriction is removed and any number of bytes can be received to any packet address without excessive bytes being overwritten.

Table 1311. GRSPW2 receive descriptor word 0 (address offset 0x0)

31	30	29	28	27	26	25	24 … 0
TR	DC	HC	EP	IE	WR	EN	PACKETLENGTH

31	Truncated (TR) - Packet was truncated due to maximum length violation.
30	Data CRC (DC) - 1 if a CRC error was detected for the data and 0 otherwise. If the Protocol ID of the received package is 0x02 a 1 indicates a CCSDS/CCITT CRC-16 error was detected.
29	Header CRC (HC) - 1 if a CRC error was detected for the header and 0 otherwise. If the Protocol ID of the received package is 0x02 a 1 indicates a 16-bit ISO-Checksum error was detected.
28	EEP termination (EP) - This packet ended with an Error End of Packet character.
27	Interrupt enable (IE) - If set, an interrupt will be generated when a packet has been received if the receive interrupt enable bit in the DMA channel control register is set.
26	Wrap (WR) - If set, the next descriptor used by the GRSPW2 will be the first one in the descriptor table (at the base address). Otherwise the descriptor pointer will be increased with 0x8 to use the descriptor at the next higher memory location. The descriptor table is limited to 1 kbytes in size and the pointer will be automatically wrap back to the base address when it reaches the 1 kbytes boundary.
25	Enable (EN) - Set to one to activate this descriptor. This means that the descriptor contains valid control values and the memory area pointed to by the packet address field can be used to store a packet.
24: 0	Packet length (PACKETLENGTH) - The number of bytes received to this buffer. Only valid after EN has been set to 0 by the GRSPW2.

Table 1312. GRSPW2 receive descriptor word 1 (address offset 0x4)

31 … 0
PACKETADDRESS

31: 0	Packet address (PACKETADDRESS) - The address pointing at the buffer which will be used to store the received packet. If the *rxunaligned* and *rmap* VHDL generics are both set to zero only bit 31 to 2 are used.

77.6.6 Setting up the DMA control register

The final step to receive packets is to set the control register in the following steps: The receiver must be enabled by setting the rxen bit in the DMA control register (see section 77.13.7). This can be done anytime and before this bit is set nothing will happen. The rxdescav bit in the DMA control register is then set to indicate that there are new active descriptors. This must always be done after the descriptors have been enabled or the core might not notice the new descriptors. More descriptors can be activated when reception has already started by enabling the descriptors and writing the rxdescav bit. When these bits are set reception will start immediately when data is arriving.

77.6.7 The effect to the control bits during reception

When the receiver is disabled all packets going to the DMA-channel are discarded if the packet's address does not fall into the range of another DMA channel. If the receiver is enabled and the address falls into the accepted address range, the next state is entered where the rxdescav bit is checked. This bit indicates whether there are active descriptors or not and should be set by the external application using the DMA channel each time descriptors are enabled as mentioned above. If the rxdescav bit is '0' and the nospill bit is '0' the packets will be discarded. If nospill is '1' the core waits until rxdescav is set and the characters are kept in the N-Char fifo during this time. If the fifo becomes full further N-char transmissions are inhibited by stopping the transmission of FCTs.

When rxdescav is set the next descriptor is read and if enabled the packet is received to the buffer. If the read descriptor is not enabled, rxdescav is set to '0' and the packet is spilled depending on the value of nospill.

The receiver can be disabled at any time and will stop packets from being received to this channel. If a packet is currently received when the receiver is disabled the reception will still be finished. The rxdescav bit can also be cleared at any time. It will not affect any ongoing receptions but no more descriptors will be read until it is set again. Rxdescav is also cleared by the core when it reads a disabled descriptor.

77.6.8 Status bits

When the reception of a packet is finished the enable bit in the current descriptor is set to zero. When enable is zero, the status bits are also valid and the number of received bytes is indicated in the length field. The DMA control register contains a status bit which is set each time a packet has been received. The core can also be made to generate an interrupt for this event.

RMAP CRC logic is included in the implementation if the *rmapcrc* or *rmap* VHDL generics are set to 1. The RMAP CRC calculation is always active for all received packets and all bytes except the EOP/EEP are included. The packet is always assumed to be an RMAP packet and the length of the header is determined by checking byte 3 which should be the command field. The calculated CRC value is then checked when the header has been received (according to the calculated number of bytes) and if it is non-zero the HC bit is set indicating a header CRC error.

The CRC value is not set to zero after the header has been received, instead the calculation continues in the same way until the complete packet has been received. Then if the CRC value is non-zero the DC bit is set indicating a data CRC error. This means that the core can indicate a data CRC error even if the data field was correct when the header CRC was incorrect. However, the data should not be used when the header is corrupt and therefore the DC bit is unimportant in this case. When the header is not corrupted the CRC value will always be zero when the calculation continues with the data field and the behaviour will be as if the CRC calculation was restarted

If the received packet is not of RMAP type the header CRC error indication bit cannot be used. It is still possible to use the DC bit if the complete packet is covered by a CRC calculated using the RMAP CRC definition. This is because the core does not restart the calculation after the header has been received but instead calculates a complete CRC over the packet. Thus any packet format with one CRC at the end of the packet calculated according to RMAP standard can be checked using the DC bit.

CCSDS/CCITT CRC-16 and 16-bit ISO Checksum logic is included in the implementation when the VHDL generct ccsdscrc are set to 1. When a package is received with a protocol ID equal to 0x02 (CCSDS), the core will use the CCSDS CRC/ISO-checksum logic instead of RMAP CRC logic to calculate the CRC. The result is presented by the same status bits as for the RMAP head/data CRC error, but the interpretation of these bits is changed to ISO-checksum/CCSDS CRC error instead.

If the packet is neither of RMAP type nor of the type above with RMAP CRC at the end, then both the HC and DC bits should be ignored.

77.6.9 Error handling

If a packet reception needs to be aborted because of congestion on the network, the suggested solution is to set link disable to '1'. Unfortunately, this will also cause the packet currently being transmitted to be truncated but this is the only safe solution since packet reception is a passive operation depending on the transmitter at the other end. A channel reset bit could be provided but is not a satisfactory solution since the untransmitted characters would still be in the transmitter node. The next character (somewhere in the middle of the packet) would be interpreted as the node address which would probably cause the packet to be discarded but not with 100% certainty. Usually this action is performed when a reception has stuck because of the transmitter not providing more data. The channel reset would not resolve this congestion.

If an AHB error occurs during reception the current packet is spilled up to and including the next EEP/EOP and then the currently active channel is disabled and the receiver enters the idle state. A bit in the channels control/status register is set to indicate this condition.

77.6.10 Promiscuous mode

The core supports a promiscuous mode where all the data received is stored to the first DMA channel enabled regardless of the node address and possible early EOPs/EEPs. This means that all non-EOP/EEP N-Chars received will be stored to the DMA channel. The rxmaxlength register is still checked and packets exceeding this size will be truncated.

RMAP commands will still be handled by the hardware RMAP target when promiscuous mode is enabled, if the RMAP enable bit in the core's Control register is set. If the RMAP enable bit is cleared, RMAP commands will also be stored to a DMA channel.

77.7 Transmitter DMA channels

The transmitter DMA engine handles transmission of data from the DMA channels to the SpaceWire network. Each receive channel has a corresponding transmit channel which means there can be up to 4 transmit channels. It is however only necessary to use a separate transmit channel for each receive channel if there are also separate entities controlling the transmissions. The use of a single channel with multiple controlling entities would cause them to corrupt each other's transmissions. A single channel is more efficient and should be used when possible.

Multiple transmit channels with pending transmissions are arbitrated in a round-robin fashion.

77.7.1 Basic functionality of a channel

A transmit DMA channel reads data from the AHB bus and stores them in the transmitter FIFO for transmission on the SpaceWire network. Transmission is based on the same type of descriptors as for the receiver and the descriptor table has the same alignment and size restrictions. When there are new descriptors enabled the core reads them and transfer the amount data indicated.

77.7.2 Setting up the core for transmission

Four steps need to be performed before transmissions can be done with the core. First the link interface must be enabled and started by writing the appropriate value to the ctrl register. Then the address to the descriptor table needs to be written to the transmitter descriptor table address register and one or more descriptors must also be enabled in the table. Finally, the txen bit in the DMA control register is written with a one which triggers the transmission. These steps will be covered in more detail in the next sections.

77.7.3 Enabling descriptors

The core reads descriptors from an area in memory pointed to by the transmit descriptor table address register. The register consists of a base address and a descriptor selector. The base address points to

the beginning of the area and must start on an address that is aligned to the size of the descriptor table. The size of the descriptor table can be determined from the formula: STS.NTXD*16. The STS.NTXD field shows the number of entries in the descriptor table, and each descriptor size is 16 bytes.

To transmit packets one or more descriptors have to be initialized in memory which is done in the following way: The number of bytes to be transmitted and a pointer to the data has to be set. There are two different length and address fields in the transmit descriptors because there are separate pointers for header and data. If a length field is zero the corresponding part of a packet is skipped and if both are zero no packet is sent. The maximum header length is 255 bytes and the maximum data length is 16 Mbyte - 1. When the pointer and length fields have been set the enable bit should be set to enable the descriptor. This must always be done last. The other control bits must also be set before enabling the descriptor.

The transmit descriptors are 16 bytes in size so the maximum number in a single table is 64. The different fields of the descriptor together with the memory offsets are shown in the tables below.

The HC bit should be set if RMAP CRC should be calculated and inserted for the header field and correspondingly the DC bit should be set for the data field. This field is only used by the core when the CRC logic is available (*rmap* or *rmapcrc* VHDL generic set to 1). The header CRC will be calculated from the data fetched from the header pointer and the data CRC is generated from data fetched from the data pointer. The CRCs are appended after the corresponding fields. The NON-CRC bytes field is set to the number of bytes in the beginning of the header field that should not be included in the CRC calculation.

When the CCSDS/CCITT CRC-16 and 16-bit ISO-checksum logic is available (VHDL generic ccsdscrc set to 1) the core uses the CRC-type filed to determine which CRC logic should be used to calculate the data checksum. When the CCSDS CRC/ISO-checksum logic is not available, the CRC-type field is assumed to be 00 (RMAP CRC).

The CRCs are sent even if the corresponding length is zero, but when both lengths are zero no packet is sent not even an EOP.

77.7.4 Starting transmissions

When the descriptors have been initialized, the transmit enable bit in the DMA control register has to be set to tell the core to start transmitting. New descriptors can be activated in the table on the fly (while transmission is active). Each time a set of descriptors is added the transmit enable bit in the corresponding DMA channel control/status register should be set. This has to be done because each time the core encounters a disabled descriptor this register bit is set to 0.

Table 1313. GRSPW2 transmit descriptor word 0 (address offset 0x0)

31 … 20	19 18	17	16	15	14	13	12	11 … 8	7 … 0
RESERVED	CRC-T	DC	HC	LE	IE	WR	EN	NONCRCLEN	HEADERLEN

31: 20	RESERVED
19: 18	CRC type (CRC-T) - Defines the type of data CRC to use. 00: RMAP CRC 01: CCSDS/CCITT CRC-16 10: 16-bit ISO Checksum (J.G. Fletcher, ISO 8473-1:1998) 11: Reserved
17	Append data CRC (DC) - Append CRC calculated according to the RMAP specification (or CCSDS/CCITT CRC-16 or 16-bit ISO-checksum when available) after the data sent from the data pointer. The CRC covers all the bytes from this pointer. A null CRC will be sent if the length of the data field is zero.
16	Append header CRC (HC) - Append CRC calculated according to the RMAP specification after the data sent from the header pointer. The CRC covers all bytes from this pointer except a number of bytes in the beginning specified by the non-crc bytes field. The CRC will not be sent if the header length field is zero.

Table 1313. GRSPW2 transmit descriptor word 0 (address offset 0x0)

15	Link error (LE) - A Link error occurred during the transmission of this packet.
14	Interrupt enable (IE) - If set, an interrupt will be generated when the packet has been transmitted and the transmitter interrupt enable bit in the DMA control register is set.
13	Wrap (WR) - If set, the descriptor pointer will wrap and the next descriptor read will be the first one in the table (at the base address). Otherwise the pointer is increased with 0x10 to use the descriptor at the next higher memory location.
12	Enable (EN) - Enable transmitter descriptor. When all control fields (address, length, wrap and crc) are set, this bit should be set. While the bit is set the descriptor should not be touched since this might corrupt the transmission. The core clears this bit when the transmission has finished.
11: 8	Non-CRC bytes (NONCRCLEN)- Sets the number of bytes in the beginning of the header which should not be included in the CRC calculation. This is necessary when using path addressing since one or more bytes in the beginning of the packet might be discarded before the packet reaches its destination.
7: 0	Header length (HEADERLEN) - Header Length in bytes. If set to zero, the header is skipped.

Table 1314. GRSPW2 transmit descriptor word 1 (address offset 0x4)

31	0
HEADERADDRESS	

31: 0	Header address (HEADERADDRESS) - Address from where the packet header is fetched. Does not need to be word aligned.

Table 1315. GRSPW2 transmit descriptor word 2 (address offset 0x8)

31 24	23 0
RESERVED	DATALEN

31: 24	RESERVED
23: 0	Data length (DATALEN) - Length in bytes of data part of packet. If set to zero, no data will be sent. If both data- and header-lengths are set to zero no packet will be sent.

Table 1316. GRSPW2 transmit descriptor word 3(address offset 0xC)

31	0
DATAADDRESS	

31: 0	Data address (DATAADDRESS) - Address from where data is read. Does not need to be word aligned.

77.7.5 The transmission process

When the transmitter enable bit in the DMA channel control/status register is set the core starts reading descriptors immediately. The number of bytes indicated are read and transmitted. When a transmission has finished, status will be written to the first field of the descriptor and a packet sent bit is set in the DMA control register. If an interrupt was requested it will also be generated. Then a new descriptor is read and if enabled a new transmission starts, otherwise the transmit enable bit is cleared and nothing will happen until it is enabled again.

77.7.6 The descriptor table address register

The internal pointer which is used to keep the current position in the descriptor table can be read and written through the APB interface. This pointer is set to zero during reset and is incremented each time a descriptor is used. It wraps automatically when the limit for the descriptor table is reached, or it can be set to wrap earlier by setting a bit in the current descriptor.

The descriptor table register can be updated with a new table anytime when no transmission is active. No transmission is active if the transmit enable bit is zero and the complete table has been sent or if

the table is aborted (explained below). If the table is aborted one has to wait until the transmit enable bit is zero before updating the table pointer.

77.7.7 Error handling

Abort Tx

The DMA control register contains a bit called Abort TX which if set causes the current transmission to be aborted, the packet is truncated and an EEP is inserted. This is only useful if the packet needs to be aborted because of congestion on the SpaceWire network. If the congestion is on the AHB bus this will not help (This should not be a problem since AHB slaves should have a maximum of 16 wait-states). The aborted packet will have its LE bit set in the descriptor. The transmit enable register bit is also cleared and no new transmissions will be done until the transmitter is enabled again.

AHB error

When an AHB error is encountered during transmission the currently active DMA channel is disabled and the transmitter goes to the idle mode. A bit in the DMA channel's control/status register is set to indicate this error condition and, if enabled, an interrupt will also be generated. Further error handling depends on what state the transmitter DMA engine was in when the AHB error occurred. If the descriptor was being read the packet transmission had not been started yet and no more actions need to be taken.

If the AHB error occurs during packet transmission the packet is truncated and an EEP is inserted. Lastly, if it occurs when status is written to the descriptor the packet has been successfully transmitted but the descriptor is not written and will continue to be enabled (this also means that no error bits are set in the descriptor for AHB errors).

The client using the channel has to correct the AHB error condition and enable the channel again. No more AHB transfers are done again from the same unit (receiver or transmitter) which was active during the AHB error until the error state is cleared and the unit is enabled again.

Link error

When a link error occurs during the transmission, the remaining part of the packet is discarded up to, and including, the next EOP/EEP. When this is done, status is immediately written back to the descriptor (with the LE bit set) and the descriptor pointer is incremented. The link will be disconnected when the link error occurs but the core will automatically try to connect again, provided that the link-start bit (LS bit in Control register) is asserted, and the link-disabled bit (LD bit in Control register) is deasserted. If the LE bit in the DMA channel's control register is not set the transmitter DMA engine will wait for the link to enter run-state, and start a new transmission immediately when possible, assuming there are packets pending. If the LE bit in the DMA channel's control register is set, the transmitter will be disabled when a link error occurs during a transmission of a packet. In that case, no more packets will be transmitted until the transmitter is enabled again. See description of the DMA channel's control register for more details.

77.8 RMAP

The Remote Memory Access Protocol (RMAP) is used to implement access to resources in the node via the SpaceWire link. Some common operations are reading and writing to memory, registers and FIFOs. The core has an optional hardware RMAP target. Whether or not the RMAP target is implemented is indicated by the CTRL.RA bit. This section describes the basics of the RMAP protocol and the target implementation.

77.8.1 Fundamentals of the protocol

RMAP is a protocol which is designed to provide remote access via a SpaceWire network to memory mapped resources on a SpaceWire node. It has been assigned protocol ID 0x01. It provides three operations: write, read and read-modify-write. These operations are posted operations, which means that a

source does not wait for an acknowledge or reply. It also implies that any number of operations can be outstanding at any time and that no timeout mechanism is implemented in the protocol. Time-outs must instead be implemented in the user application which sends the commands. Data payloads of up to 2^{24} - 1 bytes is supported by the protocol. A destination can be requested to send replies and to verify data before executing an operation. For a complete description of the protocol, see the RMAP standard (ECSS-E-ST-50-52C).

77.8.2 Implementation

The core includes a target for RMAP commands which processes all incoming packets with protocol ID = 0x01, type field (bit 7 and 6 of the 3rd byte in the packet) equal to 01b and an address falling in the range set by the default address and mask register. When such a packet is detected it is not stored to the DMA channel, instead it is passed to the RMAP receiver.

The core implements all three commands defined in the standard with some restrictions. As explained in 77.2.3, the representation of the data bytes in memory does not change with the endianness of the system. This means that the first byte received is the msb in a word. The target will not receive RMAP packets using the extended protocol ID which are always dumped to the DMA channel.

The RMAP receiver processes commands. If they are correct and accepted the operation is performed on the AHB bus and a reply is formatted. If an acknowledge is requested the RMAP transmitter automatically send the reply. RMAP transmissions have priority over DMA channel transmissions.

There is a user accessible destination key register which is compared to destination key field in incoming packets. If there is a mismatch and a reply has been requested the error code in the reply is set to 3. Replies are sent if and only if the ack field is set to ‘1’.

When a failure occurs during a bus access the error code is set to 1 (General Error). There is predetermined order in which error-codes are set in the case of multiple errors in the core. It is shown in table 1317.

Table 1317. The order of error detection in case of multiple errors. The error detected first has number 1.

Detection Order	Error Code	Error
1	12	Invalid destination logical address
2	2	Unused RMAP packet type or command code
3	3	Invalid destination key
4	9	Verify buffer overrun
5	11	RMW data length error
6	10	Authorization failure
7*	1	General Error (AHB errors during non-verified writes)
8	5/7	Early EOP / EEP (if early)
9	4	Invalid Data CRC
10	1	General Error (AHB errors during verified writes or RMW)
11	7	EEP
12	6	Cargo Too Large
*The AHB error is not guaranteed to be detected before Early EOP/EEP or Invalid Data CRC. For very long accesses the AHB error detection might be delayed causing the other two errors to appear first.		

Read accesses are performed on the fly, that is they are not stored in a temporary buffer before transmitting. This means that the error code 1 will never be seen in a read reply since the header has already been sent when the data is read. If the AHB error occurs the packet will be truncated and ended with an EEP.

Errors up to and including Invalid Data CRC (number 9) are checked before verified commands. The other errors do not prevent verified operations from being performed.

The details of the support for the different commands are now presented. All defined commands which are received but have an option set which is not supported in this specific implementation will not be executed and a possible reply is sent with error code 10.

77.8.3 Write commands

The write commands are divided into two subcategories when examining their capabilities: verified writes and non-verified writes. Verified writes have a length restriction of 4 bytes and the address must be aligned to the size. That is 1 byte writes can be done to any address, 2 bytes must be halfword aligned, 3 bytes are not allowed and 4 bytes writes must be word aligned. Since there will always be only one AHB operation performed for each RMAP verified write command the incrementing address bit can be set to any value.

Non-verified writes have no restrictions when the incrementing bit is set to 1. If it is set to 0 the number of bytes must be a multiple of 4 and the address word aligned. There is no guarantee how many words will be written when early EOP/EEP is detected for non-verified writes.

77.8.4 Read commands

Read commands are performed on the fly when the reply is sent. Thus if an AHB error occurs the packet will be truncated and ended with an EEP. There are no restrictions for incrementing reads but non-incrementing reads have the same alignment restrictions as non-verified writes. Note that the “Authorization failure” error code will be sent in the reply if a violation was detected even if the length field was zero. Also note that no data is sent in the reply if an error was detected i.e. if the status field is non-zero.

77.8.5 RMW commands

All read-modify-write sizes are supported except 6 which would have caused 3 B being read and written on the bus. The RMW bus accesses have the same restrictions as the verified writes. As in the verified write case, the incrementing bit can be set to any value since only one AHB bus operation will be performed for each RMW command. Cargo too large is detected after the bus accesses so this error will not prevent the operation from being performed. No data is sent in a reply if an error is detected i.e. the status field is non-zero.

77.8.6 Control

The RMAP target mostly runs in the background without any external intervention, but there are a few control possibilities.

There is an enable bit in the control register of the core which can be used to completely disable the RMAP target. When it is set to ‘0’ no RMAP packets will be handled in hardware, instead they are all stored to the DMA channel.

There is a possibility that RMAP commands will not be performed in the order they arrive. This can happen if a read arrives before one or more writes. Since the target stores replies in a buffer with more than one entry several commands can be processed even if no replies are sent. Data for read replies is read when the reply is sent and thus writes coming after the read might have been performed already if there was congestion in the transmitter. To avoid this the RMAP buffer disable bit can be set to force the target to only use one buffer which prevents this situation.

The last control option for the target is the possibility to set the destination key which is found in a separate register.

*Table 1318.*GRSPW2 hardware RMAP handling of different packet type and command fields.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Command	Action
Reserved	Command / Response	Write / Read	Verify data before write	Acknow-ledge	Increment Address		
0	0	-	-	-	-	Response	Stored to DMA-channel.
0	1	0	0	0	0	Not used	Does nothing. No reply is sent.
0	1	0	0	0	1	Not used	Does nothing. No reply is sent.
0	1	0	0	1	0	Read single address	Executed normally. Address has to be word aligned and data size a multiple of four. Reply is sent. If alignment restrictions are violated error code is set to 10.
0	1	0	0	1	1	Read incre-menting address.	Executed normally. No restric-tions. Reply is sent.
0	1	0	1	0	0	Not used	Does nothing. No reply is sent.
0	1	0	1	0	1	Not used	Does nothing. No reply is sent.
0	1	0	1	1	0	Not used	Does nothing. Reply is sent with error code 2.
0	1	0	1	1	1	Read-Mod-ify-Write incrament-ing address	Executed normally. If length is not one of the allowed rmw val-ues nothing is done and error code is set to 11. If the length was correct, alignment restric-tions are checked next. 1 byte can be rmw to any address. 2 bytes must be halfword aligned. 3 bytes are not allowed. 4 bytes must be word aligned. If these restrictions are violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	0	0	0	Write, sin-gle-address, do not verify before writ-ing, no acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done. No reply is sent.
0	1	1	0	0	1	Write, incre-menting address, do not verify before writ-ing, no acknowledge	Executed normally. No restric-tions. No reply is sent.
0	1	1	0	1	0	Write, sin-gle-address, do not verify before writ-ing, send acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.

*Table 1318.*GRSPW2 hardware RMAP handling of different packet type and command fields.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Command	Action
Reserved	Command / Response	Write / Read	Verify data before write	Acknow-ledge	Increment Address		
0	1	1	0	1	1	Write, incre-menting address, do not verify before writ-ing, send acknowledge	Executed normally. No restric-tions. If AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	0	0	Write, single address, ver-ify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restric-tions apply as for rmw. No reply is sent.
0	1	1	1	0	1	Write, incre-menting address, ver-ify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restric-tions apply as for rmw. If they are violated nothing is done. No reply is sent.
0	1	1	1	1	0	Write, single address, ver-ify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are vio-lated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	1	1	Write, incre-menting address, ver-ify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are vio-lated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
1	0	-	-	-	-	Unused	Stored to DMA-channel.
1	1	-	-	-	-	Unused	Stored to DMA-channel.

77.9 AMBA interface

The AMBA interface consists of an APB interface, an AHB master interface and DMA FIFOs. The APB interface provides access to the user registers which are described in section 77.13. The DMA engines have 32-bit wide FIFOs to the AHB master interface which are used when reading and writing to the bus.

The transmitter DMA engine reads data from the bus in bursts which are half the FIFO size in length. A burst is always started when the FIFO is half-empty or if it can hold the last data for the packet. The burst containing the last data might have shorter length if the packet is not an even number of bursts in size.

The receiver DMA works in the same way except that it checks if the FIFO is half-full and then performs a burst write to the bus which is half the fifo size in length. The last burst might be shorter. If the *rmap* or *rxunaligned* VHDL generics are set to 1 the interface also handles byte accesses. Byte accesses are used for non word-aligned buffers and/or packet lengths that are not a multiple of four bytes. There might be 1 to 3 single byte writes when writing the beginning and end of the received packets.

77.9.1 APB slave interface

As mentioned above, the APB interface provides access to the user registers which are 32-bits in width. The accesses to this interface are required to be aligned word accesses. The result is undefined if this restriction is violated.

77.9.2 AHB master interface

The core contains a single master interface which is used by both the transmitter and receiver DMA engines. The arbitration algorithm between the channels is done so that if the current owner requests the interface again it will always acquire it. This will not lead to starvation problems since the DMA engines always deassert their requests between accesses.

The AHB accesses are always word accesses (HSIZE = 0x010) of type incremental burst with unspecified length (HBURST = 0x001) if the RMAP target is not implemented (RA bit in Control register = 0) and unaligned transfers are not supported (RX bit in the Control register = 0). If either the RMAP target is implemented, or unaligned transfers are supported, then AHB accesses can be of size byte, halfword and word (HSIZE = 0x000, 0x001, 0x010) otherwise. Byte and halfword accesses are always NONSEQ. Note that read accesses are always word accesses (HSIZE = 0x010), which can result in destructive read.

The burst length will be half the AHB FIFO size except for the last transfer for a packet which might be smaller. Shorter accesses are also done during descriptor reads and status writes.

The AHB master also supports non-incrementing accesses where the address will be constant for several consecutive accesses. HTRANS will always be NONSEQ in this case while for incrementing accesses it is set to SEQ after the first access. This feature is included to support non-incrementing reads and writes for RMAP.

If the core does not need the bus after a burst has finished there will be one wasted cycle (HTRANS = IDLE).

BUSY transfer types are never requested and the core provides full support for ERROR, RETRY and SPLIT responses.

77.10 SpaceWire Plug-and-Play

The core supports parts of the SpaceWire Plug-and-Play protocol. The supported fields are listed in table 1321, and explained in more detail in tables 1322 through 1336. Table 1321 also shows which type of SpaceWire Plug-and-Play access type (read, write, compare-and-swap) that is allowed for the field. Note that two different amount of SpaceWire Plug-and-Play support may be included. Either only device identification through the Device Information fields is supported, or device configuration through the SpaceWire Protocol fields is supported as well. The amount of support is indicated by the CTRL.PNPA field. Note also that the CTRL.PE must be set in order to enable the SpaceWire Plug-and-Play support.

The SpaceWire Plug-and-Play protocol uses standard RMAP commands and replies with the same requirements as presented in section 77.8, but with the following differences:

- Protocol Identifier field of a command shall be set to 0x03.
- A command's address fields shall contain a word address. The SpaceWire Plug-and-Play addresses are encoded as shown in table 1319.

- The increment bit in the command's instruction field shall be set to 1, otherwise a reply with Status field set to 0x0A (authorization failure) is sent.
- RMAP read-modify-write command is replaced by a compare-and-swap command. The command's data fields shall contain the new data to be written, while the mask fields shall contain the value that the current data must match in order for the new data to be written. If there is a mismatch, a reply with Status field set to 0x0A (authorization failure) is sent.
- The reply packet's Status field can contain the additional status codes described in table 1320.

Table 1319. SpaceWire Plug-and-Play address encoding

31 24	23 19	18 14	13 0
Application Index	Protocol Index	FieldSet ID	Field ID

Table 1320. SpaceWire Plug-and-Play status codes

Value	Description
0xF0	Unauthorized access - A write, or compare-and-swap command, with an address other then the Device ID field's address, arrived when the core was not configured (Device ID field = 0), or the command did not match the owner information saved in the Link Information field and Owner Address fields.
0xF1	Reserved field set - A read, write, or compare-and-swap command's address field points to a non existing field set. [1)]
0xF2	Read-only field - A write, or compare-and-swap command's address points to a read-only field.
0xF3	Compare-and-swap-only-field - A write command's address points to a field that is only writable through a compare-and-swap-only.
Note 1: An access to a non existing field, within a existing field set, does not generate an error response. The data returned in a read access is zero, while a write access has no effect.	

Table 1321. SpaceWire Plug-and-Play support

SpW PnP Address	Name	Acronym	Service - Field set - Field	Access type
0x00000000	SpaceWire Plug-and-Play - Device Vendor and Product ID	PNPVEND	Device Information - Device Identification - Device Vendor and Product ID	read
0x00000001	SpaceWire Plug-and-Play - Version	PNPVER	Device Information - Device Identification - Version	read
0x00000002	SpaceWire Plug-and-Play - Device Status	PNPDEVSTS	Device Information - Device Identification - Device Status	read
0x00000003	SpaceWire Plug-and-Play - Active Links	PNPALINK	Device Information - Device Identification - Active Links	read
0x00000004	SpaceWire Plug-and-Play - Link Information	PNPLINFO	Device Information - Device Identification - Link Information	read
0x00000005	SpaceWire Plug-and-Play - Owner Address 0	PNPOA0	Device Information - Device Identification - Owner Address 0	read
0x00000006	SpaceWire Plug-and-Play - Owner Address 1	PNPOA1	Device Information - Device Identification - Owner Address 1	read
0x00000007	SpaceWire Plug-and-Play - Owner Address 2	PNPOA2	Device Information - Device Identification - Owner Address 2	read
0x00000008	SpaceWire Plug-and-Play - Device ID	PNPDEVID	Device Information - Device Identification - Device ID	read, cas

Table 1321. SpaceWire Plug-and-Play support

SpW PnP Address	Name	Acronym	Service - Field set - Field	Access type
0x00000009	SpaceWire Plug-and-Play - Unit Vendor and Product ID	PNPUVEND	Device Information - Device Identification - Unit Vendor and Product ID	read
0x0000000A	SpaceWire Plug-and-Play - Unit Serial Number	PNPUSN	Device Information - Device Identification - Unit Serial Number	read
0x00004000	SpaceWire Plug-and-Play - Vendor String Length	PNPVSTRL	Device Information - Vendor / Product String - Vendor String Length	read
0x00006000	SpaceWire Plug-and-Play - Product String Length	PNPPSTRL	Device Information - Vendor / Product String - Product String Length	read
0x00008000	SpaceWire Plug-and-Play - Protocol Count	PNPPCNT	Device Information - Protocol Support - Protocol Count	read
0x00008001	SpaceWire Plug-and-Play - Protocol Identification 1	PNPPID1	Device Information - Protocol Support - Protocol Identification 1	read
0x00008002	SpaceWire Plug-and-Play - Protocol Identification 2	PNPPID2	Device Information - Protocol Support - Protocol Identification 2	read
0x0000C000	SpaceWire Plug-and-Play - Application Count	PNPACNT	Device Information - Application Support- Application Count	read
0x00080000	SpaceWire Plug-and-Play - Time-Code Counter [1)]	PNPTCC	SpaceWire Protocol - Device Configuration - Time-Code Counter	read, write, cas
0x00084008	SpaceWire Plug-and-Play - Link Status [1)]	PNPLSTS	SpaceWire Protocol - Link Configuration - Link Status	read, write, cas
0x00084009	SpaceWire Plug-and-Play - Link Control [1)]	PNPLCTRL	SpaceWire Protocol - Link Configuration - Link Control	read, write, cas
0x00100000	SpaceWire Plug-and-Play - Maximum Write Length [1)]	PNPMWLEN	SpaceWire PnP Protocol - Protocol Information - Maximum Write Length	read
0x00100001	SpaceWire Plug-and-Play - Maximum Read Length [1)]	PNPMRLEN	SpaceWire PnP Protocol - Protocol Information - Maximum Read Length	read
Note 1: Register is only available when device configuration through SpaceWire Plug-and-Play is supported, which is indicated by the value of the CTRL.PNPA field.				

The layout of the SpaceWire Plug-and-Play registers used in this section is the same as for the registers described in section 77.13, and is exemplified in table 1346. The reset value field

and bit-field type definitions are also the same as in section 77.13, and are explained in tables 1347 and 1348 respectively.

*Table 1322.*0x00000000 - PNPVEND - SpaceWire Plug-and-Play - Device Vendor and Product ID

31 16	15 0
VEND	PROD
*	*
r	r

31: 16 Vendor ID (VEND) - SpaceWire vendor ID assigned at implementation time. Value taken from the VHDL generic *pnpvendid*.

15: 0 Product ID (PROD) - Product ID assigned at implementation time. Value taken from the VHDL generic *pnpprodid*.

*Table 1323.*0x00000001 - PNPVER - SpaceWire Plug-and-Play - Version

31 24	23 16	15 8	7 0
MAJOR	MINOR	PATCH	RESERVED
*	*	*	*
r	r	r	r

31: 24 Major version number (MAJOR) - Major version number set at implementation time. Value taken from the VHDL generic *pnpmajorid*.

23: 16 Minor version number (MINOR) - Minor version number set at implementation time. Value taken from the VHDL generic *pnpminorid*.

15: 8 Patch / Build number (PATCH) - Patch / Build number set at implementation time. Value taken from the VHDL generic *pnppatchid*.

7: 0 RESERVED

*Table 1324.*0x00000002 - PNPDEVSTS - SpaceWire Plug-and-Play - Device Status

31 8	7 0
RESERVED	STATUS
0x000000	0x00
r	r

31: 8 RESERVED

7: 0 Device status (STATUS) - Constant value of 0x00.

*Table 1325.*0x00000003 - PNPALINK - SpaceWire Plug-and-Play - Active Links

31 2	1	0
RESERVED	AC	R
0x00000000	0	0
r	r	r

31: 20 RESERVED

19: 1 Link active (AC) - Indicates if the link interface is in run-state. 0 = Not run-state, 1 = run-state.

0 RESERVED

Table 1326. 0x00000004 - PNPLINFO -SpaceWire Plug-and-Play - Link Information

31 24	23 22	21	20 16	15 13	12 8	7	6	5	4 0
OLA	OAL	R	OL	RES	RL	T	U	R	LC
0x00	0x0	0	0x0	0x0	0x0	1	0	0	0x13
r	r	r	r	r	r	r	r	r	r

31: 24 Owner logical address (OLA) - Shows the value of the Initiator Logical Address field from the last successful compare-and-swap command that set the Device ID field.

23: 22 Owner address length (OAL) - Shows how many of the three Owner Address fields that contain valid data.

21 RESERVED

20: 16 Owner link (OL) - Shows the number of the port which was used for the last successful operation to set the value of the Device ID field.

15: 13 RESERVED

12: 8 Return link (RL) - Shows the number of the port through which the reply to the current read command will be transmitted.

7 Device type (T) - Constant value of 0, indicating a node.

6 Unit information (U) - Indicates if the unit identification information (Unit Vendor and Product ID field, and Unit Serial Number field) are valid. 0 = invalid, 1 = valid. This bit will be 0 after reset / power-up. Once the Unit Vendor and Product ID field has been written with a non-zero value, this bit will be set to 1.

5 RESERVED

4: 0 Link count (LC) - Shows the number of router ports. Constant value of 0x13.

Table 1327. 0x00000005 - PNPOA0 - SpaceWire Plug-and-Play - Owner Address 0

31 0
RA
0x00000000
r

31: 0 Reply address (RA) - Shows byte 0-3 of the Reply Address from the last successful compare-and-swap command that set to the Device ID field. If there was no Reply Address, then this field is zero.

Table 1328. 0x00000006 - PNPOA1 - SpaceWire Plug-and-Play - Owner Address 1

31 0
RA
0x00000000
r

31: 0 Reply address (RA) - Shows byte 4-7 of the Reply Address from the last successful compare-and-swap command that set to the Device ID field. If the Reply Address was four bytes or less, then this field is zero.

Table 1329. 0x00000007 - PNPOA2 - SpaceWire Plug-and-Play - Owner Address 2

31 0
RA
0x00000000
r

31: 0 Reply address (RA) - Shows byte 8-11 of the Reply Address from the last successful compare-and-swap command that set to the Device ID field. If the Reply Address was eight bytes or less, then this field is zero.

Table 1330. 0x00000008 - PNPDEVID - SpaceWire Plug-and-Play - Device ID

31	0
DID	
0x00000000	
cas	

31: 0 Device ID (DID) - Shows the device identifier. This field is set to zero after reset / power-up, and when the link-interface is not in run-state.

Table 1331. 0x00000009 - PNPUVEND - SpaceWire Plug-and-Play - Unit Vendor and Product ID

31 16	15 0
VEND	PROD
*	*
r	r

31: 16 Unit vendor ID (VEND) - Shows the unit vendor identifier. This field is read-only through the SpaceWire Plug-and-Play protocol, however it is writable through an APB register (see section 77.13.32). Reset value is taken from the input signal PNPUVENDID. Whenever this field, or the PROD field, is set to a non-zero value, the PNPLINFO.U bit is set to 1.

15: 0 Unit product ID (PROD) - Shows the unit product identifier. This field is read-only through the SpaceWire Plug-and-Play protocol, however it is writable through an APB register (see section 77.13.32). Reset value is taken from the input signal PNPUPRODID. Whenever this field, or the VEND field, is set to a non-zero value, the PNPLINFO.U bit is set to 1.

Table 1332. 0x0000000A - PNPUSN - SpaceWire Plug-and-Play - Unit Serial Number

31	0
USN	
*	
r	

31: 0 Unit serial number (USN) - Shows the unit serial number. This field is read-only through the SpaceWire Plug-and-Play protocol, however it is writable through the APB register (see section 77.13.33). Reset value is taken from the input signal PNPUSN.

Table 1333. 0x00004000 - PNPVSTRL - SpaceWire Plug-and-Play - Vendor String Length

31 15	14 0
RESERVED	LEN
0x00000	0x0000
r	r

31: 15 RESERVED

14: 0 Vendor string length (LEN) - Constant value of 0, indicating that no vendor string is present.

Table 1334. 0x00006000 - PNPPSTRL - SpaceWire Plug-and-Play - Product String Length

31 15	14 0
RESERVED	LEN
0x00000	0x0000
r	r

31: 15 RESERVED

14: 0 Product string length (LEN) - Constant value of 0, indicating that no product string is present.

Table 1335. 0x00008000 - PNPPCNT - SpaceWire Plug-and-Play - Protocol Count

31 5	4 0
RESERVED	PC
0x0000000	*
r	r

31: 5 RESERVED

4: 0 Protocol count (PC) - Constant value of 0 when only device identification is supported (CTRL.PNPA = 1). Constant value of 2 when device configuration is supported (CTRL.PNPA = 2).

Table 1336. 0x0000C000 - PNPACNT - SpaceWire Plug-and-Play - Application Count

31 8	7 0
RESERVED	AC
0x000000	0x00
r	r

31: 8 RESERVED

7: 0 Application count (AC) - Constant value of 0, indicating that no applications can be managed by using SpaceWire Plug-and-Play.

Table 1337. 0x00080000 - PNPTCC - SpaceWire Plug-and-Play - Time-Code Counter

31 6	5 0
RESERVED	TC
0x0000000	0x00
r	rw*

31: 6 RESERVED

5: 0 Time Count (TC) - Current time value. This bitfield can be reset by writing zero to it. Writing any other value has no effect. Double map of TC.TIMECNT value (see TC register in section 77.13.6 for a functional description).

Table 1338. 0x00084008 - PNPLSTS - SpaceWire Plug-and-Play - Link Status

31	30	29 19	18 16	15 8	7	6	5	4	3	2	1	0
ND	LT	RESERVED	LS	RESERVED	R	CE	ER	PE	DE	R	IA	R
1	1	0x000	0x0	0x00	0	0	0	0	0	0	0	0
r	r	r	r	r	r	rw*	rw*	rw*	rw*	r	rw*	r

31 Network discovery (ND) - Constant value of 1, indicating that the link can be used for network discovery.

30 Link type (LT) - Constant value of 1, indicating that the link is a SpaceWire link.

29: 19 RESERVED

18: 16 Link State (LS) - The current state of the link interface. 0 = Error-reset, 1 = Error-wait, 2 = Ready, 3 = Started, 4 = Connecting, 5 = Run.

25: 8 RESERVED

7 RESERVED

6 Credit Error (CE) - A credit has occurred. Cleared when complete PNPLSTS is written with zero.

5 Escape Error (ER) - An escape error has occurred. Cleared when complete PNPLSTS is written with zero.

4 Parity Error (PE) - A parity error has occurred. Cleared when complete PNPLSTS is written with zero.

3 Disconnect Error (DE) - A disconnection error has occurred. Cleared when complete PNPLSTS is written with zero.

2 RESERVED

1 Invalid Address (IA) - Set to one when a packet is received with an invalid destination address field, i.e it does not match the DEFADDR register. Cleared when complete PNPLSTS is written with zero.

0 RESERVED

*Table 1339.*0x00084009 - PNPLCTRL - SpaceWire Plug-and-Play - Link Control

31 5	4	3	2	1	0
RESERVED	TT	R	LD	LS	AS
	0	0	0	0	*
	rw	r	rw	rw	rw

31: 5 RESERVED

4 Time-Code transmission (TT) - Enable Time-Code transmission.

3 RESERVED

2 Link Disable (LD) - Disable the SpaceWire codec link-interface.

1 Link Start (LS) - Start the link, i.e. allow a transition from ready-state to started-state.

0 Autostart (AS) - Automatically start the link when a NULL has been received. Reset value is set from input signal RMAPEN if RMAP target is available (CTRL.RA bit = 1), otherwise the reset value is '0'.

*Table 1340.*0x00100000 - PNPMWLEN - SpaceWire Plug-and-Play - Maximum Write Length

31 15	14 0
RESERVED	LEN
0x00000	0x0002
r	r

31: 15 RESERVED

14: 0 Length (LEN) - Constant value, indicating the maximum number of fields that can be written with a single write command.

*Table 1341.*0x00100001 - PNPMRLEN - SpaceWire Plug-and-Play - Maximum Read Length

31 15	14 0
RESERVED	LEN
0x00000	0x4000
r	r

31: 15 RESERVED

14: 0 Length (LEN) - Constant value, indicating the maximum number of fields that can be read with a single read command.

77.11 Implementation

77.11.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

There are five input resets, in order to cover every clock domain: AMBA reset (rst), transmitter synchronous reset (txsyncrst), receiver synchronous reset for port 0 (rxsyncrst0), receiver synchronous reset for port 1 (rxsyncrst1) and a receiver asynchronous reset shared by both ports (rxasyncrst).

Additionally, the core outputs two internal resets: the register reset, swno.ctrlregrst (bit 6 of the Control Register. See 77.13.1), and the internal reset for synchronization between transmitter and receivers, swno.rxrst.

The core does not implement any kind of internal reset generation or synchronization, the input resets are completely independent. The reset generation shall be done in a higher instance, taking into account the clock domains and also the output resets of the core. A description of how the resets shall be combined and generated can be found below. Frontgrade Gaisler recommends following these guidelines unless indicated otherwise.

- The AMBA reset is the combination of the external reset and the output register reset negated (swno.ctrlregrst), as it is active high. Then, the reset is synchronized with the AMBA clock by using a reset generator. Its output is the AMBA reset, rst.
- The transmitter reset is the combination of the external reset and the output register reset negated (swno.ctrlregrst), as it is active high. Then, the reset is synchronized with the transmitter clock by using a reset generator. Its output is the transmitter reset, txsyncrst.
- The asynchronous reset for both receiver channels is simply the output synchronization reset, swno.rxrst, connected directly to the input rxasyncrst.
- The synchronous reset for each receiver channel is the output receiver reset, swno.rxrst, synchronized with the appropriate clock domain by using a reset generator. Its output is the synchronous reset for the specific receiver port, rxsyncrst0 or rxsyncrst1.

In order to avoid the process of generating all these resets, GRLIB includes a wrapper called grspwm which instantiates either GRSPW or GRSPW2. If the latter is chosen, there is a generic to directly implement the previous reset generators in the wrapper itself, so the top entity does not need to generate the resets anymore. Further information about the proper way of combining the reset signals or using the wrapper can be found in 77.20, including an example for both possibilities.

Note that the IP may generate local asynchronous reset nets, depending on the configuration of two generics:

- If the receiver uses SDR sampling (input_type = 2) or DDR sampling (input_type = 3 or 4) and the same clock is used for TX and RX (rxtx_sameclk = 1), then the IP will not contain internal asynchronous resets.
- In any other case, i.e. different clock for TX and RX or any other receiver type (XOR, external CCC), the IP will contain internal asynchronous resets.

77.11.2 Clock-generation

The receiver module found in figure 195 should be clocked with the RXCLKO output from the GRSPW2_PHY module. See the example instantiation in this section and the GRSPW2_PHY section of the GRLIB IP Core User's Manual for more information on how to connect this clock.

The transmitter clock is generated from the TXCLK input. A separate clock input is used to allow the transmitter to be run at much higher frequencies than the system clock. The SpaceWire node contains a clock-divider which divides the TXCLK signal to the wanted frequency. The transmitter clock should be 10 MHz during initialization and any frequency above 2 MHz in the run-state.

There is an input signal called CLKDIV10 which sets the reset values for the user accessible clock divisor registers. There is one register value which is used during initialisation and one which is used in run-state. See 77.3.6 for details on how to set the clock divisor values.

Since only integer values are allowed for the clock division and the required init-frequency is 10 MHz the frequency of the TXCLK input must be a multiple of 10 MHz. The clock divisor value is 8-bits wide so the maximum TXCLK frequency supported is 2.56 GHz (note that there is also a restriction on the relation between the system and transmit clock frequencies).

77.11.3 Timers

There are two timers in the core: one for generating the 6.4/12.8 µs periods and one for disconnect timing.

The timeout periods are generated from the TX clock whose frequency must be at least 10 MHz to guarantee disconnect timing limits. The same clock divisor is used as for the TX clock during initialisation so it must be set correctly for the link timing to work.

77.11.4 Synchronization

The transmitter and receiver bit rates can be eight times higher than the system clock frequency. This includes a large margin for clock skew and jitter so it might be possible to run at even higher rate differences. Note also that the receiver clocks data at both negative and positive edges for the input modes 0 and 1 so the bitrate is twice the clock frequency. There is no direct relationship between bitrate and frequency for the sampling modes.

The clock synchronization is just one limiting factor for the clock frequency, it might for example not be possible to achieve the highest possible frequency for certain technologies.

The asynchronous reset to the receiver clock domain has to have a maximum delay of one receiver clock cycle to ensure correct operation. This is needed because the receiver uses a completely asynchronous reset. To make sure that nothing bad happens there is a synchronous reset guard which prevents any signals from being assigned before all registers have their reset signals released.

In the sampling modes this asynchronous reset can be removed if both the receiver and transmitter runs on the same clock. The core is configured to use the same receiver and transmitter clock by setting the *rxtx_sameclk* generic to 1.

77.11.5 Fault-tolerance

The core can optionally be implemented with fault-tolerance against SEU errors in the FIFO memories. The fault-tolerance is enabled through the *ft* VHDL generic. Possible options are byte parity protection (*ft = 1*) or TMR registers (*ft = 2*).

77.11.6 Synthesis

The fact there are three clock domains in the core of which all are possibly high frequency clocks makes it necessary to declare all paths between the clock domains as false paths. In Synplify this is most easily done by declaring all the clocks to be in different clockgroups in the sdc file (if Synplify does not automatically put them in different groups). This will disable any timing considerations between the clock domains and these constraints will also propagate to the place and route tool.

77.11.7 Technology mapping

The core has three generics for technology mapping: *tech*, *techfifo* and *memtech*. *Tech* selects the technology used for the clock buffers and also adds reset to some registers for technologies where they would otherwise cause problems with gate-level simulations. *Techfifo* selects whether *memtech* should be used to select the technology for the FIFO memories (the RMAP buffer is not affected by the this generic) or if they should be inferred. *Tech* and *memtech* can be set to any value from 0 to NTECH as defined in the GRLIB.TECH package.

77.11.8 RAM usage

The core maps all RAM memories on the syncram_2p component if the *ft* generic is 0 and to the syncram_2pft component for other values. The syncrams are located in the technology mapping library (TECHMAP). The organization of the different memories are described below. If techfifo and/or memtech is set to 0 the synthesis tool will infer the memories. Either RAM blocks or flip-flops will be used depending on the tool and technology. The number of flip-flops used is *syncram depth x syncram width* for all the different memories. The receiver AHB FIFO with fifosize 32 will for example use 1024 flips-flops.

Receiver AHB FIFO

The receiver AHB FIFO consists of one syncram_2p block with a width of 32-bits. The depth is determined by the configured FIFO depth. Table 1342 shows the syncram organization for the allowed configurations.

*Table 1342.*syncram_2p sizes for GRSPW2 receiver AHB FIFO.

Fifosize	Syncram_2p organization
4	4x32
8	8x32
16	16x32
32	32x32
64	64x32

Transmitter AHBFIFO

The transmitter AHB FIFO consists of one syncram_2p block with a width of 32-bits. The depth is determined by the configured FIFO depth. Table 1343 shows the syncram organization for the allowed configurations.

*Table 1343.*syncram_2p sizes for transmitter AHB FIFO.

Fifosize	Syncram_2p organization
4	4x32
8	8x32
16	16x32
32	32x32
64	64x32

Receiver N-Char FIFO

The receiver N-Char fifo consists of one syncram_2p block with a width of 10-bits. The depth is determined by the configured FIFO depth. Table 1344 shows the syncram organization for the allowed configurations.

*Table 1344.*syncram_2p sizes for the receiver N-Char FIFO.

Fifosize	Syncram_2p organization
16	16x10
32	32x10
64	64x10

RMAP buffer

The RMAP buffer consists of one syncram_2p block with a width of 8-bits. The depth is determined by the number of configured RMAP buffers. Table 1345 shows the syncram organization for the allowed configurations.

*Table 1345.*syncram_2p sizes for RMAP buffer memory.

RMAP buffers	Syncram_2p organization
2	64x8
4	128x8
8	256x8

77.11.9 Endianness

The core automatically changes its endianness behavior depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for both big-endian and little-endian systems.

77.12 AXI support

The core is designed for an AMBA system but can be adapted for AXI using the AHBM2AXI adapter.

77.13 Registers

The core is programmed through registers mapped into APB address space. The registers are listed in table, 1349 and described in detail in the subsequent tables. Addresses not listed in table 1349 are reserved. A read access to a reserved register, or reserved field with a register, will always return zero, and a write access has no effect. The register layout used is exemplified in table 1346, and the values used in the reset value row and field type row are explained in tables 1347 and 1348.

Table 1346.<APB ddress offset> - <Register acronym> - <Register name>

31 24	23 16	15 8	7 0
EF3	EF2	EF1	EF0
<Reset value for EF3>	<Reset value for EF2>	<Reset value for EF1>	<Reset value for EF0>
<Bit-field type for EF3>	<Bit-field type for EF2>	<Bit-field type for EF1>	<Bit-field type for EF0>

31: 24 Example bit-field 3 (EF3) - <Bit-field description>
23: 16 Example bit-field 2 (EF2) - <Bit-field description>
15: 8 Example bit-field 1 (EF1) - <Bit-field description>
7: 0 Example bit-field 0 (EF0) - <Bit-field description>

Table 1347. Reset value definitions

Value	Description
0	Reset value 0. Used for single-bit fields.
1	Reset value 1. Used for single-bit fields.
0xNN	Hexadecimal representation of reset value. Used for multi-bit fields.
n/r	Field not reseted
*	Special reset condition, described in textual description of the bit-field. Used for example when reset value is taken from an input signal.

Table 1348. Bit-field type definitions

Value	Description
r	Read-only. Writes have no effect.
rw	Readable and writable.
rw*	Readable and writeable. Special condition for write, described in textual description of the bit-field.
wc	Write-clear. Readable, and cleared when written with a 1. Writing 0 has no effect.

*Table 1349.*GRSPW2 registers

APB address offset	Register acronym	Register name
0x00	CTRL	Control
0x04	STS	Status
0x08	DEFADDR	Default address
0x0C	CLKDIV	Clock divisor
0x10	DKEY	Destination key
0x14	TC	Time-code
0x18 - 0x1C	-	RESERVED
0x20, 0x40, 0x60, 0x80	DMACTRL	DMA control/status, channel 1 - 4 [1)]
0x24, 0x44, 0x64, 0x84	DMAMAXLEN	DMA RX maximum length, channel 1 - 4 [1)]
0x28, 0x48, 0x68, 0x88	DMATXDESC	DMA transmit descriptor table address, channel 1 - 4 [1)]
0x2C, 0x4C, 0x6C, 0x8C	DMARXDESC	DMA receive descriptor table address, channel 1 - 4 [1)]
0x30, 0x50, 0x70, 0x90	DMAADDR	DMA address, channel 1 - 4 [1)]
0x34, 0x54, 0x74, 0x94	-	RESERVED
0x38, 0x58, 0x78, 0x98	-	RESERVED
0x3C, 0x5C, 0x7C, 0x9C	-	RESERVED
0xA0	INTCTRL	Interrupt distribution control [2)]
0xA4	INTRX	Interrupt receive [2)]
0xA8	ACKRX / INTRXEXT	Interrupt-acknowledge receive / Interrupt receive extended [2)]
0xAC	INTTO	Interrupt timeout [2) 5)]
0xB0	INTTOEXT	Interrupt timeout extended [2) 5)]
0xB4	TICKMASK	Interrupt tick-out mask [2)]
0xB8	TICKMASKEXT / AUTO-ACK	Interrupt auto acknowledge mask / Interrupt tick-out mask extended [2)]
0xBC	INTCFG	Interrupt distribution configuration [2)]
0xC0	-	RESERVED
0xC4	ISR	Interrupt distribution ISR[2)]
0xC8	ISREXT	Interrupt distribution Extended ISR [2)]
0xCC	-	RESERVED
0xD0	PRESCALER	Interrupt distribution prescaler reload [2) 3)]
0cD4	ISRTIMER	Interrupt distribution ISR timer reload [2) 3)]
0xD8	IATIMER	Interrupt distribution INT / ACK timer reload [2) 3)]
0xDC	ICTIMER	Interrupt distribution change timer reload [2) 3)]
0xE0	PNPVEND	SpaceWire PnP Device Vendor and Product ID [4)]
0xE4	PNPLINKINFO	SpaceWire PnP Link Information [4)]
0xE8	PNPOA0	SpaceWire PnP Owner Address 0 [4)]
0xEC	PNPOA1	SpaceWire PnP Owner Address 1 [4)]
0xF0	PNPOA2	SpaceWire PnP Owner Address 2 [4)]
0xF4	PNPDEVID	SpaceWire PnP Device ID [4)]
0xF8	PNPUVEND	SpaceWire PnP Unit Vendor and Product ID [4)]
0xFC	PNPUSN	SpaceWire PnP Unit Serial Number [4)]

*Table 1349.*GRSPW2 registers

APB address offset	Register acronym	Register name

Note 1: Registers for non-implemented DMA channels are reserved. The number of implemented DMA channels is indicated by the NCH field in the Control register.

Note 2: Register is only available if the interrupt distribution is supported, which is indicated by the value of the CTRL.ID bit.

Note 3: Register is only available if support for the corresponding timer is implemented, otherwise the register is reserved. This can be detected by probing the RL field of corresponding register.

Note 4: Register is only available if the SpaceWire Plug-and-Play is supported, which is indicated by the value of the CTRL.PNPA field.

Note 5: Register is only available if support for the Interrupt distribution ISR timer is implemented. This can be detected by probing the ISRTIMER.RL field.

77.13.1 Control Register

*Table 1350.*0x00 - CTRL - Control

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RA	RX	RC	NCH		PO	CC	ID	R	LE	PS	NP	PNPA		RD	RE	PE	R	TL	TF	TR	TT	LI	TQ	R	RS	PM	TI	IE	AS	LS	LD
*	*	*	*		*	*	*	0	0	0	*	*		0	*	*	0	0	0	0	0	0	0	0	0	0	0	0	*	0	0
r	r	r	r		r	r	r	r	rw	rw	rw	r		rw	rw	rw	r	rw	rw	rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw	rw

31 RMAP available (RA) - Set to one if the RMAP target is available. Value determined by the *rmap* VHDL generic.

30 RX unaligned access (RX) - Set to one if unaligned writes are available for the receiver. Value determined by the *rxunaligned* VHDL generic.

29 RMAP CRC available (RC) - Set to one if RMAP CRC is enabled in the core. Value determined by the *rmapcrc* VHDL generic.

28: 27 Number of DMA channels (NCH) - The number of available DMA channels minus one (Number of channels = NCH+1). Value determined by the *dmachan* VHDL generic.

26 Number of ports (PO) - The number of available SpaceWire ports minus one. Value determined by the *ports* VHDL generic.

25 CCSDS/CCITT CRC-16 and 16-bit ISO-checksum available (CC) - Set to one if this crc logic is enabled in the core. Value determined by the ccsdscrc VHDL generic.

24 Interrupt distribution available (ID) - Set to 1 if interrupt distribution support is available, otherwise set to 0. If set to 1, then the INTCTRL.NUMINT field indicates the number of supported interrupt numbers. Value determined by the *interruptdist* VHDL generic (ID = 1 if *interruptdist* /= 0).

23 RESERVED

22 Loop-back enable (LE). The value of this bit is driven on the LOOPBACK output signal.

21 Port select (PS) - Selects the active port when the CTRL.NP bit is zero. ‘0’ selects the port connected to data and strobe on index 0, while ‘1’ selects index data and strobe on 1. Only available in two-port configurations, which is indicated by CTRL.PO bit. This bit is reserved in one-port-configurations.

20 No port force (NP) - Disable port force. When this bit is set, the CTRL.PS bit cannot be used to select the active port. Instead, the active port is automatically selected by checking the activity on the respective receive links. Only available in two-port configurations, which is indicated by CTRL.PO bit. Reserved bit in one-port configurations. Reset value is set from input signal RMAPEN if RMAP target is available (indicated by CTRL.RA bit), otherwise the reset value is ‘0’.

19: 18 SpaceWire Plug-and-Play available (PNPA) - Indicates SpaceWire Plug-and-Play support. 0 = No support, 1 = Support for the device identification, 2 = Support for device identification and configuration. See section 77.10 for details. Value determined by the *pnp* VHDL generic.

17 RMAP buffer disable (RD) - If set, only one RMAP buffer is used. This ensures that all RMAP commands will be executed consecutively. Only available if the *rmap* VHDL generic is set to 1, otherwise the bit is reserved.

16 RMAP Enable (RE) - Enable RMAP target. Only available if *rmap* VHDL generic is set to 1, otherwise the bit is reserved. Reset value take from RMAPEN input signal.

15 SpaceWire Plug-and-Play enable (PE) - Enable SpaceWire Plug-and-Play support. Only available if the CTRL.PA bit is 1, otherwise this bit is reserved. Reset value taken from the PNPEN input signal.

14 RESERVED

13 Transmitter enable lock control (TL) - Enables / disables the transmitter enable lock functionality described by the DMACTRL.TL bit. 0 = Disabled, 1 = Enabled.

12 Time-code control flag filter (TF) - When set to 1, a received time-code must have its control flag bits set to “00” to be considered valid. When set to 0, all control flag bits are allowed. Note that if the interrupt code receive enable bit (INTCFG.IR) is set to 1, then the only time-code control flag bits of “00” are allowed, regardless of the setting of this bit.

11 Time Rx Enable (TR) - Enable time-code reception.

10 Time Tx Enable (TT) - Enable time-code transmission.

9 Link error IRQ (LI) - Enables / disables AMBA interrupt generation when a link error occurs. Note that the CTRL.IE bit also must be set for this bit to have any effect.

8 Tick-out IRQ (TQ) - Enables / disables AMBA interrupt generation when a valid time-code is received. Note that the CTRL.IE bit also must be set for this bit to have any effect.

7 RESERVED

Table 1350. 0x00 - CTRL - Control

6	Reset (RS) - Make complete reset of the SpaceWire node. Self clearing.
5	Promiscuous Mode (PM) - Enable promiscuous mode. See section 77.6.10.
4	Tick In (TI) - The host can generate a tick by writing a one to this bit. This incrementd the timer counter (TC.TIMECNT), and the new value is transmitted. This bit will stay high until the time-code has been sent. Note that the link interface must be in run-state for the time-code to be sent.
3	Interrupt Enable (IE) - If set, AMBA interrupt generation is enabled for the events that are individually maskable by the CTRL.TQ, CTRL.LI, INTCTRL.IQ, INTCTRL.AQ, and INTCTRL.TQ bits.
2	Autostart (AS) - Automatically start the link when a NULL has been received. Reset value is set from input signal RMAPEN if RMAP target is available (CTRL.RA bit = 1), otherwise the reset value is ‘0’.
1	Link Start (LS) - Start the link, i.e. allow a transition from ready-state to started-state.
0	Link Disable (LD) - Disable the SpaceWire codec.

77.13.2 Status Register

*Table 1351.*0x04 - STS - Status

31 28	27 26	25 24	23 22 21	20 19 18 17 16 15 14 13 12 11 10	9	8	7	6 5	4	3	2	1	0
RESERVED	NRXD	NTXD	LS	RESERVED	AP	EE	IA	RES	PE	DE	ER	CE	TO
0x00	*	*	0x0	0x000	0	0	0	0x0	0	0	0	0	0
r	r	r	r	r	r	wc	wc	r	wc	wc	wc	wc	wc

31: 28 RESERVED

27: 26 Number of receive descriptors (NRXD) - Shows the size of the DMA receive descriptor table. 0b00 = 128, 0b01 = 256, 0b10 = 512, 0b11 = 1024

25 24 Number of transmit descriptors (NTXD) - Shows the size of the DMA transmit descriptor table. 0b00 = 64, 0b01 = 128, 0b10 = 256, 0b11 = 512

23: 21 Link State (LS) - The current state of the start-up sequence. 0 = Error-reset, 1 = Error-wait, 2 = Ready, 3 = Started, 4 = Connecting, 5 = Run.

20: 10 RESERVED

9 Active port (AP) - Shows the currently active port. '0' = Port 0 and '1' = Port 1 where the port numbers refer to the index number of the data and strobe signals. Only available if the *ports* generic is set to 2.

8 Early EOP/EEP (EE) - Set to one when a packet is received with an EOP after the first byte for a non-rmap packet and after the second byte for an RMAP packet.

7 Invalid Address (IA) - Set to one when a packet is received with an invalid destination address field, i.e it does not match the DEFADDR register.

6: 5 RESERVED

4 Parity Error (PE) - A parity error has occurred.

3 Disconnect Error (DE) - A disconnection error has occurred.

2 Escape Error (ER) - An escape error has occurred.

1 Credit Error (CE) - A credit has occurred.

0 Tick Out (TO) - A new time count value was received and is stored in the time counter field.

77.13.3 Default Address Register

*Table 1352.*0x08 - DEFADDR - Default address

31 16	15 8	7 0
RESERVED	DEFMASK	DEFADDR
0x0000	0x00	*
r	rw	rw

31: 8 RESERVED

15: 8 Default mask (DEFMASK) - Default mask used for node identification on the SpaceWire network. This field is used for masking the address before comparison. Both the received address and the DEFADDR.DEFADDR field are anded with the inverse of this field before the address check.

7: 0 Default address (DEFADDR) - Default address used for node identification on the SpaceWire network. Reset value: 254 (taken from the *nodeaddr* VHDL generic when /= 255, else from the RMAPNODEADDR input signal)

77.13.4 Clock Divisor Register

Table 1353. 0x0C - CLKDIV - Clock divisor

31 16	15 8	7 0
RESERVED	CLKDIVSTART	CLKDIVRUN
0x0000	*	*
r	rw	rw

31: 16 RESERVED

15: 8 Clock divisor startup (CLKDIVSTART) - The value of this field is used as a clock divider during startup (link interface is in other states than run-state). See 77.3.6 for details on how to set this field. Reset value taken from the CLKDIV10 input signal.

7: 0 Clock divisor run (CLKDIVRUN) - The value of this field is used as a clock divider when the link-interface is in run-state. See 77.3.6 for details on how to set this field. Reset value taken from the CLKDIV10 input signal.

77.13.5 Destination Key Register

Table 1354. 0x10 - DKEY - Destination key

31 8	7 0
RESERVED	DESTKEY
0x000000	*
r	rw

31: 8 RESERVED

7: 0 Destination key (DESTKEY) - RMAP destination key. Only available if the *rmap* VHDL generic is set to 1. Set from *destkey* VHDL generic.

77.13.6 Time-code Register

Table 1355. 0x14 - TC - Time-code

31 8	7 6	5 0
RESERVED	TCTRL	TIMECNT
0x000000	0x0	0x00
r	rw	rw

31: 8 RESERVED

7: 6 Time control flags (TCTRL) - The current value of the time-code control flags. Sent in a time-code each time the TICKIN signal is set, or the CTRL.TI bit is written. This field is also updated with the control flags from all received time-codes, and with the value of the TIMEIN[7:6] signals if TICKINRAW is asserted.

5: 0 Time counter (TIMECNT) - The current time value. Incremented, and transmitted in a time-code, each time the TICKIN signal is set, or the CTRL.TI bit is written. This field is also updated with the time value from all received time-codes, and with the value of the TIMEIN[5:0] signals if TICKINRAW is asserted. Note that the register can be written, but that the written value is not transmitted, since the value is incremented before transmission.

77.13.7 DMA Control/Status

*Table 1356.*0x20, 0x40, 0x60, 0x80 - DMACTRL - DMA control/status

31 26	25 24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTNUM	RES	EP	TR	IE	IT	RP	TP	TL	LE	SP	SA	EN	NS	RD	RX	AT	RA	TA	PR	PS	AI	RI	TI	RE	TE
*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rw	r	wc	wc	rw	rw	wc	wc	wc	rw	rw	rw	rw	rw	rw	r	rw	wc	wc	wc	wc	rw	rw	rw	rw	rw

31: 26 Interrupt-number (INTNUM) - The interrupt-number used for this DMA channel when sending an interrupt-code that was generated due to any of the events maskable by the DMACTRL.IE and DMACTRL.IT bits. Reset value is taken from the IRQTXDEFAULT input signal. Field is only present if interrupt distribution is supported, which is indicated by the CTRL.ID bit. Note that this field must be set to a value within the range defined by the INCTRL.NUMINT and INTCTRL.BASEINT fields. A value outside the range will result in no interrupt-code being sent. This field is only available when distributed interrupt is implemented.

25: 24 RESERVED

23 EEP termination (EP) - Set to 1 when a received packet for the corresponding DMA channel ended with an Error End of Packet (EEP) character.

22 Truncated (TR) - Set to 1 when a received packet for the corresponding DMA channel is truncated due to a maximum length violation.

21 Interrupt-code transmit enable on EEP (IE) - When set to 1, and the interrupt-code transmit enable bit (INTCTRL.IT) is set, an interrupt-code is generated when a received packet on this DMA channel ended with an Error End of Packet (EEP) character. Field is only present if interrupt distribution is supported, which is indicated by the CTRL.ID bit. This field is only available when distributed interrupt is implemented.

20 Interrupt-code transmit enable on truncation (IT) - When set to 1, and the interrupt-code transmit enable (INTCTRL.IT) bit in the Interrupt distribution control register is set, an interrupt-code is generated when a received packet on this DMA channel is truncated due to a maximum length violation. Field is only present if interrupt distribution is supported, which is indicated by the CTRL.ID bit. This field is only available when distributed interrupt is implemented.

19 Receive packet IRQ (RP) - This bit is set to 1 when an AMBA interrupt was generated due to the fact that a packet was received for the corresponding DMA channel.

18 Transmit packet IRQ (TP) - This bit is set to 1 when an AMBA interrupt was generated due to the fact that a packet was transmitted for the corresponding DMA channel.

17 Transmitter enable lock (TL) - This bit is set to 1 if the CTRL.TL bit is set, and the transmitter for the corresponding DMA channel is disabled due to a link error (controlled by the DMACTRL.LE bit). While this bit is set, it is not possible to re-enable the transmitter (e.g. not possible to set the DMACTRL.TE bit to 1).

16 Link error disable (LE) - Disable transmitter when a link error occurs. No more packets will be transmitted until the transmitter is enabled again.

15 Strip pid (SP) - Remove the pid byte (second byte) of each packet. The address byte (first byte) will also be removed when this bit is set, independent of the value of the DMACTRL.SA bit.

14 Strip addr (SA) - Remove the addr byte (first byte) of each packet.

13 Enable addr (EN) - Enable separate node address for this channel.

12 No spill (NS) - If cleared, packets will be discarded when a packet is arriving and there are no active descriptors. If set, the core will wait for a descriptor to be activated.

11 Rx descriptors available (RD) - Set to one, to indicate to the core that there are enabled descriptors in the descriptor table. Cleared by the core when it encounters a disabled descriptor.

10 RX active (RX) - Is set to ‘1’ if a reception to the DMA channel is currently active, otherwise it is ‘0’.

9 Abort TX (AT) - Set to one to abort the currently transmitting packet and disable transmissions. If no packet is currently being transmitted, the only effect is to disable transmissions. Self clearing.

Table 1356. 0x20, 0x40, 0x60, 0x80 - DMACTRL - DMA control/status

8	RX AHB error (RA) - An error response was detected on the AHB bus while this receive DMA channel was accessing the bus.
7	TX AHB error (TA) - An error response was detected on the AHB bus while this transmit DMA channel was accessing the bus.
6	Packet received (PR) - This bit is set each time a packet has been received.
5	Packet sent (PS) - This bit is set each time a packet has been sent.
4	AHB error interrupt (AI) - If set, an interrupt will be generated each time an AHB error occurs when this DMA channel is accessing the bus.
3	Receive interrupt (RI) - If set, an interrupt will be generated when a packet is received, if the interrupt enable (IE) bit in the corresponding receive descriptor is set as well. This happens both if the packet is terminated by an EEP or EOP.
2	Transmit interrupt (TI) - If set, an interrupt will be generated when a packet is transmitted, if the interrupt enable (IE) bit in the corresponding transmit descriptor is set as well. The interrupt is generated regardless of whether the transmission was successful or not.
1	Receiver enable (RE) - Set to one when packets are allowed to be received to this channel.
0	Transmitter enable (TE) - Enables the transmitter for the corresponding DMA channel. Setting this bit to 1 will cause the DMA channel to read a new descriptor and try to transmit the packet it points to. Note that it is only possible to set this bit to 1 if the TL bit is 0. This bit is automatically cleared when the DMA channel encounters a descriptor which is disabled, or if a link error occurs during the transmission of a packet, and the LE bit is set.

77.13.8 DMA RX Maximum Length

*Table 1357.*0x24, 0x44, 0x64, 0x84 - DMAMAXLEN - DMA RX maximum length

31 25	24 2	1 0
RESERVED	RXMAXLEN	RES
0x00	n/r	0x0
r	rw	r

31: 25 RESERVED

24: 2 RX maximum length (RXMAXLEN) - Receiver packet maximum length, counted in 32-bit words.

1: 0 RESERVED

77.13.9 DMA Transmit Descriptor Table Address

*Table 1358.*0x28, 0x48, 0x68, 0x88 - DMATXDESC - DMA transmit descriptor table address

31 x+1	x 4	3 0
DESCBASEADDR	DESCSEL	RESERVED
n/r	0x00	0x0
rw	rw	r

31: x+1 Descriptor table base address (DESCBASEADDR) - Sets the base address of the descriptor table. The number of bits in this field depends on the size of the DMA transmit descriptor table. The value of x is given by the formula: 9 + STS.NTXD.

x: 4 Descriptor selector (DESCSEL) - Offset into the descriptor table. Shows which descriptor is currently used by the core. For each new descriptor read, the selector will increase with 16 and eventually wrap to zero again. The number of bits in this field depends on the size of the DMA transmit descriptor table. The value of x is given by the formula: 9 + STS.NTXD.

3: 0 RESERVED

77.13.10 DMA Receive Descriptor Table Address

*Table 1359.*0x2C, 0x4C, 0x6C, 0x8C - DMARXDESC - DMA receive descriptor table address

31 x+1	x 3	2 0
DESCBASEADDR	DESCSEL	RESERVED
n/r	0x00	0x0
rw	rw	r

31: 10 Descriptor table base address (DESCBASEADDR) - Sets the base address of the descriptor table. The number of bits in this field depends on the size of the DMA receive descriptor table. The value of x is given by the formula: 9 + STS.NRXD.

9: 3 Descriptor selector (DESCSEL) - Offset into the descriptor table. Shows which descriptor is currently used by the core. For each new descriptor read, the selector will increase with 8 and eventually wrap to zero again. The number of bits in this field depends on the size of the DMA receive descriptor table. The value of x is given by the formula: 9 + STS.NRXD

2: 0 RESERVED

77.13.11 DMA Address

*Table 1360.*0x30, 0x50, 0x70, 0x90 - DMAADDR - DMA address

31 16	15 8	7 0
RESERVED	MASK	ADDR
0x0000	n/r	n/r
r	rw	rw

31: 8 RESERVED

15: 8 Mask (MASK) - Mask used for node identification on the SpaceWire network. This field is used for masking the address before comparison. Both the received address and the ADDR field are anded with the inverse of MASK before the address check.

7: 0 Address (ADDR) - Address used for node identification on the SpaceWire network for the corresponding dma channel when the EN bit in the DMA control register is set.

77.13.12 Interrupt Distribution Control

*Table 1361.*0xA0 - INTCTRL - Interrupt distribution control

31 26	25	24	23	22 21	20	19	18	17 16	15	14	13	12 8	7	6	5 0
INTNUM	RS	EE	IA	RES	TQ	AQ	IQ	RES	AA	AT	IT	RES	ID	II	TXINT
*	0	0	0	0	0	0	0	0	0	0	0	0x00	0	0	*
rw	rw	rw	rw	r	rw	rw	rw	r	rw	rw	rw	r	wc	rw	rw

31: 26 Interrupt number (INTNUM) - The interrupt-number used when sending an interrupt-code that was generated due to any of the events maskable by the RS, ER or IA bits. Reset value is taken from the IRQTXDEFAULT input signal. Note that this field must be set to a value within the range defined by the NUMINT and BASEINT fields. A value outside the range will result in no interrupt-code being sent.

25 Interrupt-code transmit on run-state entry (RS) - If set to 1, and interrupt-code with the interrupt number specified in the INTNUM field is sent each time the link interface enters run-state.

24 Interrupt-code transmit on early EOP/EEP (EE) - If set to 1, an interrupt-code with the interrupt number specified in the INTNUM field is sent each time an event occurs such that the STS.EE bit is set to 1 (even if the bit was already set when the event occurrepriod).

23 Interrupt-code transmit on invalid address (IA) - If set to 1, an interrupt-code with the interrupt number specified in the INTNUM field is sent each time an event occurs such that the STS.IA bit is set to 1 (even if the bit was already set when the event occurred).

22: 21 RESERVED

20 Interrupt-code timeout IRQ enable (TQ) - When set to 1, an AMBA interrupt is generated when a bit in the INTTO register is set. Note that the IE bit in the Control register also must be set for this bit to have any effect. Bit is only available if support for the Interrupt distribution ISR timer is implemented.

19 Interrupt-code-acknowledge receive IRQ enable (AQ) - When set to 1, an AMBA interrupt is generated when an interrupt-acknowledge-code is received for which the corresponding bit in the Interrupt tick-out mask register is set, and the core was the source of the matching interrupt-code. Note that the IE bit in the Control register also must be set for this bit to have any effect.

18 Interrupt-code receive IRQ enable (IQ) - When set to 1, an AMBA interrupt is generated when an interrupt-code is received for which the corresponding bit in the Interrupt tick-out mask register is set to 1. Note that the IE bit in the Control register also must be set for this bit to have any effect.

17: 16 RESERVED

15 Handle all interrupt acknowledgement codes (AA) - Is set to 0, only those received interrupt acknowledgement codes that match an interrupt code sent by software are handled. If set to 1, all received interrupt acknowledgement codes are handled.

14 Interrupt acknowledgement / extended interrupt tickout enable (AT) - When set to 1, the internal tickout signal is set when an interrupt acknowledgement code or extended interrupt code is received such that a bit in the AUTOACK / INTRXEXT register is set to 1 (even if the bit was already set when the code was received).

Table 1361. 0xA0 - INTCTRL - Interrupt distribution control

13	Interrupt tickout enable (IT) - When set to 1, the internal tickout signal is set when an interrupt code is received such that a bit in the INTRX register is set to 1 (even if the bit was already set when the code was received).
12: 8	RESERVED
7	Interrupt-code discarded (ID) - This bit is set to 1 when and interrupt-code that software tried to send by writing the II bit was discarded, either because there already was a pending request to send an interrupt-code with the same interrupt-number, or because the corresponding ISR bit is 1. There is a one clock cycle delay between the II bit being written and this bit being set.
6	Interrupt-code tick-in (II) - When this field is written to 1 the interrupt- / interrupt-acknowledge-code specified in the TXINT field will be sent. The actual sending of the interrupt- / interrupt-acknowledge-code might be delayed, depending on the value for the corresponding ISR bit and INT/ACK-timer. Note that the interrupt-code transmit enable bit (IT) must be set to '1', otherwise writing this bit has no effect. This bit is automatically cleared and always reads '0'. Writing a '0' has no effect.
5: 0	Transmit interrupt- / interrupt-code (TXINT) - The interrupt- / interrupt-acknowledge-code that the core will send when the Interrupt-code tick-in bit (II) is written with 1. Reset value for bit 5 is '0', while bits 4:0 are set from the input signal IRQTXDEFAULT. Note that bits 4:0 of this field must be set to a value within the range defined by the NUMINT and BASEINT fields. A value outside the range will result in no interrupt-code being sent.

77.13.13 Interrupt Receive

Table 1362. 0xA4 - INTRX - Interrupt-code receive

31	0
RXIRQ	
0x00000000	
wc	

31: 0 Received interrupt-code (RXIRQ) - Each bit corresponds to the interrupt number with the same number as the bit index. The core sets a bit to 1 when it receives an interrupt-code for which the corresponding bit in the Interrupt tick out mask register is set to 1. Note that the number of implemented bits depends on the number of supported interrupts (INTCTRL.NUMINT field).

77.13.14 Interrupt-acknowledge-code Receive

Table 1363. 0xA8 - ACKRX / INTRXEXT - Interrupt-acknowledge-code receive / Interrupt receive extended

31	0
RXACK / INTRXEXT	
0x00000000	
wc	

31: 0 Received interrupt-acknowledge-code (RXACK) / Interrupt receive extended (INTRXEXT) - Each bit corresponds to the interrupt number with the same number as the bit index. The core sets a bit to 1 when it receives a interrupt-acknowledge-code for which the corresponding bit in the Interrupt tick out mask register is set, and for which the matching interrupt-code was sent by software (valid for interrupt-acknowledge). Note that the number of implemented bits depends on the number of supported interrupts (INTCTRL.NUMINT field). When extended interrupt mode is enabled this register is an extension of the Interrupt Receive register for interrupt 32-63.

77.13.15 Interrupt Timeout

Table 1364. 0xAC - INTTO - Interrupt timeout

31	0
INTTO	
0x00000000	
wc	

31: 0 Interrupt-code timeout (INTTO) - Each bit corresponds to the interrupt number with the same number as the bit index. The core sets a bit to 1 when an interrupt-code that was sent by software doesn't receive an interrupt-acknowledge-code for the duration of a timeout period (specified in the Interrupt distribution ISR timer reload registers), and if the corresponding bit in the Interrupt-code tick out mask register is set. Note that the number of implemented bits depends on the number of supported interrupts (INTCTRL.NUMINT field).

77.13.16 Interrupt Timeout Extended

Table 1365. 0xB0 - INTTOEXT - Interrupt timeout extended

31	0
INTTOEXT	
0x00000000	
wc	

31: 0 Interrupt timeout extended (INTTOEXT) - When extended interrupt mode is enabled, each bit corresponds to the interrupt number between 32 and 63. The core sets a bit to 1 when an interrupt-code that was sent by software and the time specified in the Interrupt distribution ISR timer reload registers has past and if the corresponding bit in the Interrupt-code tick out mask register is set. Note that the number of implemented bits depends on the number of supported interrupts (INTCTRL.NUMINT field).

77.13.17 Interrupt Tick-out Mask

Table 1366. 0xB4 - TICKMASK - Interrupt tick-out mask

31	0
MASK	
0x00000000	
rw*	

31: 0 Interrupt tick-out mask (MASK) - Each bit corresponds to the interrupt number with the same value as the bit index. If a bit is set, the TICKOUT signal as well as the corresponding bit in the Interrupt-code receive register, Interrupt-acknowledge-code receive register, and Interrupt-code timeout register is set when respective event occurs. Note that the number of implemented bits depends on the number of supported interrupts (INTCTRL.NUMINT field).

77.13.18 Interrupt-code Auto Acknowledge Mask

Table 1367. 0xB8 - AUTOACK / TICKMASKEXT - Interrupt-code auto acknowledge mask / Interrupt tick-out mask extended.

31	0
AAMASK	
0x00000000	
rw*	

31: 0 Auto acknowledge mask (AAMASK) - For each bit set to 1, the core will automatically send an interrupt-acknowledge-code when it receives an interrupt-code with the corresponding interrupt number. If the interrupt distribution timers are implemented (VHDL generic intiatimerbits /= 0) and enabled, then the core will reload the INT-to-ACK timer and wait until it expires before the interrupt-acknowledge-code is sent. Note that the number of implemented bits depends on the number of supported interrupts (INTCFG.NUMINT field). When extended interrupt mode is enabled this register is an extension of the Interrupt Tick-out Mask register.

77.13.19Interrupt Distribution Configuration

Table 1368. 0xBC - INTCFG - Interrupt distribution configuration

31 26	25 20	19 14	13 8	7 4	3	2	1	0
INTNUM3	INTNUM2	INTNUM1	INTNUM0	NUMINT	PR	IR	IT	EE
*	*	*	*	0	0	0	0	0
rw	rw	rw	rw	r	rw	rw	rw	rw

31: 26 Interrupt number 3 (INTNUM3) - Defines the interrupt number to support when the device supports fewer than 32 interrupts.

25: 20 Interrupt number 2 (INTNUM2) - Defines the interrupt number to support when the device supports fewer than 32 interrupts.

19: 14 Interrupt number 1 (INTNUM1) - Defines the interrupt number to support when the device supports fewer than 32 interrupts.

13: 8 Interrupt number 0 (INTNUM0) - Defines the interrupt number to support when the device supports fewer than 32 interrupts.

7: 4 Number of interrupts (NUMINT) - Indicates the number of supported interrupts according to the formula: *Number of interrupts* = 2^{NUMINT}.

3 Interrupt- / interrupt-acknowledge-code priority (PR) - When set to 0, interrupt-codes have priority over interrupt-acknowledge-codes when there are multiple codes waiting to be sent. When set to 1, interrupt-acknowledge-codes have priority.

2 Interrupt receive enable (IR) - Enable interrupt- / interrupt-acknowledge-code reception.

1 Interrupt transmit enable (IT) - Enable interrupt- / interrupt-acknowledge-code transmission. Must be set to 1 in order for any interrupt- / interrupt-acknowledge-codes to be sent.

0 Enable external interrupt (EE) - Enable the external interrupt mode, which enable the core to use and interpret the interrupt-acknowledge-code as interrupt 32-63.

77.13.20 Interrupt Distribution ISR

Table 1369. 0xC4 - ISR - Interrupt distribution ISR

31	0
ISR	
0x00000000	
wc	

31: 0 Interrupt distribution ISR (ISR) - Each bit index holds the ISR bit value for the corresponding interrupt number. A bit value of 1 indicates that a interrupt-code with the corresponding interrupt number has been received, and that it has not yet been acknowledged (and not yet timed-out). A bit value of 0 indicates that either no interrupt-code with that interrupt number has been received, or that the interrupt has been acknowledged (or timed out). This register is write-clear, but should normally only be used for diagnostics and/or FDIR. Note that the number of implemented bits depends on the number of supported interrupts (INTCTRL.NUMINT field).

77.13.21 Interrupt Distribution ISR Extended

Table 1370. 0xC8 - ISREXT - Interrupt distribution ISR extended

31	0
ISR	
0x00000000	
wc	

31: 0 Interrupt distribution ISR (ISR) - Each bit index holds the ISR bit value for the corresponding interrupt number. A bit value of 1 indicates that a interrupt-code with the corresponding interrupt number has been received, and that it has not yet been acknowledged (and not yet timed-out). A bit value of 0 indicates that either no interrupt-code with that interrupt number has been received, or that the interrupt has been acknowledged (or timed out). This register is write-clear, but should normally only be used for diagnostics and/or FDIR. Note that the number of implemented bits depends on the number of supported interrupts (INTCTRL.NUMINT field).

77.13.22 Interrupt Distribution Prescaler Reload

Table 1371. 0xD0 - PRESCALER - Interrupt distribution prescaler reload

31	30	0
R	RL	
0	*	
r	rw	

31 RESERVED

30: 0 Prescaler reload (RL) - Reload value for the interrupt distribution prescaler. The prescaler runs on the system clock, and an internal tick is generated every RL+1 cycle. The number of bits implemented for this field is set by the VHDL generic *intscalerbits*, and might be lower than the 31 depicted here. Any unimplemented bits are reserved. Reset value set from the input signal INTPRELOAD.

77.13.23 Interrupt Distribution ISR Timer Reload

*Table 1372.*0xD4 - ISRTIMER - Interrupt distribution ISR timer reload

31	30 … 0
EN	RL
1	*
rw	rw

31 Timer enable (EN) - Enables the use of ISR timer for each ISR bit. One global timer enable bit used for all ISR bits. If this bit is set to 1, the timer for each ISR bit is reloaded with the value in the RL field when the ISR bit is set. If the timer expires before an interrupt-code-acknowledge has been received, then ISR bit is cleared.

30: 0 Timer reload (RL) - Common reload value for the interrupt distribution ISR timers. The number of bits implemented for this field is set by the VHDL generic *intisrtimerbits*, and might be lower than the 31 depicted here. Any unimplemented bits are reserved. Reset value set from the input signal INTTRELOAD.

77.13.24 Interrupt Distribution INT/ACK Timer Reload

*Table 1373.*0xD8 - IATIMER - Interrupt distribution INT / ACK timer reload

31	30 … 0
EN	RL
1	*
rw	rw

31 Timer enable (EN) - Enables the use of timers to control the time between an interrupt-code and an interrupt-acknowledge-code, and vice versa. One global timer enable bit is used for all ISR bits. If this bit is set to 1, the timer for each ISR bit is reloaded with the value in the RL field each time an interrupt-code is received. The core will then wait until the timer expires before an interrupt-code-acknowledge with the same interrupt number is sent. The same applies when an interrupt-code-acknowledge is received and a new interrupt-code with the same number should be sent.

30: 0 Timer reload (RL) - The number of bits implemented for this field is set by the VHDL generic *intia-timerbits*, and might be lower than the 31 depicted here. Any unimplemented bits are reserved. Reset value set from the input signal INTIARELOAD.

77.13.25 Interrupt Distribution Change Timer Reload

*Table 1374.*0xDC - ICTIMER - Interrupt distribution change timer reload

31	30 … 0
EN	RL
1	*
rw	rw

31 Timer enable (EN) - Enables the use of timers to control the time that must pass between two changes in value for the same ISR bit. One global timer enable bit is used for all ISR bits. If this bit is set to 1, the timer for each ISR bit is reloaded with the value in the RL field each time the ISR bit changes value. All potential interrupt- / interrupt-acknowledge-codes received before the timer expires is discarded.

30: 0 Timer reload (RL) - The number of bits implemented for this field is set by the VHDL generic *intc-timerbits*, and might be lower than the 31 depicted here. Any unimplemented bits are reserved. Reset value set from the input signal INTCRELOAD.

77.13.26 SpaceWire Plug-and-Play - Device Vendor and Product ID

Table 1375. 0xE0 - PNPVEND - SpaceWire Plug-and-Play - Device Vendor and Product ID

31 16	15 0
VEND	PROD
*	*
r	r

Note: Register is double mapped from SpaceWire Plug-and-Play address space into APB address space. See section 77.10 for details.

77.13.27 SpaceWire Plug-and-Play - Link Information

Table 1376. 0xE4 - PNPLINFO -SpaceWire Plug-and-Play - Link Information

31 24	23 22	21	20 16	15 13	12 8	7	6	5	4 0
OLA	OAL	R	OL	RES	RL	T	U	R	LC
0x00	0x0	0	0x0	0x0	0x0	1	0	0	0x13
r	r	r	r	r	r	r	r	r	r

Note: Register is double mapped from SpaceWire Plug-and-Play address space into APB address space. See section 77.10 for details.

77.13.28 SpaceWire Plug-and-Play - Owner Address 0

Table 1377. 0xE8 - PNPOA0 - SpaceWire Plug-and-Play - Owner Address 0

31 0
RA
0x00000000
r

Note: Register is double mapped from SpaceWire Plug-and-Play address space into APB address space. See section 77.10 for details.

77.13.29 SpaceWire Plug-and-Play - Owner Address 1

Table 1378. 0xEC - PNPOA1 - SpaceWire Plug-and-Play - Owner Address 1

31 0
RA
0x00000000
r

Note: Register is double mapped from SpaceWire Plug-and-Play address space into APB address space. See section 77.10 for details.

77.13.30 SpaceWire Plug-and-Play - Owner Address 2

Table 1379. 0xF0 - PNPOA2 - SpaceWire Plug-and-Play - Owner Address 2

31 0
RA
0x00000000
r

Note: Register is double mapped from SpaceWire Plug-and-Play address space into APB address space. See section 77.10 for details.

77.13.31 SpaceWire Plug-and-Play - Device ID

Table 1380. 0xF4 - PNPDEVID - SpaceWire Plug-and-Play - Device ID

31	0
DID	
0x00000000	
r	

Note: Register is double mapped from SpaceWire Plug-and-Play address space into APB address space. See section 77.10 for details.

77.13.32 SpaceWire Plug-and-Play - Unit Vendor and Product ID

Table 1381. 0xF8 - PNPUVEND - SpaceWire Plug-and-Play - Unit Vendor and Product ID

31	16	15	0
VEND		PROD	
*		*	
rw		rw	

Note: Register is double mapped from SpaceWire Plug-and-Play address space into APB address space. See section 77.10 for details. This register is read-only in SpaceWire Plug-and-Play interface, while it is writable from the APB address space.

77.13.33 SpaceWire Plug-and-Play - Unit Serial Number

Table 1382. 0xFC - PNPUSN - SpaceWire Plug-and-Play - Unit Serial Number

31	0
USN	
*	
rw	

Note: Register is double mapped from SpaceWire Plug-and-Play address space into APB address space. See section 77.10 for details. This register is read-only in SpaceWire Plug-and-Play interface, while it is writable from the APB address space.

77.14 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x29. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

77.15 Configuration options

Table 1383 shows the configuration options of the core (VHDL generics).

*Table 1383.*Configuration options

Generic	Function	Allowed values	Default
tech	Selects technology for transmitter DDR registers (if output_type=1) and enables a reset of additional registers for ASIC technologies.	0 - NTECH	inferred
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by GRSPW2.	0 - NAHBIRQ-1	0
rmap	Include hardware RMAP target. RMAP CRC logic will also be added. If set to 2 the core will only implement the RMAP target, provide a limited APB interface, enable time code reception and its interrupt.	0 - 2	0
rmapcrc	Enable RMAP CRC logic.	0 - 1	0
ccsdscrc	Enables CCSDS/CCITT CRC-16 and 16-bit ISO-checksum (J.G. Fletcher, ISO 8473-1:1998) logic. For the core to calculate CRC rmap or rmapcrc also need to be set to 1.	0 - 1	0
fifosize1	Sets the number of entries in the 32-bit receiver and transmitter AHB fifos.	4 - 64	32
fifosize2	Sets the number of entries in the 9-bit receiver fifo (N-Char fifo).	16 - 64	64
rxunaligned	Receiver unaligned write support. If set, the receiver can write any number of bytes to any start address without writing any excessive bytes.	0 - 1	0
rmapbufs	Sets the number of buffers to hold RMAP replies.	2 - 8	4
ft	Enable fault-tolerance against SEU errors. The core can optionally be implemented with fault-tolerance against SEU errors in the FIFO memories. The fault-tolerance is enabled through the ft VHDL generic to 1, 2, 4 or 5. Possible options are byte parity protection (ft = 1), TMR registers (ft = 2), SECDED BCH (ft = 4) or technology specific protection (ft = 5).	0 - 2, 4-5	0
scantest	Enables scantest support	0 - 1	0
techfifo	Enables technology specific RAM blocks selected with memtech. When disabled the memtech generic will have no effect.	0 - 1	1
ports	Sets the number of ports	1 - 2	1
dmachan	Sets the number of DMA channels	1 - 4	1
memtech	Selects technology for RAM blocks.	0 - NTECH	DEFMEMTECH
input_type	Select receiver type. 0 = Self clocking (xor), 1 = Interface for Frontgrade UT200SpWPHY01 SpaceWire transceiver, 2 = Single data rate sampling, 3 and 4 = Double data rate sampling, 5 = Self-clocking with external recovery, 6 = Self-clocking with external recovery and DDR register for data.. This generic must be set to the same value as the GRSPW2_PHY generic with the same name.	0 - 6	0

*Table 1383.*Configuration options

Generic	Function	Allowed values	Default
output_type	Select transmitter type. 0 = Single data rate, 1 = Double data rate, 2 = Interface for Frontgrade UT200SpWPHY01 SpaceWire transceiver.	0 - 2	0
rxtx_sameclk	Set to one if the same clock net is connected to both the receiver and transmitter (which means this feature is only applicable when the receiver uses SDR or DDR sampling). This will remove some unnecessary synchronization registers as well as local asynchronous reset nets.	0 - 1	0
netlist	Select pre-synthesized netlist instead of synthesizing from source. When enabled the specific netlist is selected with the tech generic.	0 - 1	0
nodeaddr	When set to 0 - 254: Generic specifies the reset value for the node address (DEFADDR.DEFADDR field). When set to 255: Reset value for the node address is taken from the RMAPNODEADDR input signal.	0 - 255	254
destkey	Sets the reset value for the core's destination key.	0 - 255	0
interruptdist	Enables interrupt distribution support and sets the number of supported interrupts.	0, 1, 2, 4, 8, 16, 32	0
intscalerbits	Sets the number of bits for the interrupt distribution prescaler. 0 = No Interrupt distribution prescaler implemented.	0 - 31	0
intisrtimerbits	Sets the number of bits for the Interrupt distribution ISR timers. 0 = No Interrupt distribution ISR timer implemented.	0 - 31	0
intiatimerbits	Sets the number of bits for the Interrupt distribution INT / ACK timers. 0 = No Interrupt distribution INT / ACK timers implemented.	0 - 31	0
intctimerbits	Sets the number of bits for the Interrupt distribution change timers. 0 = No Interrupt distribution change timers implemented.	0 - 31	0
tickinasync	Determines whether the TICKIN, TICKINRAW, and TIMEIN signals are synchronous or asynchronous to CLK. 0 = synchronous, 1 = asynchronous	0 - 1	0
pnp	Enabled / disables support for SpaceWire Plug-and-Play	0 - 1	0
pnpvendid	Specifies the SpaceWire Plug-and-Play Device Vendor ID	0 - 16#FFFF#	0
pnpprodid	Specifies the SpaceWire Plug-and-Play Device Product ID	0 - 16#FFFF#	0
pnpmajorver	Specifies the device's SpaceWire Plug-and-Play Major Version	0 - 16#FFFF#	0
pnpminorver	Specifies the device's SpaceWire Plug-and-Play Minor Version	0 - 16#FFFF#	0
pnppatch	Specifies the device's SpaceWire Plug-and-Play Patch/ Build Number	0 - 16#FFFF#	0
num_txdesc	Specifies the number of entries in the transmit descriptor table	64, 128, 256, 512	64
num_rxdesc	Specifies the number of entries in the receive descriptor table	128, 256, 512, 1024	128
rstsrctmr	Enables the Triple Module Redundancy for the asynchronous reset nets of the core	0 - 1	0

77.16 Signal descriptions

Table 1384 shows the interface signals of the core (VHDL ports).

Table 1384. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
RXASYNCRST	N/A	Input	Asynchronous reset for ports 0 and 1 of the receiver	Low
RXSYNCRST0	N/A	Input	Synchronous reset for port 0 of the receiver	Low
RXCLK0	N/A	Input	Receiver clock for port 0	-
RXSYNCRST1	N/A	Input	Synchronous reset for port 1 of the receiver	Low
RXCLK1	N/A	Input	Receiver clock for port 1. Unused in one-port configurations.	-
TXSYNCRST	N/A	Input	Synchronous reset for the transmitter	Low
TXCLK	N/A	Input	Transmitter default run-state clock	-
TXCLKN	N/A	Input	Transmitter inverted default run-state clock. Only used in DDR transmitter mode for technologies not supporting local generation of inverted clock.	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
SWNI	D[3:0]	Input	Data input. D[1:0] correspond to port 0, D[3:2] correspond to port 1. In a one-port configuration D[3:2] are unused. Connect to the DO[1:0] output of one GRSP-W2_PHY per implemented port.	-
	DV[3:0]	Input	Data valid input corresponding to SWNI.D[3:0]. Bits 3:2 are unused in one-port configurations. Connect to the DOV[1:0] output of one GRSP-W2_PHY per implemented port.	-
	DCONNECT[3:0]	Input	Disconnect. Bits 3:2 are unused in one-port configurations.	
	DCONNECT2[3:0]	Input	Disconnect. Bits 3:2 are unused in one-port configurations. This is a copy of DCONNECT, as part of the triplication of the combinational logic related to asynchronous reset nets.	
	DCONNECT3[3:0]	Input	Disconnect. Bits 3:2 are unused in one-port configurations. This is a copy of DCONNECT, as part of the triplication of the combinational logic related to asynchronous reset nets.	
	TICKIN	Input	Time counter tick input. Increments internal time-counter and transmits the new value.	High
	TICKINRAW	Input	Raw tick input. Send time-code or interrupt- / interrupt-acknowledge-code from TIMEIN signal. TICKINRAW must be deasserted the same clock cycle as the output signal TICKINDONE is asserted.	High
	TIMEIN[7:0]	Input	The time-code or interrupt- / interrupt-acknowledge-code sent when TICKINRAW is asserted.	

Table 1384. Signal descriptions

Signal name	Field	Type	Function	Active
SWNI	CLKDIV10[7:0]	Input	Reset value for the Clock divisor register.	-
	RMAPEN	Input	Reset value for the CTRL.RE bit (RMAP enable bit in the Control register).	-
	RMAPNODEADDR[7:0]	Input	Reset value for the node address (DEFADDR.DEFADDR field) when the *nodeaddr* VHDL generic = 255. Unused if *nodeaddr* /= 255.	-
	INTPRELOAD[30:0]	Input	Reset value for the Interrupt distribution prescaler reload register	-
	INTTRELOAD[30:0]	Input	Reset value for the Interrupt distribution ISR timer reload register	-
	INTIARELOAD[30:0]	Input	Reset value the Interrupt distribution INT / ACK timer reload register	-
	INTCRELOAD[30:0]	Input	Reset value the Interrupt distribution change timer reload register	-
	IRQTXDEFAULT[5:0]	Input	Reset value for the different transmit interrupt numbers for the interrupt distribution	-
	PNPEN	Input	Reset value for the SpaceWire Plug-and-Play enable bit (CTRL.PE)	-
	PNPUVENDID	Input	Reset value for the SpaceWire Plug-and-Play Unit Vendor ID (PNPUVEND.VEND field)	-
	PNPUPRODID	Input	Reset value for the SpaceWire Plug-and-Play Unit Product ID (PNPUVEND.PROD field)	-
	PNPUSN	Input	Reset value for the SpaceWire Plug-and-Play Unit Serial Number (PNPUSN register)	-

Table 1384. Signal descriptions

Signal name	Field	Type	Function	Active
SWNO	D[3:0]	Output	Data output. The meaning of this signal changes depending on the value of the *output_type* generic. For *output_type*=0 and *output_type*=1, D[0] is the data output for port 0, and D[1] is the data output for port 1 while D[3:2] are unused. For *output_type*=2 (SpaceWire transceiver), D[1:0] is the data output for port 0, and D[3:2] is the data output for port 1.	-
	S[3:0]	Output	Strobe output. The meaning of the bits in this signal depends on the *output_type* generic in the same way as SWNO.D[3:0] described above.	-
	TICKOUT	Output	Tick-out signal that is asserted when a valid time-code or interrupt- / interrupt-acknowledge-code has been received.	High
	TICKOUTRAW	Output	Tick-out that is always asserted when a time-code or interrupt- / interrupt-acknowledge-code is received.	High
	TIMEOUT[7:0]	Output	Contains the received time-code or interrupt- / interrupt-acknowledge-code when TICK-OUTRAW is asserted.	
	TICKINDONE	Output	Asserted when a time-code or interupt- / interrupt-acknowledge-code sent via the TICKIN-RAW signal has been accepted for transmission. TICKINRAW must be deasserted the same clock cycle as TICKINDONE is asserted.	High
	RXDAV	Output	Asserted for one CLK cycle when a character has been received on the SpaceWire link.	High
	RXDATAOUT[8:0]	Output	When RXDAV is asserted, these signals contain the received character.	
	LINKDIS	Output	Asserted when the link is disabled	High
	LOOPBACK	Output	Reflects the value of the loopback bit in the Control register. Can be used to control on-chip loopback for test purposes.	High
	RXRST	Output	Internal reset generated by the transmitter for synchronization purpose between transmitter and both receiver channels. It shall be used to generate the asynchronous and synchronous receiver resets.	Low
	CTRLREGRST	Output	Register reset. It corresponds to the bit 6 of the Control Register (see 77.13.1). It shall be used negated (it is active high) to generate the AMBA and transmitter resets.	High
	RMAPACT	Output	RMAP command processing active.	High
* see GRLIB IP Library User's Manual				

77.17 Signal definitions and reset values

The signals and their reset values are described in table 1385.

*Table 1385.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
spw_clk	Input	Transmitter default run-state clock	Rising edge	-
spw_rxd	Input, LVDS	Data input, positive	High	-
spw_rxdn	Input, LVDS	Data input, negative	Low	-
spw_rxs	Input, LVDS	Strobe input, positive	High	-
spw_rxsn	Input, LVDS	Strobe input, negative	Low	-
spw_txd	Output, LVDS	Data output, positive	High	Logical 0
spw_txdn	Output, LVDS	Data output, negative	Low	Logical 1
spw_txs	Output, LVDS	Strobe output, positive	High	Logical 0
spw_txsn	Output, LVDS	Strobe output, negative	Low	Logical 1

77.18 Timing

The timing waveforms and timing parameters are shown in figure 200 and are defined in table 1386.

The SpaceWire jitter and skew timing waveforms and timing parameters are shown in figure 201 and are defined in table 1387.

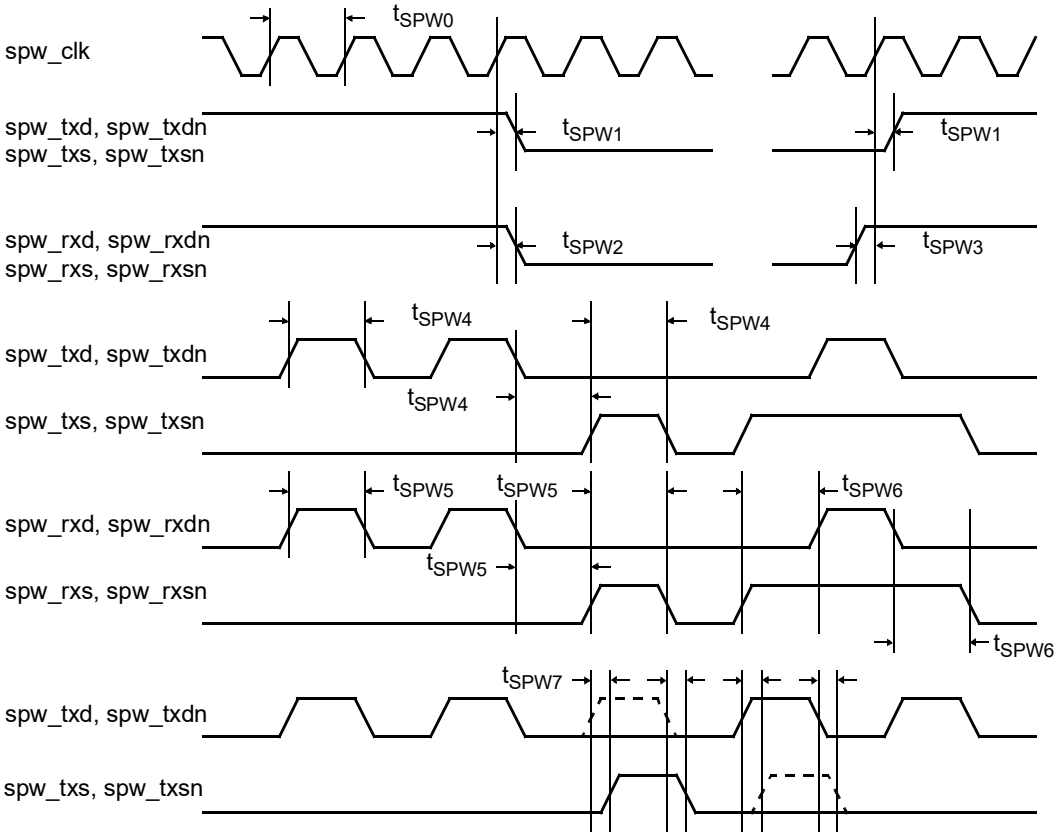


Figure 200. Timing waveforms

Table 1386. Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{SPW0}	transmit clock period	-	TBD	-	ns
t_{SPW1}	clock to output delay	rising *spw_clk* edge	TBD	TBD	ns
t_{SPW2}	input to clock hold	-	-	-	not applicable
t_{SPW3}	input to clock setup	-	-	-	not applicable
t_{SPW4}	output data bit period	-	-		*clk* periods
		-	t_{SPW0} - TBD	t_{SPW0} +TBD	ns
t_{SPW5}	input data bit period	-	TBD	-	ns
t_{SPW6}	data & strobe edge separation	-	TBD	-	ns
t_{SPW7}	data & strobe output skew	-	-	TBD	ns

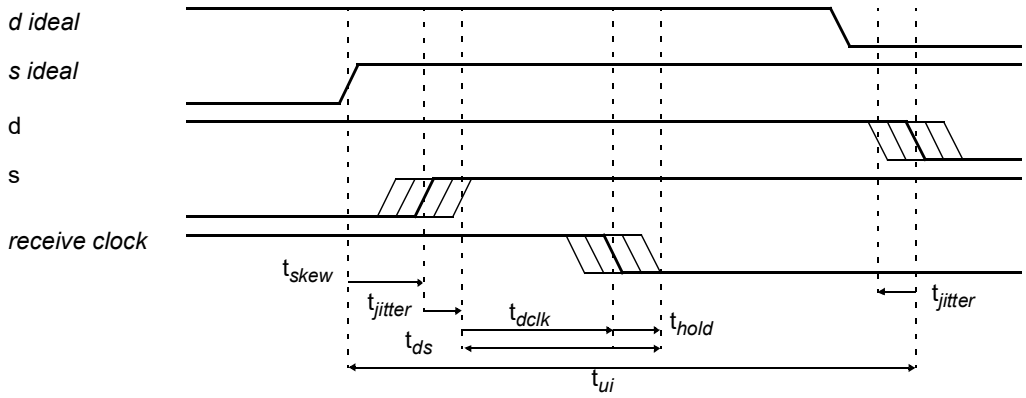


Figure 201. Skew and jitter timing waveforms

*Table 1387.*Skew and jitter timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{skew}	skew between data and strobe	-	-	TBD	ns
t_{jitter}	jitter on data or strobe	-	-	TBD	ns
t_{ds}	minimum separation between data and strobe edges	-	TBD	-	ns
t_{dclk}	delay from edge of data or strobe to the receiver flip-flop	-	-	TBD	ns
t_{hold}	hold timer on receiver flip-flop	-	TBD	-	ns
t_{ui}	unit interval (bit period)	-	TBD	-	ns

77.19 Library dependencies

Table 1388 shows libraries used when instantiating the core (VHDL libraries).

*Table 1388.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	SPACEWIRE	Signals, component	Component and record declarations.

77.20 Instantiation

This example shows how the core can be instantiated.

Normally di, si, do and so should be connected to input and output pads configured with LVDS drivers. How this is done is technology dependent.

The core in the example is configured with non-ft memories of size 32, 32 and 4 entries for AHB FIFOs, N-Char FIFO and RMAP buffers respectively.

The memory technology is inferred which means that the synthesis tool will select the appropriate components.

The hardware RMAP target is enabled which also automatically enables rxunaligned and rmapcrc.

The core can be instantiated directly (grspw2) or using a wrapper (grspwm). In the first case, the top level instance is in charge of generating the resets appropriately, whereas the wrapper may include the reset generation and synchronization if the generic internalrstgen is set to 1 (default value).

Example of direct instantiation, including reset generators:

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.spacewire.all;

entity spacewire_ex is
  port (
    clk           : in  std_ulogic;
    rstn          : in  std_ulogic;

    -- spacewire signals
    spw_rxdp      : in  std_ulogic;
    spw_rxdn      : in  std_ulogic;
    spw_rxsp      : in  std_ulogic;
    spw_rxsn      : in  std_ulogic;
    spw_txdp      : out std_ulogic;
    spw_txdn      : out std_ulogic;
    spw_txsp      : out std_ulogic;
    spw_txsn      : out std_ulogic;

    spw_rxtxclk   : in  std_ulogic;
    spw_rxclkn    : in  std_ulogic
    );
end;

architecture rtl of spacewire_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- Spacewire signals
  signal swni      : grspw_in_type;
  signal swno      : grspw_out_type;
  signal dtmp      : std_ulogic;
  signal stmp      : std_ulogic;
  signal rxclk0    : std_ulogic;
  -- Only 1 port in this design. rxclk1 is unused. Instantiate
  -- two PHY's (grspw2_phy) if two ports are enabled in the receiver
  signal rxclk1    : std_ulogic := '0';

  -- Internal resets
  signal tmp_reset : std_ulogic;
  signal mrst      : std_ulogic;
  signal rxasyncrst: std_ulogic;
  signal rxsyncrst0: std_ulogic;
  signal rxsyncrst1: std_ulogic := '0'; -- Unused in this design
  signal txsyncrst : std_ulogic;

begin

  -- AMBA Components are instantiated here

  spw_phy0 : grspw2_phy
    generic map(
      scantest   => 0,
      tech       => 0,
      input_type => 3)
    port map(
      rstn       => rstn,
      rxclki     => spw_rxtxclk,
      rxclkin    => spw_rxclkn,
      nrxclki    => spw_rxtxclk,
```

```
    di         => dtmp,
    si         => stmp,
    do         => swni.d(1 downto 0),
    dov        => swni.dv(1 downto 0),
    dconnect   => swni.dconnect(1 downto 0),
    dconnect2  => swni.dconnect2(1 downto 0),
    dconnect3  => swni.dconnect3(1 downto 0),
    rxclko     => rxclk0);

-- Internal reset generators (TX, RX0 and RX1 clock domains)

-- The AMBA and TX resets take into account the bit 6 of the Control
-- Register (software reset)
tmp_reset <= rstn and not swno.ctrlregrst;

-- CLK domain (synchronous reset)
AMBA_rst : rstgen
port map (tmp_reset, clk, vcc, mrst, open);

-- TX domain (synchronous reset)
txrst : rstgen
port map (tmp_reset, spw_rxtxclk, vcc, txsyncrst, open);

-- RX domain (asynchronous reset)
rxasyncrst <= swno.rxrst;

-- RX domain (synchronous reset)
rxsyncrst0 : rstgen
port map (swno.rxrst, rxclk0, vcc, rxsyncrst0, open);
-- RXCLK1 is unused. Uncomment if two ports are enabled
--rxsyncrst1 : rstgen
--port map (swno.rxrst, rxclk1, vcc, rxsyncrst1, open);

sw0 : grspw2
generic map(
  tech         => 0,
  hindex       => 0,
  pindex       => 10,
  paddr        => 10,
  pirq         => 10,
  ports        => 1,
  dmachan      => 1,
  rmap         => 1,
  rmapcrc      => 1,
  fifosize1    => 32,
  fifosize2    => 32,
  rxunaligned  => 1,
  rmapbufs     => 4,
  output_type  => 1,
  input_type   => 3,
  rxtx_sameclk => 1)
port map(mrst, clk, rxasyncrst, rxsyncrst0, rxclk0, rxsyncrst1, rxclk1,
     txsyncrst, spw_rxtxclk, spw_rxtxclk, ahbmi, ahbmo(0), apbi, apbo(10),
     swni, swno);

swni.tickin <= '0'; swni.rmapen <= '1';
swni.clkdiv10 <= conv_std_logic_vector(SPW_TX_FREQ_KHZ/10000-1, 8);

spw_rxd_pad : inpad_ds generic map (padtech, lvds, x25v)
  port map (spw_rxdp, spw_rxdn, dtmp);
spw_rxs_pad : inpad_ds generic map (padtech, lvds, x25v)
  port map (spw_rxsp, spw_rxsn, stmp);
spw_txd_pad : outpad_ds generic map (padtech, lvds, x25v)
  port map (spw_txdp, spw_txdn, swno.d(0), gnd(0));
spw_txs_pad : outpad_ds generic map (padtech, lvds, x25v)
  port map (spw_txsp, spw_txsn, swno.s(0), gnd(0));
...
```

Example using the wrapper and the internal reset generators enabled. Replace the reset generators and the GRSPW2 core with the following wrapper:

```
sw0 : grspwm
generic map(
     tech           => fabtech,
     hindex         => CFG_NCPU+CFG_AHB_UART+CFG_AHB_JTAG+CFG_GRETH+i,
     pindex         => 13+i,
     paddr          => 13+i,
     pirq           => 8+i,
     sysfreq        => CPU_FREQ,
     nsync          => 1,
     rmap           => CFG_SPW_RMAP,
     rmapcrc        => CFG_SPW_RMAPCRC,
     fifosize1      => CFG_SPW_AHBFIFO,
     fifosize2      => CFG_SPW_RXFIFO,
     rxclkbuftype   => 1,
     memtech        => memtech,
     rmapbufs       => CFG_SPW_RMAPBUF,
     ft             => CFG_SPW_FT,
     ports          => CFG_SPW_PORTS,
     dmachan        => CFG_SPW_DMACHAN,
     netlist        => CFG_SPW_NETLIST,
     spwcore        => 2, -- GRSPW2 is instantiated
     input_type     => CFG_SPW_INPUT,
     output_type    => CFG_SPW_OUTPUT,
     rxtx_sameclk   => CFG_SPW_RTSAME,
     rxunaligned    => CFG_SPW_RXUNAL,
     internalrstgen => 1) -- The wrapper will instantiate internally the reset generators
port map( rst => rstn, clk => clkm, rxasyncrst => gnd,
          rxsyncrst0 => gnd, rxclk0 => spw_rxclk0,
          rxsyncrst1 => gnd, rxclk1 => spw_rxclk1,
          txsyncrst => gnd, txclk => spw_txclk, txclkn => spw_txclk,
          ahbmi => ahbmi, ahbmo => ahbmo(CFG_NCPU+CFG_AHB_UART+CFG_AHB_JTAG+CFG_GRETH+i),
          apbi => apbi, apbo => apbo(13),
          swni => spwi, swno => spwo);
```

It is important to note that the input resets are tied to GND in the second example, as these signals are internally generated in the wrapper and therefore nor generated neither used in the top level entity.

77.21 Constraints

This section contains example constraints for GRSPW2.

0. Define a clock called 'spw_clk'
1. spw_peri = SpaceWire Maximum clock frequency
2. tech_lib_setup = Setup timing for FlipFlop used in technology
3. tech_lib_hold = Hold timing for FlipFlop used in technology

```
### SpaceWire (LVDS)
set spw_rx_rise_max  [expr $spw_peri / 2 - $tech_lib_setup]
set spw_rx_rise_min  [expr $tech_lib_hold ]
set spw_rx_fall_max  [expr $spw_peri / 2 - $tech_lib_setup]
set spw_rx_fall_min  [expr $tech_lib_hold ]

set spw_tx_max [expr      $tech_lib_setup ]
set spw_tx_min [expr -1 * $tech_lib_hold  ]
```

```
for { set port 0 } { $port < 2 } { incr port } {
   # Inputs
   set_input_delay -max -clock [get_clocks spw_clk]  $spw_rx_rise_max [get_ports
   lvds_rxp[[expr $port + 1]]] -add
   set_input_delay -min -clock [get_clocks spw_clk]  $spw_rx_rise_min [get_ports
   lvds_rxp[[expr $port + 1]]] -add
   set_input_delay -max -clock [get_clocks spw_clk]  $spw_rx_fall_max [get_ports
   lvds_rxp[[expr $port + 1]]] -add -clock_fall
   set_input_delay -min -clock [get_clocks spw_clk]  $spw_rx_fall_min [get_ports
   lvds_rxp[[expr $port + 1]]] -add -clock_fall

   set_input_delay -max -clock [get_clocks spw_clk]  $spw_rx_rise_max [get_ports
   lvds_rxn[[expr $port + 1]]] -add
   set_input_delay -min -clock [get_clocks spw_clk]  $spw_rx_rise_min [get_ports
   lvds_rxn[[expr $port + 1]]] -add
   set_input_delay -max -clock [get_clocks spw_clk]  $spw_rx_fall_max [get_ports
   lvds_rxn[[expr $port + 1]]] -add -clock_fall
   set_input_delay -min -clock [get_clocks spw_clk]  $spw_rx_fall_min [get_ports
   lvds_rxn[[expr $port + 1]]] -add -clock_fall
}

for { set port 0 } { $port < 2 } { incr port } {
   set_output_delay -clock [get_clocks spw_clk] -max $spw_tx_max [get_ports lvds_txp[[expr
   $port + 1]]] -add
   set_output_delay -clock [get_clocks spw_clk] -min $spw_tx_min [get_ports lvds_txp[[expr
   $port + 1]]] -add_delay

   set_output_delay -clock [get_clocks spw_clk] -max $spw_tx_max [get_ports lvds_txn[[expr
   $port + 1]]] -add
   set_output_delay -clock [get_clocks spw_clk] -min $spw_tx_min [get_ports lvds_txn[[expr
   $port + 1]]] -add_delay
}
```

77.22 API

A simple Application Programming Interface (API) is provided together with the GRSPW2. The API is located in $(GRLIB)/software/spw. The files are rmapapi.c, spwapi.c, rmapapi.h, spwapi.h. The spwapi.h file contains the declarations of the functions used for configuring the GRSPW2 and transferring data. The corresponding definitions are located in spwapi.c. The rmapapi is structured in the same manner and contains a function for building RMAP packets.

These functions could be used as a simple starting point for developing drivers for the GRSPW2. The different functions are described in this section.

77.22.1 GRSPW2 Basic API

The basic GRSPW2 API is based on a struct spwvars which stores all the information for a single GRSPW2 core. The information includes its address on the AMBA bus as well as SpaceWire parameters such as node address and clock divisor. A pointer to this struct is used as a input parameter to all the functions. If several cores are used, a separate struct for each core is created and used when the specific core is accessed.

Table 1389. The spwvars struct

Field	Description	Allowed range
regs	Pointer to the GRSPW2	-
nospill	The nospill value used for the core.	0 - 1
rmap	Indicates whether the core is configured with RMAP. Set by spw_init.	0 - 1
rxunaligned	Indicates whether the core is configured with rxunaligned support. Set by spw_init.	0 - 1
rmapcrc	Indicates whether the core is configured with RMAPCRC support. Set by spw_init.	0 - 1
clkdiv	The clock divisor value used for the core.	0 - 255
nodeaddr	The node address value used for the core.	0 - 255
destkey	The destination key value used for the core.	0 - 255
rxmaxlen	The Receiver maximum length value used for the core.	0 - 33554431
rxpnt	Pointer to the next receiver descriptor.	0 - 127
rxchkpnt	Pointer to the next receiver descriptor that will be polled.	0 - 127
txpnt	Pointer to the next transmitter descriptor.	0 - 63
txchkpnt	Pointer to the next transmitter descriptor that will be polled.	0 - 63
timetxen	The timetxen value used for this core.	0 - 1
timerxen	The timerxen value used for this core.	0 - 1
txd	Pointer to the transmitter descriptor table.	-
rxd	Pointer to the receiver descriptor table	-

The following functions are available in the basic API:

```
int spw_setparam(int nodeaddr, int clkdiv, int destkey, int nospill, int timetxen, int
timerxen, int rxmaxlen, int spwadr, struct spwvars *spw);
```

Used for setting the different parameters in the spwvars struct. Should always be run first after creating a spwvars struct. This function only initializes the struct. Does not write anything to the SpaceWire core.

Table 1390. Return values for spw_setparam

Value	Description
0	The function completed successfully
1	One or more of the parameters had an illegal value

*Table 1391.*Parameters for spw_setparam

Parameter	Description	Allowed range
nodeaddr	Sets the node address value of the struct spw passed to the function.	0-255
clkdiv	Sets the clock divisor value of the struct spw passed to the function.	0-255
destkey	Sets the destination key of the struct spw passed to the function.	0-255
nospill	Sets the nospill value of the struct spw passed to the function.	0 - 1
timetxen	Sets the timetxen value of the struct spw passed to the function.	0 - 1
timerxen	Sets the timerxen value of the struct spw passed to the function.	0 - 1
rxmaxlen	Sets the receiver maximum length field of the struct spw passed to the function.	0 - 2^{25}-1
spwadr	Sets the address to the GRSPW2 core which will be associated with the struct passed to the function.	0 - 2^{32}-1

```
int spw_init(struct spwvars *spw);
```

Initializes the GRSPW2 core located at the address set in the struct spw. Sets the following registers: node address, destination key, clock divisor, receiver maximum length, transmitter descriptor table address, receiver descriptor table address, ctrl and dmactrl. All bits are set to the values found in the spwvars struct. If a register bit is not present in the struct it will be set to zero. The descriptor tables are allocated to an aligned area using malloc. The status register is cleared and lastly the link interface is enabled. The run state frequency will be set according to the value in clkdiv.

*Table 1392.*Return values for spw_init

Value	Description
0	The function completed successfully
1	One or more of the parameters could not be set correctly or the link failed to initialize.

*Table 1393.*Parameters for spw_init

Parameter	Description	Allowed range
spw	The spwvars struct associated with the GRSPW2 core that should be initialized.	-

```
int set_txdesc(int pnt, struct spwvars *spw);
```

Sets a new address to the transmitter descriptor table address register. Should only be used when no transmission is active. Also resets the pointers for spw_tx and spw_checktx (Explained in the section for those functions).

*Table 1394.*Return values for spw_txdesc

Value	Description
0	The function completed successfully
1	The new address could not be written correctly

*Table 1395.*Parameters for spw_txdesc

Parameter	Description	Allowed range
pnt	The new address to the descriptor table area	0 - 2^{32}-1
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be configured	-

```
int set_rxdesc(int pnt, struct spwvars *spw);
```

Sets a new address to the Receiver descriptor table address register. Should only be used when no transmission is active. Also resets the pointers for spw_rx and spw_checkrx (Explained in the section for those functions).

*Table 1396.*Return values for spw_rxdesc

Value	Description
0	The function completed successfully
1	The new address could not be written correctly

*Table 1397.*Parameters for spw_rxdesc

Parameter	Description	Allowed range
pnt	The new address to the descriptor table area	0 - 2^{32}-1
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be configured	-

```
void spw_disable(struct spwvars *spw);
```

Disables the GRSPW2 core (the link disable bit is set to ‘1’).

*Table 1398.*Parameters for spw_disable

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be configured	-

```
void spw_enable(struct spwvars *spw);
```

Enables the GRSPW2 core (the link disable bit is set to ‘0’).

*Table 1399.*Parameters for spw_enable

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be configured	-

```
void spw_start(struct spwvars *spw);
```

Starts the GRSPW2 core (the link start bit is set to ‘1’).

*Table 1400.*Parameters for spw_start

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be configured	-

```
void spw_stop(struct spwvars *spw);
```

Stops the GRSPW2 core (the link start bit is set to ‘0’).

*Table 1401.*Parameters for spw_start

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be configured	-

```
int spw_setclockdiv(struct spwvars *spw);
```

Sets the clock divisor register with the clock divisor value stored in the spwvars struct.

*Table 1402.*Return values for spw_setclockdiv

Value	Description
0	The function completed successfully
1	The new clock divisor value is illegal.

*Table 1403.*Parameters for spw_setclockdiv

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be configured	-

```
int spw_set_nodeadr(struct spwvars *spw);
```

Sets the node address register with the node address value stored in the spwvars struct.

*Table 1404.*Return values for spw_set_nodeadr

Value	Description
0	The function completed successfully
1	The new node address value is illegal.

*Table 1405.*Parameters for spw_set_nodeadr

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be configured	-

```
int spw_set_rxmaxlength(struct spwvars *spw);
```

Sets the Receiver maximum length register with the rxmaxlen value stored in the spwvars struct.

*Table 1406.*Return values for spw_set_rxmaxlength

Value	Description
0	The function completed successfully
1	The new node address value is illegal.

*Table 1407.*Parameters for spw_set_rxmaxlength

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be configured	-

```
int spw_tx(int crc, int skipcrcsize, int hsize, char *hbuf, int dsize, char *dbuf, struct
spwvars *spw);
```

Transmits a packet. Separate header and data buffers can be used. If CRC logic is available the GSPW inserts RMAP CRC values after the header and data fields if crc is set to one. This function only sets a descriptor and initiates the transmission. Spw_checktx must be used to check if the packet has been transmitted. A pointer into the descriptor table is stored in the spwvars struct to keep track of the next location to use. It is incremented each time the function returns 0.

*Table 1408.*Return values for spw_tx

Value	Description
0	The function completed successfully
1	There are no free transmit descriptors currently available
2	There was illegal parameters passed to the function

*Table 1409.*Parameters for spw_tx

Parameter	Description	Allowed range
crc	Set to one to append RMAP CRC after the header and data fields. Only available if hardware CRC is available in the core.	0 - 1
skipcrcsize	The number of bytes in the beginning of a packet that should not be included in the CRC calculation	0 - 15
hsize	The size of the header in bytes	0 - 255
hbuf	Pointer to the header data	-
dsize	The size of the data field in bytes	0 - 2^{24}-1
dbuf	Pointer to the data area.	-
spw	Pointer to the spwvars struct associated with GRSPW2 core that should transmit the packet	-

```
int spw_rx(char *buf, struct spwvars *spw);
```

Enables a descriptor for reception. The packet will be stored to buf. Spw_checkrx must be used to check if a packet has been received. A pointer in the spwvars struct is used to keep track of the next location to use in the descriptor table. It is incremented each time the function returns 0.

*Table 1410.*Return values for spw_rx

Value	Description
0	The function completed successfully
1	There are no free receive descriptors currently available

*Table 1411.*Parameters for spw_rx

Parameter	Description	Allowed range
buf	Pointer to the data area.	-
spw	Pointer to the spwvars struct associated with GRSPW2 core that should receive the packet	-

```
int spw_checkrx(int *size, struct rxstatus *rxs, struct spwvars *spw);
```

Checks if a packet has been received. When a packet has been received the size in bytes will be stored in the size parameter and status is found in the rxs struct. A pointer in the spwvars struct is used to keep track of the location in the descriptor table to poll. It is incremented each time the function returns nonzero.

*Table 1412.*Return values for spw_checkrx

Value	Description
0	No packet has been received
1	A packet has been received

*Table 1413.*Parameters for spw_checkrx

Parameter	Description	Allowed range
size	When the function returns 1 this variable holds the number of bytes received	-
rxs	When the function returns 1 this variable holds status information	-
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be polled	-

Table 1414. The rxstatus struct

Field	Description	Allowed range
truncated	Packet was truncated	0 - 1
dcrcerr	Data CRC error bit was set. Only indicates an error if the packet received was an RMAP packet.	0 - 1
hcrcerr	Header CRC error bit was se.t. Only indicates an error if the packet received was an RMAP packet.	0 - 1
eep	Packet was terminated with EEP	0 - 1

```
int spw_checktx(struct spwvars *spw);
```

Checks if a packet has been transmitted. A pointer is used to keep track of the location in the descriptor table to poll. It is incremented each time the function returns nonzero.

Table 1415. Return values for spw_checktx

Value	Description
0	No packet has been transmitted
1	A packet has been correctly transmitted
2	A packet has been incorrectly transmitted

Table 1416. Parameters for spw_checktx

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be polled	-

```
void send_time(struct spwvars *spw);
```

Sends a new time-code. Increments the time-counter in the GRSPW2 and transmits the value.

Table 1417. Parameters for send time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be polled	-

```
int check_time(struct spwvars *spw);
```

Check if a new time-code has been received.

Table 1418. Return values for check_time

Value	Description
0	No time-code has been received
1	A new time-code has been received

Table 1419. Parameters for check_time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be polled	-

```
int get_time(struct spwvars *spw);
```

Get the current time counter value.

*Table 1420.*Return values for get_time

Value	Description
0 - 63	Returns the current time counter value

*Table 1421.*Parameters for get_time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be polled	-

```
void spw_reset(struct spwvars *spw);
```

Resets the GRSPW2.

*Table 1422.*Parameters for spw_reset

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be reset	-

```
void spw_rmapen(struct spwvars *spw);
```

Enables hardware RMAP. Has no effect if the RMAP command handler is not available in GRSPW2.

*Table 1423.*Parameters for spw_rmapen

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be set	-

```
void spw_rmapdis(struct spwvars *spw);
```

Disables hardware RMAP. Has no effect if the RMAP command handler is not available in GRSPW2

*Table 1424.*Parameters for spw_rmapdis

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be set	-

```
int spw_setdestkey(struct spwvars *spw);
```

Set the destination key of the GRSPW2. Has no effect if the RMAP command handler is not available. The value from the spwvars struct is used.

*Table 1425.*Return values for spw_setdestkey

Value	Description
0	The function completed successfully
1	The destination key parameter in the spwvars struct contains an illegal value

*Table 1426.*Parameters for spw_setdestkey

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be set.	-

77.22.2 GRSPW2 RMAP API

The RMAP API contains only one function which is used for building RMAP headers.

```
int build_rmap_hdr(struct rmap_pkt *pkt, char *hdr, int *size);
```

Builds an RMAP header to the buffer pointed to by hdr. The header data is taken from the rmap_pkt struct.

*Table 1427.*Return values for build_rmap_hdr

Value	Description
0	The function completed successfully
1	One or more of the parameters contained illegal values

*Table 1428.*Parameters for build_rmap_hdr

Parameter	Description	Allowed range
pkt	Pointer to an rmap_pkt struct which contains the data from which the header should be built	
hdr	Pointer to the buffer where the header will be built	
spw	Pointer to the spwvars struct associated with GRSPW2 core that should be set	-

*Table 1429.*rmap_pkt struct fields

Field	Description	Allowed Range
type	Selects the type of packet to build.	writecmd, readcmd, rmwcmd, writerep, readrep, rmwrep
verify	Selects whether the data should be verified before writing	yes, no
ack	Selects whether an acknowledge should be sent	yes, no
incr	Selects whether the address should be incremented or not	yes, no
destaddr	Sets the destination address	0 - 255
destkey	Sets the destination key	0 - 255
srcaddr	Sets the source address	0 - 255
tid	Sets the transaction identifier field	0 - 65535
addr	Sets the address of the operation to be performed. The extended address field is currently always set to 0.	0 - 2^{32}-1
len	The number of bytes to be writte, read or read-modify-written	0 - 2^{24}-1
status	Sets the status field	0 - 11
dstspalen	Number of source path address bytes to insert before the destination address	0 - 228
dstspa	Pointer to memory holding the destination path address bytes	-
srcspalen	Number of source path address bytes to insert in a command. For a reply these bytes are placed before the return address	0 - 12
srcspa	Pointer to memory holding the source path address bytes	-

78 GRSPW2_GEN - GRSPW2 wrapper with Std_Logic interface

78.1 Overview

The GRSPW2_GEN wrapper provides an interface to the GRSPW2 core only using Std_Logic signals instead of GRLIB records. The GRLIB AMBA plug and play extensions have also been removed. This document describes the signal interface and how they map to the signal names in GRLIB. For a user manual on the core function please refer to the GRSPW2 section. An example instantiation of the core can be found at the end of this document.

78.2 Signal descriptions

Table 1384 shows the interface signals of the core (VHDL ports).

Table 1430. Signal descriptions

Signal name	Type	Function	Active	GRLIB signal name
RST	Input	Reset	Low	RST
CLK	Input	System (AMBA) Clock.	-	CLK
RXASYNCRST	Input	Receiver asynchronous reset for both ports 0 and 1	Low	RXASYNCRST
RXSYNCRST0	Input	Receiver synchronous reset for port 0	Low	RXSYNCRST0
RXCLK0	Input	Receiver clock for port 0.	-	RXCLK0
RXSYNCRST1	Input	Receiver synchronous reset for port 1	Low	RXSYNCRST1
RXCLK1	Input	Receiver clock for port 1. Unused if the VHDL generic ports is 2.	-	RXCLK1
TXSYNCRST	Input	Transmitter synchronous reset	Low	TXSYNCRST
TXCLK	Input	Transmitter clock	-	TXCLK
TXCLKN	Input	Transmitter inverted clock. Only used in DDR transmitter mode for technologies not supporting local generation of inverted clock.	-	TXCLKN
HGRANT	Input	See AMBA manual.	-	AHBMI.HGRANT
HREADY	Input	See AMBA manual.	-	AHBMI.HREADY
HRESP[1:0]	Input	See AMBA manual.	-	AHBMI.HRESP
HRDATA[31:0]	Input	See AMBA manual.	-	AHBMI.HRDATA
ENDIAN	Input	Sideband signal selecting the endianness of the host system	-	AHBMI.ENDIAN
HBUSREQ	Output	See AMBA manual.	-	AHBMO.HBUSREQ
HLOCK	Output	See AMBA manual.	-	AHBMO.HLOCK
HTRANS[1:0]	Output	See AMBA manual.	-	AHBMO.HTRANS
HADDR[31:0]	Output	See AMBA manual.	-	AHBMO.HADDR
HWRITE	Output	See AMBA manual.	-	AHBMO.HWRITE
HSIZE[2:0]	Output	See AMBA manual.	-	AHBMO.HSIZE
HBURST[2:0]	Output	See AMBA manual.	-	AHBMO.HBURST
HPROT[3:0]	Output	See AMBA manual.	-	AHBMO.HPROT
HWDATA[31:0]	Output	See AMBA manual.	-	AHBMO.HWDATA

Table 1430. Signal descriptions

Signal name	Type	Function	Active	GRLIB signal name
PSEL	Input	See AMBA manual.	-	APBI.PSEL
PENABLE	Input	See AMBA manual.	-	APBI.PENABLE
PADDR[31:0]	Input	See AMBA manual.	-	APBI.PADDR
PWRITE	Input	See AMBA manual.	-	APBI.PWRITE
PWDATA[31:0]	Input	See AMBA manual.	-	APBI.PWDATA
PRDATA[31:0]	Output	See AMBA manual.	-	APBO.PRDATA
D[3:0]	Input	SpaceWire Data input	-	SWNI.D
DV[3:0]	Input	SpaceWire Data valid.	High	SWNI.DV
DCONNECT[3:0]	Input	SpaceWire Disconnect.	-	SWNI.DCONNECT
DCONNECT2[3:0]	Input	SpaceWire Disconnect. Copy of DCONNECT (triplication of the asynchronous resets logic)	-	SWNI.DCONNECT2
DCONNECT3[3:0]	Input	SpaceWire Disconnect. Copy of DCONNECT (triplication of the asynchronous resets logic)	-	SWNI.DCONNECT3
DO[3:0]	Output	SpaceWire Data output.	-	SWNO.D
SO[3:0]	Output	SpaceWire Strobe output.	-	SWNO.S
TICKIN	Input	Time counter tick input. Increments internal time-counter and transmits the new value.	High	SWNI.TICKIN
TICKINRAW	Input	Raw tick input. Send time-code from timein input.	High	SWNI.TICKINRAW
TIMEIN[7:0]	Input	Raw tick input. Send time-code from timein input.	-	SWNI.TIMEIN
TICKINDONE	Output	Asserted when a time-code has been accepted for transmission when tickinraw is asserted. Tickinraw must be deasserted the same clock cycle as tickindone is asserted.	High	SWNO.TICKINDONE
TICKOUT	Output	Time counter tick output. Asserted when a valid time-code has been received	High	SWNO.TICKOUT
TICKOUTRAW	Output	Tick out which is always set when a time-code is received.	High	SWNO.TICKOUTRAW
TIMEOUT[7:0]	Output	Contains the received time-code when tickinraw is asserted.	-	SWNO.TIMEOUT
IRQ	Output	Common interrupt line for the core. Asserted one clock cycle for each interrupt.	High	N/A
CLKDIV10[7:0]	Input	Clock divisor value used during initialization and as reset value for the clock divisor register	-	SWNI.CLKDIV10
LINKDIS	Output	Asserted when the link is disabled	High	SWNO.LINKDIS
TESTRST	Output	Scan test reset	Low	-
TESTEN	Input	Scan test enable	High	-
RMAPEN	Input	Reset value for the rmapen control register bit	High	SWNI.RMAPEN
RXDAV	Output	Asserted each cycle a character has been received on the SpaceWire link.	High	SWNO.RXDAV
RXDATAOUT[8:0]	Output	Contains the received character when rxdav is asserted	-	SWNO.RXDATAOUT

Table 1430. Signal descriptions

Signal name	Type	Function	Active	GRLIB signal name
CTRLREGRST	Output	Register reset. It corresponds to the bit 6 of the Control Register (see 77.13.1). It shall be used negated (it is active high) to generate the AMBA and transmitter resets.	High	SWNO.CTRLREGRST
RXRST	Output	Internal reset generated by the transmitter for synchronization purpose between transmitter and both receiver channels. It shall be used to generate the asynchronous and synchronous receiver resets.	Low	SWNO.RXRST

78.3 Instantiation

This example shows how the core can be instantiated. The reset generators have been instantiated according to the implementation section in GRSPW2 (refer to 77.11.1 for further information).

Normally di, si, do and so should be connected to input and output pads configured with LVDS drivers. How this is done is technology dependent. Instead the single ended inputs and outputs are connected directly to the external interface. The example uses only one port and the output is in SDR mode which means parts of the SpaceWire input and output vectors are unused.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.spacewire.all;

entity spacewire_ex is
  port (
    clk           : in  std_ulogic;
    rstn          : in  std_ulogic;
    txclk         : in  std_ulogic;
    --ahb mst in
    hgrant        : in  std_ulogic;
    hready        : in  std_ulogic;
    hresp         : in  std_logic_vector(1 downto 0);
    hrdata        : in  std_logic_vector(31 downto 0);
    --ahb mst out
    hbusreq       : out  std_ulogic;
    hlock         : out  std_ulogic;
    htrans        : out  std_logic_vector(1 downto 0);
    haddr         : out  std_logic_vector(31 downto 0);
    hwrite        : out  std_ulogic;
    hsize         : out  std_logic_vector(2 downto 0);
    hburst        : out  std_logic_vector(2 downto 0);
    hprot         : out  std_logic_vector(3 downto 0);
    hwdata        : out  std_logic_vector(31 downto 0);
    --apb slv in
    psel  : in   std_ulogic;
    penable  : in   std_ulogic;
    paddr  : in   std_logic_vector(31 downto 0);
    pwrite  : in   std_ulogic;
    pwdata  : in   std_logic_vector(31 downto 0);
    --apb slv out
    prdata  : out  std_logic_vector(31 downto 0);
    --spw in
    di            : in   std_ulogic;
    si            : in   std_ulogic;
    --spw out
    do            : out  std_ulogic;
    so            : out  std_ulogic;
```

```
    --time iface
    tickin       : in   std_ulogic;
    tickout      : out  std_ulogic;
    --misc
    irq          : out  std_logic;
    clkdiv10     : in   std_logic_vector(7 downto 0);
    rmapen       : in   std_ulogic;
    );
end;

architecture rtl of spacewire_ex is
  signal rxclko    : std_ulogic;
  signal d         : std_logic_vector(3 downto 0);
  signal do_int    : std_logic_vector(3 downto 0);
  signal so_int    : std_logic_vector(3 downto 0);
  signal dv        : std_logic_vector(3 downto 0);
  signal dconnect  : std_logic_vector(3 downto 0);
  signal dconnect2 : std_logic_vector(3 downto 0);
  signal dconnect3 : std_logic_vector(3 downto 0);
  signal timein    : std_logic_vector(7 downto 0);
  signal tickinraw : std_ulogic;
  signal testen    : std_ulogic;
  signal testrst   : std_ulogic;

  -- Internal resets
  signal tmp_reset : std_ulogic;
  signal mrst      : std_ulogic;
  signal rxasyncrst: std_ulogic;
  signal rxsyncrst0: std_ulogic;
  signal rxsyncrst1: std_ulogic;
  signal txsyncrst : std_ulogic;

begin

  do <= do_int(0);
  so <= so_int(0);
  tickinraw <= '0';
  testrst <= '0';
  testen  <= '0';

  spw_phy0 : grspw2_phy
    generic map(
      scantest   => 0,
      tech       => 0,
      input_type => 3)
    port map(
      rstn       => rstn,
      rxclki     => rxclk0,
      rxclkin    => rxclk0,
      nrxclki    => rxclk0,
      di         => di,
      si         => si,
      do         => d(1 downto 0),
      dov        => dv(1 downto 0),
      dconnect   => dconnect(1 downto 0),
      dconnect2  => dconnect2(1 downto 0),
      dconnect3  => dconnect3(1 downto 0),
      rxclko     => rxclko);

  -- Internal reset generators (TX, RX0 and RX1 clock domains)

  -- The AMBA and TX resets take into account the bit 6 of the Control
  -- Register (software reset)
  tmp_reset <= rstn and not ctrlregrst;

  -- CLK domain (synchronous reset)
  AMBA_rst : rstgen
  port map (tmp_reset, clk, vcc, mrst, open);

  -- TX domain (synchronous reset)
  txrst : rstgen
```

```
port map (tmp_reset, spw_rxtxclk, vcc, txsyncrst, open);

-- RX domain (asynchronous reset)
rxasyncrst <= rxrst;

-- RX domain (synchronous reset)
rxsyncrst0 : rstgen
port map (rxrst, rxclk0, vcc, rxsyncrst0, open);

rxsyncrst1 : rstgen
port map (rxrst, rxclk1, vcc, rxsyncrst1, open);

sw0 : grspw2_gen
generic map(
  tech         => 0,
  ports        => 1,
  dmachan      => 1,
  rmap         => 0,
  rmapcrc      => 1,
  fifosize1    => 32,
  fifosize2    => 32,
  rxunaligned  => 1,
  rmapbufs     => 4,
  output_type  => 1,
  input_type   => 3,
  rxtx_sameclk => 1)
port map(
  rst          => mrst,
  clk          => clk,
  rxasyncrst   => rxasyncrst,
  rxsyncrst0   => rxsyncrst0,
  rxclk0       => rxclk0,
  rxsyncrst1   => rxsyncrst1,
  rxclk1       => rxclk1,
  txsyncrst    => txsyncrst,
  txclk        => txclk,
  txclkn       => txclkn,
  --ahb mst in
  hgrant       => hgrant,
  hready       => hready,
  hresp        => hresp,
  hrdata       => hrdata,
  --ahb mst out
  hbusreq      => hbusreq,
  hlock        => hlock,
  htrans       => htrans,
  haddr        => haddr,
  hwrite       => hwrite,
  hsize        => hsize,
  hburst       => hburst,
  hprot        => hprot,
  hwdata       => hwdata,
  --apb slv in
  psel => psel,
  penable => penable,
  paddr => paddr,
  pwrite => pwrite,
  pwdata => pwdata,
  --apb slv out
  prdata => prdata,
  --spw in
  d            => d,
  dv           => dv,
  dconnect     => dconnect,
  dconnect2    => dconnect2,
  dconnect3    => dconnect3,
  --spw out
  do           => do_int,
  so           => so_int,
  --time iface
  tickin       => tickin,
```

```
tickinraw    => tickinraw,
timein       => timein,
tickindone   => open,
tickout      => tickout,
tickoutraw   => open,
timeout      => open,
--irq
irq          => irq,
--misc
clkdiv10     => clkdiv10,
linkdis      => open,
testrst      => testrst,
testen       => testen,
--rmapen
rmapen       => rmapen,
--parallel rx data out
rxdav        => open,
rxdataout    => open,
-- Reset interconnection
ctrlregrst   => ctrlregrst,
rxrst        => rxrst);
```

79 GRSPW2_PHY - GRSPW2 Receiver Physical Interface

79.1 Overview

The GRSPW2_PHY provides a common interface for the receiver modules in GRSPW2, GRSPWROUTER, and GRSPW_CODEC to the actual data recovery circuit. The data can be recovered in four different ways: Self-clocking (xor on data and strobe), Single Data Rate (SDR) sampling, Double Data Rate (DDR) sampling and from an SpaceWire transceiver. The GRSPW2_PHY presents the data with a data, and a data-valid signal, as well as outputs the clock used by the receiver modules. One GRSPW2_PHY core is needed for each SpaceWire link (data and strobe pair).

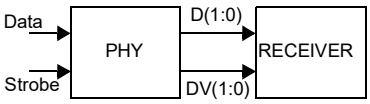


Figure 202. Data connection between the GRSPW2_PHY and the GRSPW2 / GRSPWROUTER / GRSPW_CODEC receiver.

79.2 Operation

As mentioned above the core supports four different input schemes, configurable through the VHDL generic input_type. The GRSPW2, GRSPWROUTER and GRSPW_CODEC cores also each have a generic named input_type, and it is important that it is set to the same value as for GRSPW2_PHY. The four different input schemes are explained in more detail in the following subsections.

79.2.1 Self-clocking (input_type = 0)

In self-clocking mode the receiver clock is recovered from the SpaceWire data and strobe by using an xor gate. The recovered clock is then used to clock the registers to which the data is stored. The recovered clock is coupled to the GRSPW2_PHY output signal RXCLKO, which should be connected to the receiver clock input of GRSPW2 / GRSPWROUTER / GRSPW_CODEC.

Since data will appear on both the rising and falling edge of the recovered clock there must be two registers, one for each edge. Figure 203 shows the clock recovery scheme. As can be seen in the figure, the SpaceWire data input is used for generating the receiver clock as well as input of several flip-flops. Care must be taken so that the delay from the data and strobe signals through the clock network is longer than the delay to the flip-flop's data input + setup time.

For self-clocking mode, the core's RXCLKI and RXCLKIN inputs are unused. The NRXCLKI input is unused if scan test support is not implemented (scantest generic set to 0). If scan test mode is implemented (scantest generic set to 1) then the NRXCLKI input is used as the receiver clock during scan test (TESTEN input high), otherwise it is not used. Details on how to connect the other input / output signals are found in section 79.5.

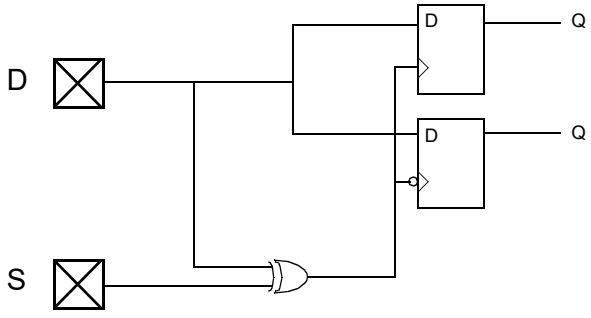


Figure 203. The clocking scheme for self clocking mode. The clock is recovered from the SpaceWire data and strobe signals using an xor gate.

79.2.2 Frontgrade transceiver (input_type = 1)

When using the Frontgrade UT200SpWPHY01 transceiver, the RxClk output from the transceiver shall be connected to the RXCLKI input of GRSPW2_PHY. RxDR and RxDF should be connected to DI and SI respectively. The GRSPW2_PHY outputs DO, DOV and DCONNECT are generated in the same way as for the self-clocking mode.

RXCLKIN is unused in this configuration. NRXCLKI is unused if scan test support is not implemented (SCANTEST generic set to 0). If scan test support is implemented (SCANTEST generic set to 1) the NRXCLKI is used in the following way: When not in scan test mode (TESTEN input is low) the NRXCLKI input should be the inverse of the RXCLKI input, and when in scan test mode (TESTEN input is high) the NRXCLKI input should be the same clock as the RXCLKI input. Details on how to connect the other input / output signals is found in section 79.5.

79.2.3 SDR sampling (input_type = 2)

In this mode the sampling clock should be connected to the RXCLKI input. RXCLKIN and NRXCKLI are unused in this mode. Details on how to connect the other input / output signals is found in section 79.5.

The core internally detects new data on the data line. Since only one bit can be detected each clock cycle only bit 0 of the data signal to the receiver (DO output) is used. Normally the sampling frequency must be at least 1.5 times higher than the maximum bitrate for correct operation.

79.2.4 DDR sampling (input_type = 3)

The sampling clock should be connected to RXCLKI. If the selected technology does not have DDR primitives that automatically generate the inverted clock then the inverted sampling clock has to be connected to RXCLKIN. NRXCLKI is unused in this mode. Details on how to connect the other input / output signals is found in section 79.5.

The core internally detects new data on the data line. There can be two bits detected each clock cycle so both data signals to the receive (DO output) have to be used. The sampling frequency times two has to be at least 1,5 times the maximum bitrate.

79.2.5 DDR sampling with internal pad (input_type = 4)

Same as DDR sampling (input_type = 3) except that both DDR registers and input pads are instantiated within the GRSPW2_PHY. This option can be used on technologies where DDR registers and pads must be instantiated as one entity. This option (input_type = 4) should not be used unless recommended by Frontgrade Gaisler. If input_type = 4 is selected then the level and voltage of the instantiated pads are specified via the VHDL generics input_level and input_voltage, and external pads should not be instantiated for the SpaceWire input signals.

79.2.6 Self-clocking with external clock recovery (input_type = 5)

Same as self-clocking (input_type = 0) except that the clock recovery is done externally to the GRSPW2_PHY. The rxclki input is used to clock the receiver registers within the core.

The core's RXCLKIN input is unused. The NRXCLKI input is unused if scan test support is not implemented (scantest generic set to 0). If scan test mode is implemented (scantest generic set to 1) then the NRXCLKI input is used as the receiver clock during scan test (TESTEN input high), otherwise it is not used.

79.2.7 Self-clocking with external clock recovery and DDR register (input_type = 6)

Same as self-clocking with external clock recovery (input_type = 5) except that a DDR register is used internally in GRSPW2_PHY for the incoming data.

79.3 Configuration options

Table 1431 shows the configuration options of the core (VHDL generics).

*Table 1431.*Configuration options

Generic	Function	Allowed range	Default
scantest	Enable scantest mode	0 - 1	0
tech	Selects technology for DDR registers when input_type = 3, 4 or 6, as well as technology for clock buffer when input_type = 0, 5, 6 and rxclkbuftype /= 0.	0 - NTECH	1
input_type	Selects receiver type. 0 = Self clocking (xor), 1 = Interface for Frontgrade UT200SpWPHY01 SpaceWire transceiver, 2 = Single data rate sampling, 3 and 4 = Double data rate sampling, 5 = Self-clocking with external recovery, 6 = Self-clocking with external recovery and DDR register for data. This generic must be set to the same value as the GRSPW2 / GRSPWROUTER / GRSPW_CODEC generic with the same name.	0 - 4	0
input_level	Selects level for pads instantiated within PHY. Only used when input_type = 4.	-	0
input_voltage	Selects voltage for pads instantiated within PHY. Only used when input_type = 4.	-	0
rxclkbuftype	Selects clock buffer type for receiver clock. 0 = No clock buffer (synthesis tools may still infer a buffer). 1 = Hard-wired clock, 2 = Routed clock.	0 - 2	0
rstsrctmr	Enables the Triple Module Redundancy for the asynchronous reset nets of the core	0 - 1	0

79.4 Scan support

Scan support is enabled by setting the SCANTEST generic to 1. When enabled, the asynchronous reset of any flip-flop will be connected to TESTRT when TESTEN = ‘1’.

79.5 Signal descriptions

Table 1432 shows the interface signals of the model (VHDL ports).

*Table 1432.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	-	Input	Reset	Low
RXCLKI	-	Input	Receiver clock input. See sections 79.2.1 through 79.2.1 for details on how this signal is used for respective input type.	-
RXCLKIN	-	Input	The inverse of RXCLKI should be inputed here. See sections 79.2.1 through 79.2.1 for details on how this signal is used for respective input type.	-
NRXCLKI	-	Input	The inverse of RXCLKI should be inputed here. See sections 79.2.1 through 79.2.1 for details on how this signal is used for respective input type.	-
DI	-	Input	SpaceWire data input. When using a transceiver (input_type = 1), the transceiver's RxDR output should be connected to this input.	-
SI	-	Input	SpaceWire strobe input. When using a transceiver (input_type = 1), the transceiver's RxDF output should be connected to this input.	-
DO(1:0)	-	Output	Recovered data. Synchronous to RXCLKO. Should be connected to data inputs on GRSPW2 / GRSPWROUTER / GRSPW_CODEC. Each bit is valid when the bit at the corresponding index in the DOV signal is asserted. If only one bit is received on a given clock cycle then only the bit at the lower index is valid. If both bits are valid then the one at the bit at the lower index is the one received first.	-
DOV(1:0)	-	Output	Data valid. Synchronous to RXCLKO. Should be connected to data valid inputs on GRSPW2 / GRSPWROUTER / GRSPW_CODEC.	High
DCONNECT(1:0)	-	Output	Disconnect strobe signals. Synchronous to RXCLKO. Should be connected to the disconnect inputs on GRSPW2 / GRSPWROUTER / GRSPW_CODEC.	-
DCONNECT2 (1:0)	-	Output	If the generic rstsrctmr is set to 1 (TMR enabled), it is a replication of DCONNECT. It should be connected to the disconnect inputs on GRSPW2 / GRSPW_CODEC. If the generic rstsrctmr is set to 0 (TMR disabled), this signal is set to low and has no use.	-
DCONNECT3 (1:0)	-	Output	If the generic rstsrctmr is set to 1 (TMR enabled), it is a replication of DCONNECT. It should be connected to the disconnect inputs on GRSPW2 / GRSPW_CODEC. If the generic rstsrctmr is set to 0 (TMR disabled), this signal is set to low and has no use.	-
RXCLKO	-	Output	Receiver clock output. Should be connected to receiver clock inputs on GRSPW2 / GRSPWROUTER / GRSPW_CODEC.	-
TESTRST	-	Input	Scan test reset.	Low
TESTEN	-	Input	Scan test enable.	High

79.6 Library dependencies

Table 1433 shows the libraries used when instantiating the model (VHDL libraries).

*Table 1433.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	SPACEWIRE	component	Component declaration.

79.7 Instantiation

This example shows how the core can be instantiated in DDR mode (input_type = 3). rxclkin is used if the technology cannot generate the inverted clock in the DDR register otherwise it can be left floating. nrxclki is not needed at all for this mode and spw_rxtxclk is only connected not to violate VHDL syntax. Only one port is used so the same clock is coupled to both receiver clock inputs.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.spacewire.all;

entity spacewire_ex is
  port (
    clk   : in  std_ulogic;
    rstn          : in  std_ulogic;

    -- spacewire signals
    spw_rxdp      : in  std_ulogic;
    spw_rxdn      : in  std_ulogic;
    spw_rxsp      : in  std_ulogic;
    spw_rxsn      : in  std_ulogic;
    spw_txdp      : out std_ulogic;
    spw_txdn      : out std_ulogic;
    spw_txsp      : out std_ulogic;
    spw_txsn      : out std_ulogic;

    spw_rxtxclk   : in  std_ulogic;
    spw_rxclkn    : in  std_ulogic
    );
end;

architecture rtl of spacewire_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- Spacewire signals
  signal swni      : grspw_in_type;
  signal swno      : grspw_out_type;
  signal spw_rxclk : std_logic_vector(1 downto 0);
  signal dtmp    : std_ulogic;
  signal stmp      : std_ulogic;
  signal rxclko    : std_ulogic;
begin

  -- AMBA Components are instantiated here

  spw_phy0 : grspw2_phy
    generic map(
      scantest   => 0,
```

```
    tech       => memtech,
    input_type => 3)
  port map(
    rstn       => rstn,
    rxclki     => spw_rxtxclk,
    rxclkin    => spw_rxclkn,
    nrxclki    => spw_rxtxclk,
    di         => dtmp,
    si         => stmp,
    do         => swni.d(1 downto 0),
    dov        => swni.dov(1 downto 0),
    dconnect   => swni.dconnect(1 downto 0),
    dconnect2  => swni.dconnect2(1 downto 0),
    dconnect3  => swni.dconnect3(1 downto 0),
    rxclko     => rxclko);

sw0 : grspw2
generic map(
  tech         => memtech,
  hindex       => 0,
  pindex       => 10,
  paddr        => 10,
  pirq         => 10,
  ports        => 1,
  dmachan      => 1,
  rmap         => 0,
  rmapcrc      => 1,
  fifosize1    => 32,
  fifosize2    => 32,
  rxunaligned  => 1,
  rmapbufs     => 4,
  output_type  => 1,
  input_type   => 3,
  rxtx_sameclk => 1)
port map(rstn, clkm, rxclko, rxclko, spw_rxtxclk, spw_rxtxclk, ahbmi,
     ahbmo(0), apbi, apbo(10), swni, swno);

swni.tickin <= '0'; swni.rmapen <= '1';
swni.clkdiv10 <= conv_std_logic_vector(SPW_TX_FREQ_KHZ/10000-1, 8);

spw_rxd_pad : inpad_ds generic map (padtech, lvds, x25v)
  port map (spw_rxdp, spw_rxdn, dtmp);
spw_rxs_pad : inpad_ds generic map (padtech, lvds, x25v)
  port map (spw_rxsp, spw_rxsn, stmp);
spw_txd_pad : outpad_ds generic map (padtech, lvds, x25v)
  port map (spw_txdp, spw_txdn, swno.d(0), gnd(0));
spw_txs_pad : outpad_ds generic map (padtech, lvds, x25v)
  port map (spw_txsp, spw_txsn, swno.s(0), gnd(0));
...
```


Frontgrade Gaisler AB
Kungsgatan | SE-411 19 | Goteborg | Sweden
+46 31 7758650 | frontgrade.com/gaisler

80 GRSPW_CODEC - SpaceWire encoder-decoder

80.1 Overview

The SpaceWire encoder-decoder implements an encoder-decoder compliant to the SpaceWire standard (ECSS-E-50-12C). It provides a generic host-interface consisting of control signals, status signals, time-code interface and 9-bit wide data buses connecting to a pair of FIFOs.

The core can also be configured with two SpaceWire ports with manual or automatic switching between them.

Transmitter outputs can be either Single Data Rate (SDR), Double Data Rate (DDR) or connected to an SpaceWire transceiver. The receiver can be connected either to an SpaceWire transceiver or recover the data itself using a self-clocking scheme or sampling (SDR or DDR).

The core is practically identical to the encoder-decoder used in the GRSPW2, the only difference being the host-interface.

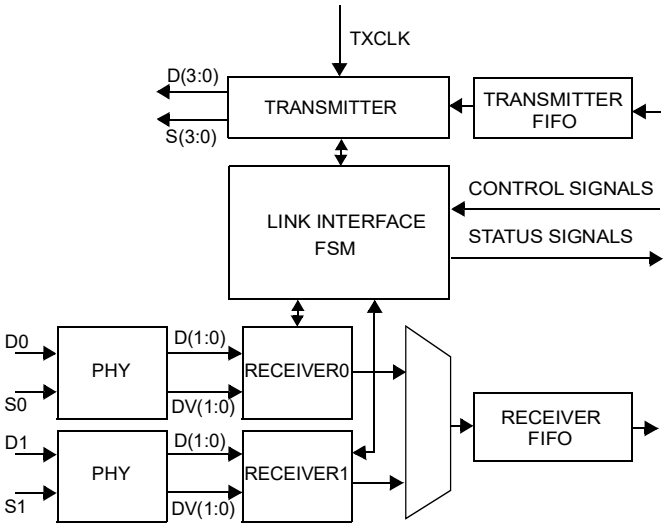


Figure 204. Block diagram

80.2 Operation

80.2.1 Overview

A block diagram of the internal structure of the core can be found in figure 204. It consists of the receiver, transmitter and the link interface FSM. They handle communication on the SpaceWire network. The PHY blocks provides a common interface for the receiver to the four different data recovery schemes and is external to this core. A short description is found in section 80.2.6. The complete documentation is found in the GRSPW2_PHY section of the GRIP user manual.

Time-codes are transmitted through a signal interface as specified in the SpaceWire standard.

80.2.2 Link-interface FSM

The link-interface FSM controls the link interface (a more detailed description is found in the SpaceWire standard). The low-level protocol handling (the signal and character level of the SpaceWire standard) is handled by the transmitter and receiver while the FSM handles the exchange level.

The link-interface FSM is controlled through the control signals. The link can be disabled through the link disabled signal, which depending on the current state, either prevents the link-interface from reaching the started state or forces it to the error-reset state. When the link is not disabled, the link

interface FSM is allowed to enter the started state when either the link start signal is asserted or when a NULL character has been received and the autostart signal is asserted.

The current state of the link-interface determines which type of characters are allowed to be transmitted which together with the requests made from the host interface determine what character will be sent.

Time-codes are sent when the FSM is in the run-state and a request is made through the time-interface (described in section 80.2.8).

When the link-interface is in the connecting- or run-state it is allowed to send FCTs. FCTs are sent automatically by the link-interface when possible. This is done based on the maximum value of 56 for the outstanding credit counter and the currently free space in the receiver FIFO. FCTs are sent as long as the outstanding counter is less than or equal to 48 and there are at least 8 more empty FIFO entries than the counter value.

N-Chars are sent in the run-state when they are available from the transmitter FIFO and there are credits available. NULLs are sent when no other character transmission is requested or the FSM is in a state where no other transmissions are allowed.

The credit counter (incoming credits) is automatically increased when FCTs are received and decreased when N-Chars are transmitted. Received N-Chars are stored to the receiver N-Char FIFO while received Time-codes are handled by the time-interface.

80.2.3 Transmitter

The state of the FSM, credit counters, requests from the time-interface and requests from the transmitter FIFO are used to decide the next character to be transmitted. The type of character and the character itself (for N-Chars and Time-codes) to be transmitted are presented to the low-level transmitter which is located in a separate clock-domain.

This is done because the SpaceWire link is usually run on a different frequency than the host system clock. The core has a separate clock input which is used to generate the transmitter clock. More information on transmitter clock generation is found in section 80.5.2. Since the transmitter often runs on high frequency clocks (> 100 MHz) as much logic as possible has been placed in the system clock domain to minimize power consumption and timing issues.

The transmitter logic in the host clock domain decides what character to send next and sets the proper control signal and presents any needed character to the low-level transmitter as shown in figure 205. The transmitter sends the requested characters and generates parity and control bits as needed. If no requests are made from the host domain, NULLs are sent as long as the transmitter is enabled. Most of the signal and character levels of the SpaceWire standard is handled in the transmitter. External LVDS drivers are needed for the data and strobe signals. The outputs can be configured as either single- or double data rate. The latter increases maximum bitrate significantly but is not available for all technologies.

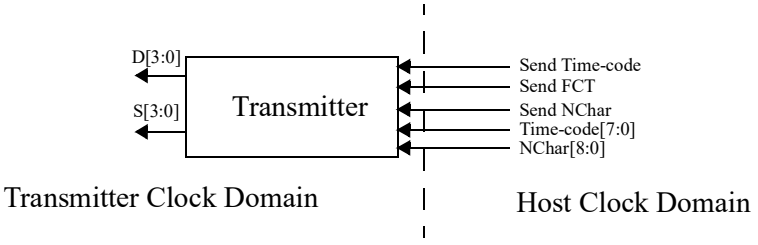


Figure 205. Schematic of the link interface transmitter. Some bits in D[3:0] and S[3:0] may be unused, depending on the values of the *output_type*, and *ports* VHDL generics (see section 80.8).

80.2.4 Receiver

The receiver detects connections from other nodes and receives characters as a bit stream recovered from the data and strobe signals by the GRSPW2_PHY module which presents it as a data and data-valid signal. Both the receiver and GRSPW2_PHY are located in a separate clock domain which runs on a clock driven by the PHY. More information on the clock-generation can be found in section 80.5.2.

The receiver is activated as soon as the link-interface leaves the error reset state. Then after a NULL is received it can start receiving any characters. It detects parity, escape and credit errors which causes the link interface to enter the error-reset state. Disconnections are handled in the link-interface part in the tx clock domain because no receiver clock is available when disconnected.

Received characters are flagged to the host domain and the data is presented in parallel form. The interface to the host domain is shown in figure 206. L-Chars are the handled automatically by the host domain link-interface part while all N-Chars are stored in the receiver FIFO for further handling. If two or more consecutive EOPs/EEPs are received all but the first are discarded.

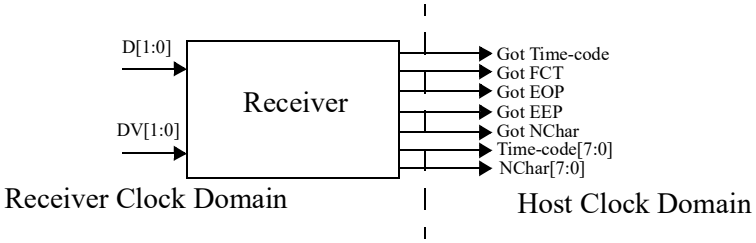


Figure 206. Schematic of the link interface receiver.

80.2.5 Dual port support

The core can be configured to include an additional SpaceWire port. With dual ports the transmitter drives an additional pair of data/strobe output signals and one extra receiver is added to handle a second pair of data/strobe input signals.

One of the ports is set as active (how the active port is selected is explained below) and the transmitter drives the data/strobe signals of the active port with the actual output values as explained in section 80.2.3. The inactive port is driven with zero on both data and strobe.

Both receivers will always be active but only the active port's interface signals (see figure 206) will be propagated to the link interface FSM. Each time the active port is changed, the link will be reset so that the new link is started in a controlled manner.

When the noportforce signal is zero, the portsel signal selects the active port. When the noportforce signal is set to one, the active port is automatically selected during initialization. For the latter mode, the port on which the first bit is received will be selected as the active port. If the initialization attempt fails on that port the link is reset and the active port is again sected based on which port the first bit is received.

80.2.6 Receiver PHY

The receiver supports four different input data recovery schemes: self-clocking (xor), sampling SDR, sampling DDR and the SpaceWire transceiver. These four recovery types are handled in the PHY module and data is presented to the receiver as a data and data-valid signal. This part of the receiver must often be constrained and placing it in a separate module makes this process easier with the most common synthesis tools. The input type is selected using a VHDL generic. More information about the PHY can be found in the GRSPW2_PHY section of the GRIP user manual.

80.2.7 Setting link-rate

The input signal IDIVISOR determines the link-rate during initialization (all states up to and including the connecting-state). The value of this input signal is also used to calculate the link interface FSM timeouts (6.4 µs and 12.8 µs, as defined in the SpaceWire standard). The IDIVISOR signal should always be set so that a 10 Mbit/s link-rate is achieved during initialization. In that case the timeout values will also be calculated correctly.

To achieve a 10 Mbit/s link-rate, the IDIVISOR signal should be set according to the following formulas:

With single data rate (SDR) outputs:

IDIVISOR = (<frequency in MHz of TXCLK> / 10) - 1

With double data rate (DDR) outputs, or when connected to Frontgrade SpaceWire transceiver:

IDIVISOR = (2 x <frequency in MHz of TXCLK> / 10) - 1

The link-rate in run-state is controlled with the run-state divisor, the RDIVISOR input signal. The link-rate in run-state is calculated according to the following formulas:

With SDR outputs:

<link-rate in Mbits/s> = <frequency in MHz of TXCLK> / (RDIVISOR+1)

With DDR outputs / Frontgrade SpaceWire transceiver:

<link-rate in Mbits/s> = 2 x <frequency in MHz of TXCLK> / (RDIVISOR+1)

The value of RDIVISOR only affects the link-rate in run-state, and does not affect the 6.4 us or 12.8 us timeouts values.

Note that when using DDR ouputs, or when connected to Frontgrade SpaceWire transceiver, there is a limitation in the usable clock divisor values. All even values (except 0) will result in the same bitrate as the one higher odd number.

An example of clock divisor and resulting link-rate, with a TXCLK frequency of 50 MHz, is shown in the table 1434. Also see 80.5.2 for information on clock requirements.

Table 1434. SpaceWire link-rate example with 50 MHz TXCLK

	Link-rate in Mbit/s	
Clock divisor value	**SDR outputs**	**DDR outputs / Frontgrade SpaceWire transceiver**
0	50	100
1	25	50
2	16.67	25
3	12.5	25
4	10	16.67
5	8.33	16.67
6	7.14	12.5
7	6.25	12.5
8	5.56	10
9	5	10

80.2.8 Time interface

The time interface is used for sending Time-codes over the SpaceWire network and consists of a timein signal, tickin signal, timeout signal and a tickout signal, plus the status flags tickin_done and tickin_busy.

Each Time-code sent from the core is a concatenation of the time-code control bits (timein(7:6), represented as timectrlin in the diagram below) and the time-code value (timein(5:0)). It is transmitted each time tickin is kept asserted until the tickin_done output is asserted. Time-codes are only transmitted when the link-interface FSM is in run-state and when the previous character transmission is finished. This can cause a delay between when the assertion of tickin_done after tickin has been asserted. If tickin is not kept asserted until tickin_done is asserted the time-code will not be transmitted. Figure 207 shows an example of how a time-code is transmitted.

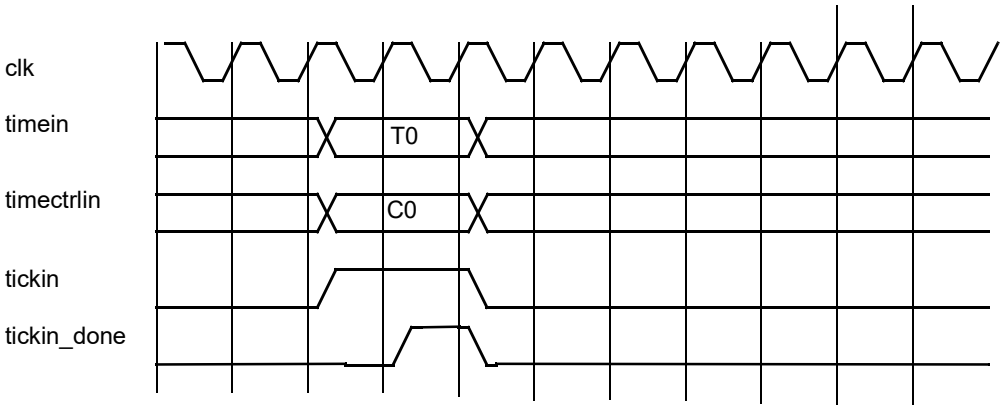


Figure 207. Transmitting a time-code using tickin, timein and timectrlin.

Received Time-codes are presented on the timeout signal, where the control bits are timeout(7:6) and displayed as timectrlout in the diagram below. They are valid when the tickout output is asserted. A tickout is generated each time a valid time-code is received. When the tickout is generated the tick-out signal will be asserted one clock-cycle. Figure 208 shows how time-codes are received.

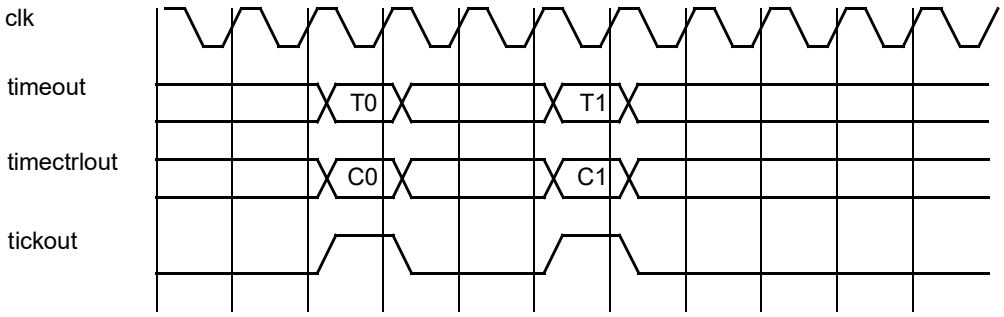


Figure 208. Receiving time-codes using tickout, timeout and timectrlout.

80.3 Receiver interface

The receiver interface consists of the following signals connected to the receiver FIFO: rxicharav, rxicharcnt, rxichar, rxiread. Rxicharav is asserted when there are one or more characters available in the receiver FIFO whereas rxicharcnt shows the actual number available. Rxiread should be asserted for one cycle when rxicharav is asserted to read out a character. The character is available immediately, so it shall be read during the same clock cycle that rxiread is asserted. Figure 209 shows an example of reading characters from the receiver FIFO.

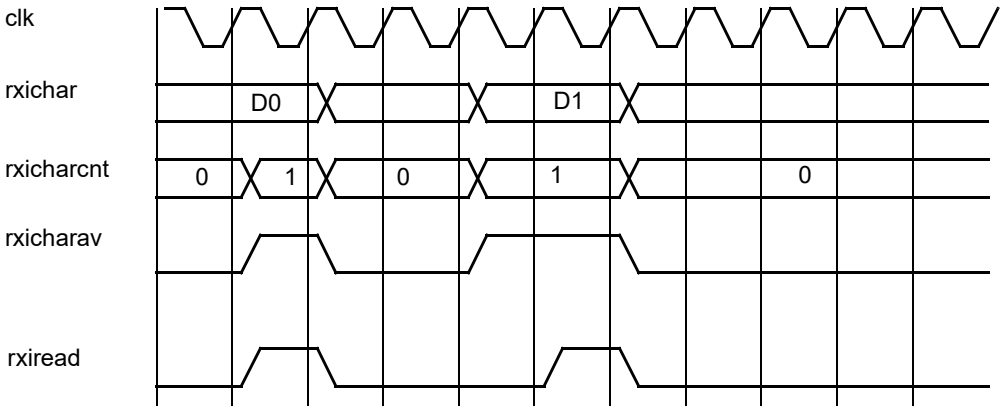


Figure 209. Receiving characters through the FIFO interface.

80.3.1 Link errors

When an link error occurs during reception an EEP is automatically inserted into the Receiver FIFO if the previous character written to the FIFO was not an EOP or EEP.

80.4 Transmitter interface

The transmitter interface consists of the following signals: txiwrite, txichar, txififorst, txicharcnt, txifull, txiempty. Txifull is asserted when the transmitter FIFO is full while txicharcnt shows the actual number of characters currently in the FIFO. Txiwrite should be asserted for one cycle to write the value on txichar into the FIFO. Txififorst should be asserted for one cycle to discard all characters in the FIFO. No new characters should be written to the FIFO the same or the following cycle that txififorst is asserted. Figure 210 shows an example of writing characters to the transmitter FIFO.

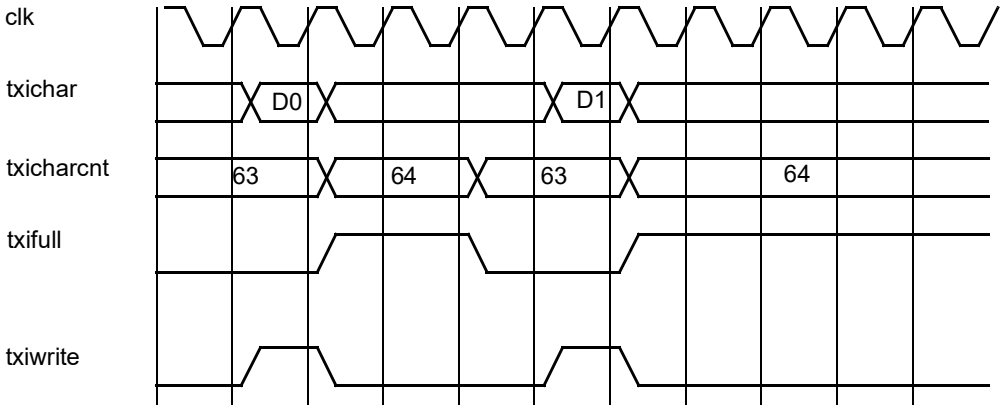


Figure 210. Transmitting characters through the FIFO interface.

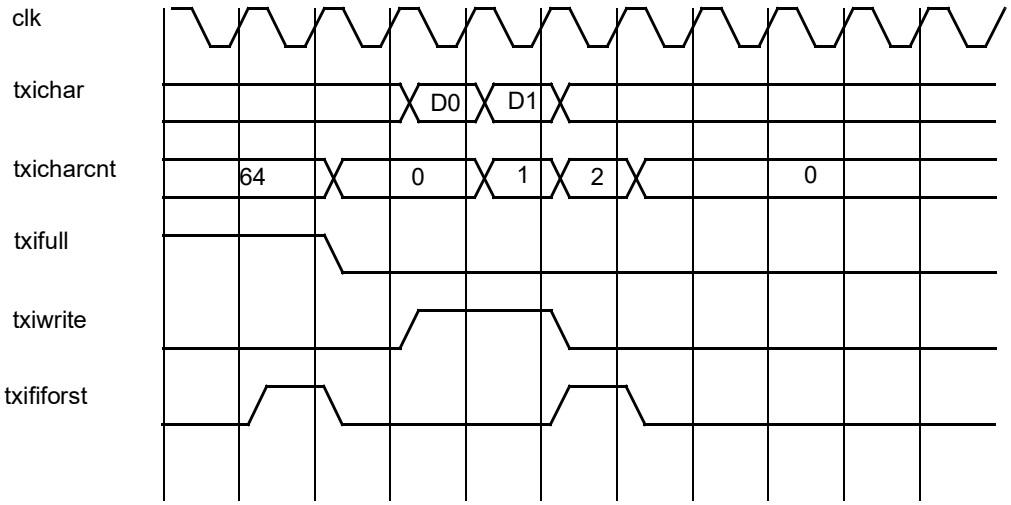


Figure 211. Use of txififorst.

80.4.1 Link errors

When a link error occurs characters read from the transmitter FIFO by the transmission logic will be discarded up to and including the next EOP or EEP character.

80.5 Implementation

80.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

There are five input resets, in order to cover every clock domain: AMBA reset (rst), transmitter synchronous reset (txsyncrst), receiver synchronous reset for port 0 (rxsyncrst0), receiver synchronous reset for port 1 (rxsyncrst1) and a receiver asynchronous reset shared by both ports (rxasyncrst).

Additionally, the core outputs an internal reset for synchronization between transmitter and receivers, lio.rxrst.

The core does not implement any kind of internal reset generation or synchronization, the input resets are completely independent. The reset generation shall be done in a higher instance, taking into account the clock domains and also the output reset of the core. A description of how the resets shall be combined and generated can be found below. Frontgrade Gaisler advise to follow these guidelines unless indicated otherwise.

- The AMBA reset is the external reset synchronized with the AMBA clock by using a reset generator.
- The transmitter reset is the external reset synchronized with the transmitter clock by using a reset generator.
- The asynchronous reset for both receiver channels is simply the output synchronization reset, lio.rxrst, connected directly to the input rxasyncrst.
- The synchronous reset for each receiver channel is the output receiver reset, lio.rxrst, synchronized with the appropriate clock domain by using a reset generator. Its output is the synchronous reset for the specific receiver port, rxsyncrst0 or rxsyncrst1.

Further information about the proper way of combining the reset signals or using the wrapper can be found in 80.13. Instantiation.

80.5.2 Clock-generation

The receiver module found in figure 204 should be clocked with the RXCLKO output signal from the GRSPW2_PHY module. See the example instantiation in this section and the GRSPW2_PHY section of the GRIP manual for more information on how to connect this clock.

The transmitter clock is generated from the TXCLK input. A separate clock input is used to allow the transmitter to be run at much higher frequencies than the system clock. The SpaceWire node contains a clock-divider which divides the TXCLK signal to the wanted frequency. The transmitter clock should be 10 MHz during initialization and any frequency above 2 MHz in the run-state.

There is an input signal called CLKDIV10 which sets the frequency during initialisation and one called CLKDIV which is used in run-state. See 80.2.7 for details on how to set the clock divisor values.

Since only integer values are allowed for the clock division and the required init-frequency is 10 MHz the frequency of the TXCLK input must be a multiple of 10 MHz. The clock divisor value is 8-bits wide so the maximum txclk frequency supported is 2.56 GHz (note that there is also a restriction on the relation between the system and transmit clock frequencies).

80.5.3 Timers

There are two timers in the codec: one for generating the 6.4/12.8 µs periods and one for disconnect timing.

The timeout periods are generated from the TX clock whose frequency must be at least 10 MHz to guarantee disconnect timing limits. The same clock divisor is used as for the TX clock during initialisation so it must be set correctly for the link timing to work.

80.5.4 Synchronization

The transmitter and receiver bit rates can be eight times higher than the system clock frequency. This includes a large margin for clock skew and jitter so it might be possible to run at even higher rate differences. Note also that the receiver clocks data at both negative and positive edges for the input modes 0 and 1 so the bitrate is twice the clock frequency. There is no direct relationship between bitrate and frequency for the sampling modes.

The clock synchronization is just one limiting factor for the clock frequency, it might for example not be possible to achieve the highest possible frequency for certain technologies.

The asynchronous reset to the receiver clock domain has to have a maximum delay of one receiver clock cycle to ensure correct operation. This is needed because the receiver uses a completely asynchronous reset. To make sure that nothing bad happens there is a synchronous reset guard which prevents any signals from being assigned before all registers have their reset signals released.

In the sampling modes this asynchronous reset can be removed if both the receiver and transmitter runs on the same clock. In that case set the RXTX_SAMECLK generic to 1.

80.5.5 Fault-tolerance

The core can optionally be implemented with fault-tolerance against SEU errors in the FIFO memories. The fault-tolerance is enabled through the *ft* VHDL generic. Possible options are byte parity protection (*ft = 1*) or TMR registers (*ft = 2*). Note: the GPL version of GRLIB does not include fault-tolerance, and the core will not work unless the *ft* VHDL generic is 0.

80.5.6 Synthesis

The fact there are three clock domains in the core of which all are possibly high frequency clocks makes it necessary to declare all paths between the clock domains as false paths. In Synplify this is most easily done by declaring all the clocks to be in different clockgroups in the sdc file (if Synplify

does not automatically put them in different groups). This will disable any timing considerations between the clock domains and these constraints will also propagate to the place and route tool.

80.5.7 Technology mapping

The core has two generics for technology mapping: *tech and techfifo*. *Tech* selects the technology for DDR registers (if applicable) and FIFO memories. *Techfifo* selects whether *tech* should be used to select the technology for the FIFO memories or if they should be inferred. *Tech* and *memtech* can be set to any value from 0 to NTECH as defined in the GRLIB.TECH package.

80.5.8 RAM usage

The core maps all RAM memories on the syncram_2p component if the *ft* generic is 0 and to the syncram_2pft component for other values. The syncrams are located in the technology mapping library (TECHMAP). The organization of the different memories are described below. If techfifo and/or memtech is set to 0 the synthesis tool will infer the memories. Either RAM blocks or flip-flops will be used depending on the tool and technology. The number of flip-flops used is *syncram depth x syncram width* for all the different memories.

Transmitter FIFO

The transmitter FIFO consists of one syncram_2p block with a width of 9-bits. The depth is determined by the configured FIFO depth. Table 1435 shows the syncram organization for the allowed configurations.

*Table 1435.*syncram_2p sizes for the transmitter FIFO.

Fifosize	Syncram_2p organization
16	16x9
32	32x9
64	64x9
...	...
2048	2048x9

Receiver FIFO

The receiver FIFO consists of one syncram_2p block with a width of 9-bits. The depth is determined by the configured FIFO depth. Table 1436 shows the syncram organization for the allowed configurations.

*Table 1436.*syncram_2p sizes for the receiver FIFO.

Fifosize	Syncram_2p organization
16	16x9
32	32x9
64	64x9
...	...
2048	2048x9

80.6 Registers

There are no user accessible registers in the core.

80.7 Vendor and device identifiers

The vendor and device identifiers are only applicable for cores with AHB interfaces.

80.8 Configuration options

Table 1437 shows the configuration options of the core (VHDL generics).

*Table 1437.*Configuration options

Generic	Function	Allowed range	Default
ports	Sets the number of ports	1 - 2	1
input_type	Select receiver type. 0 = Self clocking (xor), 1 = Interface for Frontgrade UT200SpWPHY01 SpaceWire transceiver, 2 = Single data rate sampling, 3 and 4 = Double data rate sampling, 5 = Self-clocking with external recovery, 6 = Self-clocking with external recovery and DDR register for data.. This generic must be set to the same value as the GRSPW2_PHY generic with the same name.	0 - 6	0
output_type	Select transmitter type. 0 = single data rate, 1 = double data rate, 2 = interface for Frontgrade UT200SpWPHY01 SpaceWire transceiver	0 - 2	0
rxtx_sameclk	Set to one if the same clock net is connected to both the receiver and transmitter (which means this feature is only applicable when the receiver uses sampling). This will remove some unnecessary synchronization registers.	0 - 1	0
fifosize	Sets the number of entries in the 9-bit wide transmitter and receiver FIFOs.	16 - 2048	64
tech	Technology for FIFO memories.	0 - NTECH	inferred
scantest	Enable scantest features.	0 - 1	0
techfifo	Enable GRLIB technology mapped FIFO memories. If not enabled a behavioral model is used and the result will be synthesis tool dependent.	0 - 1	0
ft	Enable fault-tolerance against SEU errors	0 - 2	0
rstsrctmr	Enables the Triple Module Redundancy for the asynchronous reset nets of the core	0 - 1	0

80.9 Signal descriptions

Table 1438 shows the interface signals of the core (VHDL ports).

Table 1438. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AMBA Reset	Low
CLK	N/A	Input	AMBA Clock	-
RXASYNCRST	N/A	Input	Asynchronous reset for ports 0 and 1 of the receiver	Low
RXSYNCRST0	N/A	Input	Synchronous reset for port 0 of the receiver	Low
RXCLK0	N/A	Input	Receiver clock for port 0.	-
RXSYNCRST1	N/A	Input	Synchronous reset for port 1 of the receiver	Low
RXCLK1	N/A	Input	Receiver clock for port 1. Unused if the VHDL generic ports is 2.	-
TXSYNCRST	N/A	Input	Synchronous reset for the transmitter	Low
TXCLK	N/A	Input	Transmitter default run-state clock	-

Table 1438. Signal descriptions

Signal name	Field	Type	Function	Active
TXCLKN	N/A	Input	Transmitter inverted default run-state clock. Only used in DDR transmitter mode for technologies not supporting local generation of inverted clock.	-
TESTEN	N/A	Input	Scan test enable	High
TESTRST	N/A	Input	Scan test reset	Low
LII	D[3:0]	Input	Data input. Each bit is valid when the bit at the corresponding index in the DV signal is asserted. Bits (1:0) are used for port 0 and bits (3:2) are used for port 1 if enabled. If only one bit is received during one clock cycle then only the bit at the lower index is valid. If both bits are valid then the one at the lower index was the one received first. Bits (1:0) should be synchronous to RXCLK0 and bits (3:2) to RXCLK1. Connect to the DO[1:0] output of one GRSP-W2_PHY per implemented port.	-
	DV[3:0]	Input	Data valid qualifier for the LII.D[3:0] input. Connect to the DOV[1:0] output of one GRSP-W2_PHY per implemented port.	High
	DCONNECT[3:0]	Input	Disconnect reset. When asserted the corresponding disconnect counter will be reset. The counters connected to bits (1:0) apply to port 0 while (3:2) apply to port 1 if enabled.	Low
	DCONNECT2[3:0]	Input	Disconnect. Bits 3:2 are unused in one-port configurations. This is a copy of DCONNECT, as part of the triplication of the combinational logic related to asynchronous reset nets.	Low
	DCONNECT3[3:0]	Input	Disconnect. Bits 3:2 are unused in one-port configurations. This is a copy of DCONNECT, as part of the triplication of the combinational logic related to asynchronous reset nets.	Low
	LINKDISABLED	Input	Disables the SpaceWire link	High
	LINKSTART	Input	Starts the SpaceWire link	High
	AUTOSTART	Input	Enables the autostart feature for the SpaceWire link	High
	PORTSEL	Input	Selects the active port if the NOPORTFORCE signal is set to 0 and the core is configured with dual ports.	-
	NOPORTFORCE	Input	Disables forced port selection using the PORTSEL signal and lets the core automatically select the active link.	High
	RDIVISOR[7:0]	Input	Clock divisor value used for generating the transmit frequency from the txclk input in run-state. Bit 0 is the least significant. See 80.2.7 for details on how to set this input signal.	-
	IDIVISOR[7:0]	Input	Clock divisor value used for generating the transmit frequency from the txclk input during initialization (started and connecting states). Bit 0 is the least significant. See 80.2.7 for details on how to set this input signal.	-

Table 1438. Signal descriptions

Signal name	Field	Type	Function	Active
LII	RXIREAD	Input	Receiver FIFO read. Assert for one cycle (synchronous to CLK) and the next character will be available on the RXICHAR output the next clock cycle if available in the FIFO.	High
	RXIFIFORST	Input	Empty the receiver FIFO. When asserted for one cycle (synchronous to CLK) all characters currently present in the FIFO will be discarded. The reset will have taken on the second cycle after the reset. The first cycle after characters may still be present.	High
	TXIWRITE	Input	Transmitter FIFO write. Assert for one cycle (synchronous to CLK) to write the character on the TXICHAR input into the FIFO on the rising edge of the clock.	High
	TXICHAR[8:0]	Input	Transmit character input to FIFO. Characters are transmitted in the order they are written into the FIFO. Bit 0 is the SpaceWire control bit. When set to 1 only bits (2:1) are transmitted (EOP or EEP). Bits are transmitted in order beginning at the lowest index.	-
	TXIFIFORST	Input	Empty the transmitter FIFO. When asserted for one cycle (synchronous to CLK) all characters currently present in the FIFO will be discarded. New characters shall not be written to the transmitter FIFO the same cycle or the cycle after TXIFIFORST is asserted.	High
	TICKIN	Input	Time counter tick input. Should be asserted one clock cycle (synchronous to CLK) to transmit a time-code. If the FSM is not in run-state no time-code will be transmitted and the tick will not be registered. The TICKIN_DONE output can be used to verify that the time-code has been transmitted,	High
	TIMEIN[7:0]	Input	Control bits (7:6) and value (5:0) for the time-code interface. Transmitted in order starting at the lowest index when tickin is asserted.	-
LIO	DO[3:0]	Output	Data output. The meaning of the bits depends on the value of the *output_type* generic. When output_type=0 or 1, DO[0] is the data output for port 0, DO[1] is the data output for port 1, and DO[3:2] are unused. In single-port configurations DO[3:1] are unused. When output_type=2 (external UT200SpW-PHY01 SpaceWire transceiver), DO[1:0] is the output for port 0 and DO[3:2] is the output for port 1. In single-port configurations DO[3:2] are unused.	-
	SO[3:0]	Output	Strobe output. The bit mapping is the same as for the LIO.DO[3:0] output (see above).	-
	STATE[2:0]	Output	Current linkinterface FSM state. 0=error-reset, 1=error-wait, 2=ready, 3=started, 4=connecting, 5=run.	-
	ACTPORT	Output	0=port 0 is active and 1=port 1 is active. Unused if the second port is not enabled.	-
	DCONNECTERR	Output	Asserted for one clock cycle (synchronous to CLK) when a disconnect error is detected.	High

Table 1438. Signal descriptions

Signal name	Field	Type	Function	Active
LIO	CREDERR	Output	Asserted for one clock cycle (synchronous to CLK) when a credit error is detected.	High
	ESCERR	Output	Asserted for one clock cycle (synchronous to CLK) when an escape error is detected.	High
	PARERR	Output	Asserted for one clock cycle (synchronous to CLK) when a parity error is detected.	High
	RXICHARAV	Output	Asserted when one or more characters are available in the receiver FIFO. Synchronous to CLK.	High
	RXICHARCNT [11:0]	Output	Number of characters present in the receiver FIFO. Synchronous to CLK.	-
	RXICHAR[8:0]	Output	Character read from the receiver FIFO. Valid the clock cycle following the assertion of the RXIREAD input. Synchronous to CLK. Bit 0 corresponds to the SpaceWire control bit and when set to 1 only bits (2:1) are valid (EOP or EEP).	-
	TXICHARCNT [11:0]	Output	Number of characters present in the transmitter FIFO. Synchronous to CLK.	-
	TXIFULL	Output	Asserted when the transmitter FIFO is full. Synchronous to CLK.	High
	TXIEMPTY	Output	Asserted when the transmitter FIFO is empty. Synchronous to CLK.	High
	TXIFIFORST	Output	Asserted when the transmitter FIFO reset is active. If set, no new characters shall be written to the FIFO. Synchronous to CLK.	High
	TICKIN_DONE	Output	Asserted for one cycle when the time-code has been transmitted resulting from an assertion of TICKIN. When TICKIN_DONE is asserted the TICKIN should be deasserted the same cycle if it is not desired that another time-code should be deasserted.	High
	TICKIN_BUSY	Output	Set while the time-code is being transmitted.	High
	TICKOUT	Output	Time-code tickout output. Asserted for one clock cycle (synchronous to CLK) when a time-code has been received. When asserted the TIMEOUT output is valid.	High
	TIMEOUT[7:0]	Output	Time-code output. Synchronous to CLK.	-
	MERROR	Output	Asserted one clock cycle when an EDAC memory error is detected.	High
	OCREDCNT[5:0]	Output	Value of the outstanding credit counter.	-
	CREDCNT[5:0]	Output	Value of the incoming credit counter.	-
	POWERDOWN	Output	Power-down indicator.	High
	POWERDOWNRX	Output	RX power-down indicator.	High
	RXMERROR	Output	Asserted one clock cycle when an EDAC memory error is detected in the RX FIFO.	High
	TXMERROR	Output	Asserted one clock cycle when an EDAC memory error is detected in the TX FIFO.	High
	RXRST	Output	Internal reset generated by the transmitter for synchronization purpose between transmitter and both receiver channels. It shall be used to generate the asynchronous and synchronous receiver resets.	Low

80.10 Signal definitions and reset values

The signals and their reset values are described in table 1439.

*Table 1439.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
spw_clk	Input	Transmitter default run-state clock	Rising edge	-
spw_rxd	Input, LVDS	Data input, positive	High	-
spw_rxdn	Input, LVDS	Data input, negative	Low	-
spw_rxs	Input, LVDS	Strobe input, positive	High	-
spw_rxsn	Input, LVDS	Strobe input, negative	Low	-
spw_txd	Output, LVDS	Data output, positive	High	Logical 0
spw_txdn	Output, LVDS	Data output, negative	Low	Logical 1
spw_txs	Output, LVDS	Strobe output, positive	High	Logical 0
spw_txsn	Output, LVDS	Strobe output, negative	Low	Logical 1

80.11 Timing

The timing waveforms and timing parameters are shown in figure 212 and are defined in table 1440.

The SpaceWire jitter and skew timing waveforms and timing parameters are shown in figure 213 and are defined in table 1441.

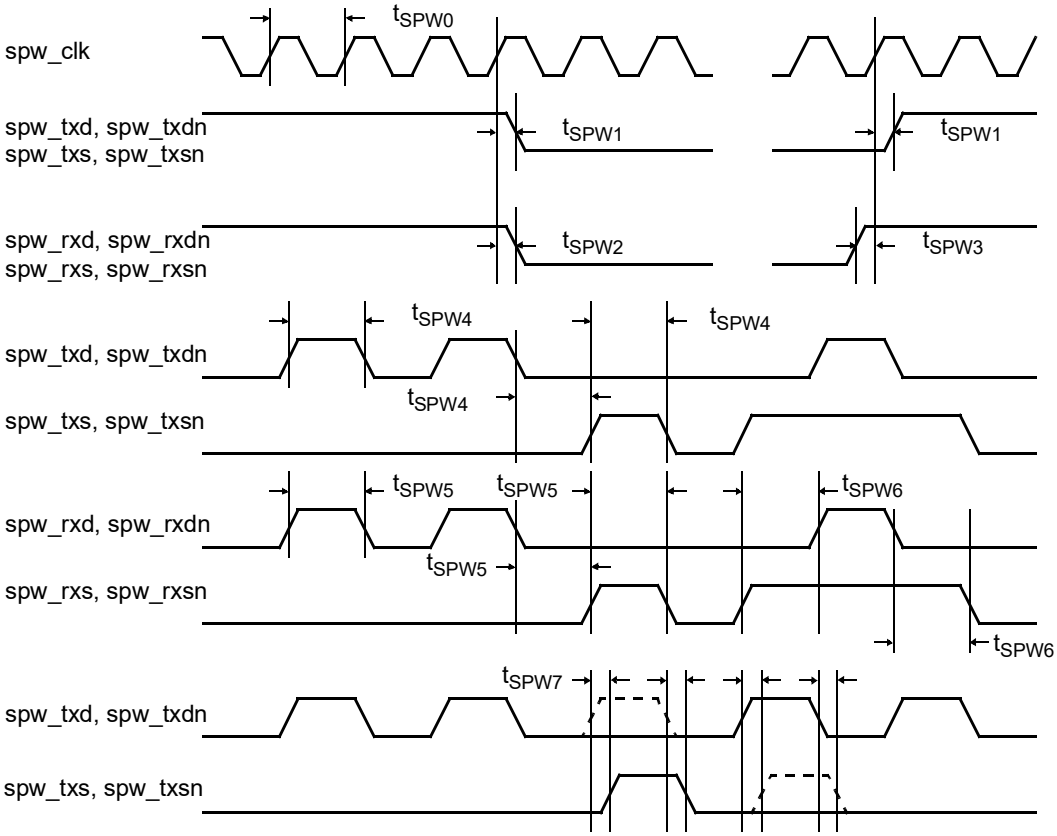


Figure 212. Timing waveforms

*Table 1440.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{SPW0}	transmit clock period	-	TBD	-	ns
t_{SPW1}	clock to output delay	rising *spw_clk* edge	TBD	TBD	ns
t_{SPW2}	input to clock hold	-	-	-	not applicable
t_{SPW3}	input to clock setup	-	-	-	not applicable
t_{SPW4}	output data bit period	-	-		*clk* periods
		-	t_{SPW0} - TBD	t_{SPW0} +TBD	ns
t_{SPW5}	input data bit period	-	TBD	-	ns
t_{SPW6}	data & strobe edge separation	-	TBD	-	ns
t_{SPW7}	data & strobe output skew	-	-	TBD	ns

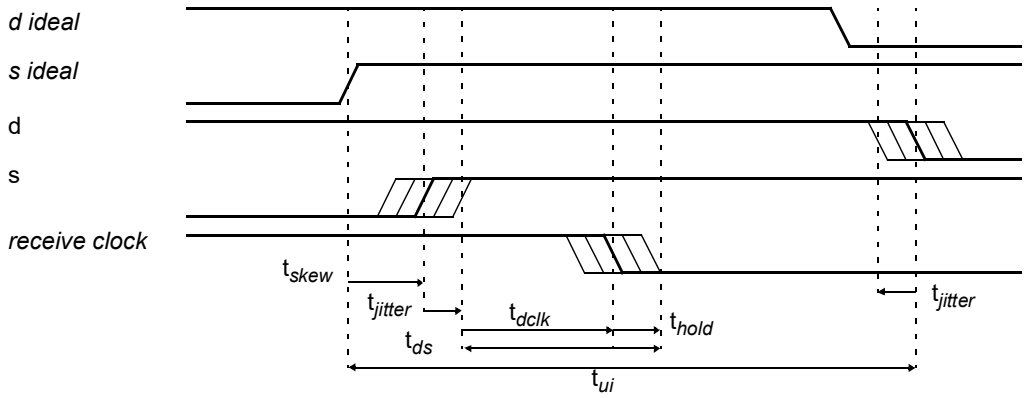


Figure 213. Skew and jitter timing waveforms

*Table 1441.*Skew and jitter timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{skew}	skew between data and strobe	-	-	TBD	ns
t_{jitter}	jitter on data or strobe	-	-	TBD	ns
t_{ds}	minimum separation between data and strobe edges	-	TBD	-	ns
t_{dclk}	delay from edge of data or strobe to the receiver flip-flop	-	-	TBD	ns
t_{hold}	hold timer on receiver flip-flop	-	TBD	-	ns
t_{ui}	unit interval (bit period)	-	TBD	-	ns

80.12 Library dependencies

Table 1442 shows libraries used when instantiating the core (VHDL libraries).

*Table 1442.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	SPACEWIRE	Signals, component	Component and record declarations.

80.13 Instantiation

This example shows how the core can be instantiated.

Normally di, si, do and so should be connected to input and output pads configured with LVDS drivers. How this is done is technology dependent.

The core in the example is configured with non-ft technology mapped FIFOs of size 64. It uses DDR sampling on the input and the combined sampling/transmitter frequency (spw_rxtxclk) is 50MHz.

The minimum amount of signals that need to be driven to put the codec in a deterministic initial state is shown.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.tech.all;
library gaisler;
use gaisler.spacewire.all;
```

```
entity spacewire_ex is
  port (
    clk           : in  std_ulogic;
    rstn          : in  std_ulogic;

    -- spacewire signals
    spw_rxdp      : in  std_ulogic;
    spw_rxdn      : in  std_ulogic;
    spw_rxsp      : in  std_ulogic;
    spw_rxsn      : in  std_ulogic;
    spw_txdp      : out std_ulogic;
    spw_txdn      : out std_ulogic;
    spw_txsp      : out std_ulogic;
    spw_txsn      : out std_ulogic;

    spw_rxtxclk   : in  std_ulogic;
    spw_rxclkn    : in  std_ulogic;
    testen        : in  std_ulogic;
    testrst       : in  std_ulogic
    );
end;

architecture rtl of spacewire_ex is

  -- Spacewire signals
  signal lii       : grspw_codec_in_type;
  signal lio       : grspw_codec_out_type;
  signal di        : std_ulogic;
  signal si        : std_ulogic;
  signal rxclko    : std_ulogic;

  -- Internal resets
  signal mrst      : std_ulogic;
  signal rxasyncrst: std_ulogic;
  signal rxsyncrst0: std_ulogic;
  signal rxsyncrst1: std_ulogic;
  signal txsyncrst : std_ulogic;

begin

  spw_phy0 : grspw2_phy
    generic map(
      scantest   => 0,
      tech       => memtech,
      input_type => 3)
    port map(
      rstn       => rstn,
      rxclki     => spw_rxtxclk,
      rxclkin    => spw_rxclkn,
      nrxclki    => spw_rxtxclk,
      di         => di,
      si         => si,
      do         => lii.d(1 downto 0),
      dov        => lii.dv(1 downto 0),
      dconnect   => lii.dconnect(1 downto 0),
      dconnect2  => lii.dconnect2(1 downto 0),
      dconnect3  => lii.dconnect3(1 downto 0),
      rxclko     => rxclko);

  -- Internal reset generators (TX, RX0 and RX1 clock domains)

  -- CLK domain (synchronous reset)
  AMBA_rst : rstgen
  port map (rstn, clk, vcc, mrst, open);

  -- TX domain (synchronous reset)
  txrst : rstgen
  port map (rstn, spw_rxtxclk, vcc, txsyncrst, open);

  -- RX domain (asynchronous reset)
  rxasyncrst <= lio.rxrst;
```

```
-- RX domain (synchronous reset)
rxsyncrst0 : rstgen
port map (lio.rxrst, rxclk0, vcc, rxsyncrst0, open);
rxsyncrst1 : rstgen
port map (lio.rxrst, rxclk1, vcc, rxsyncrst1, open);

codec : grspw_codec
  generic map(
    ports        => 1,
    input_type   => 0,
    output_type  => 3,
    rxtx_sameclk => 1,
    fifosize     => 64,
    tech         => 11,
    scantest     => 0,
    techfifo     => 1,
    ft           => 0)
  port map(
    rst          => mrst,
    clk          => clk,
    rxasyncrst   => rxasyncrst,
    rxsyncrst0   => rxsyncrst0,
    rxclk0       => rxclk0,
    rxsyncrst1   => rxsyncrst1,
    rxclk1       => rxclk1,
    txsyncrst    => txsyncrst,
    txclk        => spw_rxtxclk,
    txclkn       => spw_rxtxclk,
    testen       => testen,
    testrst      => testrst,
    lii          => lii,
    lio          => lio);

lii.linkdisabled <= '1';
lii.linkstart    <= '0';
lii.autostart    <= '0';
lii.portsel      <= '0';
lii.noportforce  <= '0';
lii.idivisor     <= conv_std_logic_vector(4, 8);
lii.rdivisor     <= conv_std_logic_vector(0, 8);

lii.rxiread      <= '0';
lii.txiwrite     <= '0';
lii.txichar      <= (others => '0');
lii.txififorst   <= '0';
lii.tickin       <= '0';

spw_rxd_pad : inpad_ds generic map (padtech, lvds, x25v)
  port map (spw_rxdp, spw_rxdn, di);
spw_rxs_pad : inpad_ds generic map (padtech, lvds, x25v)
  port map (spw_rxsp, spw_rxsn, si);
spw_txd_pad : outpad_ds generic map (padtech, lvds, x25v)
  port map (spw_txdp, spw_txdn, lio.d(0), gnd(0));
spw_txs_pad : outpad_ds generic map (padtech, lvds, x25v)
  port map (spw_txsp, spw_txsn, lio.s(0), gnd(0));
...
```

81 GRSPW_CODEC_GEN - GRSPW_CODEC wrapper with Std_Logic interface

81.1 Overview

The GRSPW_CODEC_GEN wrapper provides an interface to the GRSPW_CODEC core only using Std_Logic signals instead of GRLIB records. The GRLIB AMBA plug and play extensions have also been removed. This document describes the signal interface and how they map to the signal names in GRLIB. An example instantiation of the core can be found at the end of this document.

81.2 Signal descriptions

Table 1438 shows the interface signals of the core (VHDL ports).

Table 1443. Signal descriptions

Signal name	Type	Function	Active	GRLIB signal name
RST	Input	AMBA reset	Low	RST
CLK	Input	System (AMBA) Clock.	-	CLK
RXASYNCRST	Input	Receiver asynchronous reset for both ports 0 and 1	Low	RXASYNCRST
RXSYNCRST0	Input	Receiver synchronous reset for port 0	Low	RXSYNCRST0
RXCLK0	Input	Receiver clock for port 0.	-	RXCLK0
RXSYNCRST1	Input	Receiver synchronous reset for port 1	Low	RXSYNCRST1
RXCLK1	Input	Receiver clock for port 1. Unused if the VHDL generic ports is 2.	-	RXCLK1
TXSYNCRST	Input	Transmitter synchronous reset	Low	TXSYNCRST
TXCLK	Input	Transmitter default run-state clock	-	TXCLK
TXCLKN	Input	Transmitter inverted default run-state clock. Only used in DDR transmitter mode for technologies not supporting local generation of inverted clock.	-	TXCLKN
TESTEN	Input	Scan test enable	High	TESTEN
TESTRST	Input	Scan test reset	Low	TESTRST
D[3:0]	Input	Data input. Should be connected to the GRSPW2_PHY. Each bit is valid when the bit at the corresponding index in the DV signal is asserted. Bits (1:0) are used for port 0 and bits (3:2) are used for port 1 if enabled. If only one bit is received during one clock cycle then only the bit at the lower index is valid. If both bits are valid then the one at the lower index was the one received first. Bits (1:0) should be synchronous to RXCLK0 and bits (3:2) to RXCLK1.	-	LII.D
DV[3:0]	Input	Data valid qualifier for the D input.	High	LII.DV
DCONNECT[3:0]	Input	Disconnect reset. When asserted the corresponding disconnect counter will be reset. The counters connected to bits (1:0) apply to port 0 while (3:2) apply to port 1 if enabled.	Low	LII.DCONNECT
DCONNECT2 [3:0]	Input	SpaceWire Disconnect reset. Copy of DCONNECT (triplication of the asynchronous resets logic).	Low	LII.DCONNECT2
DCONNECT3 [3:0]	Input	SpaceWire Disconnect reset. Copy of DCONNECT (triplication of the asynchronous resets logic).	Low	LII.DCONNECT3
LINKDISABLED	Input	Disables the SpaceWire link	High	LII.LINKDISABLED

Table 1443. Signal descriptions

Signal name	Type	Function	Active	GRLIB signal name
LINKSTART	Input	Starts the SpaceWire link	High	LII.LINKSTART
AUTOSTART	Input	Enables the autostart feature for the SpaceWire link	High	LII.AUTOSTART
PORTSEL	Input	Selects the active port if the NOPORTFORCE signal is set to 0 and the core is configured with dual ports.	-	LII.PORTSEL
NOPORTFORCE	Input	Disables forced portselection using the PORTSEL signal and lets the core automatically select the active link.	High	LII.NOPORTFORCE
RDIVISOR[7:0]	Input	Clock divisor value used for generating the transmit frequency from the txclk input in run-state. Bit 0 is the least significant.	-	LII.RDIVISOR
IDIVISOR[7:0]	Input	Clock divisor value used for generating the transmit frequency from the txclk input during initialization (started and connecting states). This value is also used to generate the disconnect timing and the FSM timeouts (6.4 us and 12.8 us). Bit 0 is the least significant.	-	LII.IDIVISOR
RXIREAD	Input	Receiver FIFO read. Assert for one cycle (synchronous to CLK) and the next character will be available on the RXICHAR output the next clock cycle if available in the FIFO.	High	LII.RXIREAD
RXIFIFORST	Input	Empty the receiver FIFO. When asserted for one cycle (synchronous to CLK) all characters currently present in the FIFO will be discarded. The reset will have taken on the second cycle after the reset. The first cycle after characters may still be present.	High	LII.RXIFIFORST
TXIWRITE	Input	Transmitter FIFO write. Assert for one cycle (synchronous to CLK) to write the character on the TXICHAR input into the FIFO on the rising edge of the clock.	High	LII.TXIWRITE
TXICHAR[8:0]	Input	Transmit character input to FIFO. Characters are transmitted in the order they are written into the FIFO. Bit 0 is the SpaceWire control bit. When set to 1 only bits (2:1) are transmitted (EOP or EEP). Bits are transmitted in order beginning at the lowest index.	-	LII.TXICHAR
TXIFIFORST	Input	Empty the transmitter FIFO. When asserted for one cycle (synchronous to CLK) all characters currently present in the FIFO will be discarded. New characters shall not be written to the transmitter FIFO the same cycle or the cycle after TXIFIFORST is asserted.	High	LII.TXIFIFORST
TICKIN	Input	Time counter tick input. Should be asserted one clock cycle (synchronous to CLK) to transmit a time-code. If the FSM is not in run-state no time-code will be transmitted and the tick will not be registered. The TICKIN_DONE output can be used to verify that the time-code has been transmitted,	High	LII.TICKIN
TIMEIN[7:0]	Input	Time-code input. Transmitted in order starting at the lowest index when tickin is asserted.	-	LII.TIMEIN

Table 1443. Signal descriptions

Signal name	Type	Function	Active	GRLIB signal name
DO[3:0]	Output	Data output. The meaning of the bits depends on the value of the *output_type* generic. When output_type=0 or 1, DO[0] is the data output for port 0, DO[1] is the data output for port 1, and DO[3:2] are unused. In single-port configurations DO[3:1] are unused. When output_type=2 (external UT200SpW-PHY01 SpaceWire transceiver), DO[1:0] is the output for port 0 and DO[3:2] is the output for port 1. In single-port configurations DO[3:2] are unused.	-	LIO.DO
SO[3:0]	Output	Strobe output. Bit indexes are used correspondingly to the DO output.	-	LIO.SO
STATE[2:0]	Output	Current link interface FSM state. 0=error-reset, 1=error-wait, 2=ready, 3=started, 4=connecting, 5=run.	-	LIO.STATE
ACTPORT	Output	0=port 0 is active and 1=port 1 is active. Unused if the second port is not enabled.	-	LIO.ACTPORT
DCONNECTERR	Output	Asserted for one clock cycle (synchronous to CLK) when a disconnect error is detected.	High	LIO.DCONNECTERR
CREDERR	Output	Asserted for one clock cycle (synchronous to CLK) when a credit error is detected.	High	LIO.CREDERR
ESCERR	Output	Asserted for one clock cycle (synchronous to CLK) when an escape error is detected.	High	LIO.ESCERR
PARERR	Output	Asserted for one clock cycle (synchronous to CLK) when a parity error is detected.	High	LIO.PARERR
RXICHARAV	Output	Asserted when one or more characters are available in the receiver FIFO. Synchronous to CLK.	High	LIO.RXICHARAV
RXICHARCNT [11:0]	Output	Number of characters present in the receiver FIFO. Synchronous to CLK.	-	LIO.RXICHARCNT
RXICHAR[8:0]	Output	Character read from the receiver FIFO. Valid the clock cycle following the assertion of the RXIREAD input. Synchronous to CLK. Bit 0 corresponds to the SpaceWire control bit and when set to 1 only bits (2:1) are valid (EOP or EEP).	-	LIO.RXICHAR
TXICHARCNT [11:0]	Output	Number of characters present in the transmitter FIFO. Synchronous to CLK.	-	LIO.TXICHARCNT
TXIFULL	Output	Asserted when the transmitter FIFO is full. Synchronous to CLK.	High	LIO.TXIFULL
TXIEMPTY	Output	Asserted when the transmitter FIFO is empty. Synchronous to CLK.	High	LIO.TXIEMPTY
TXIFIFORSTACT	Output	Transmitter FIFO reset active. Used in router when writing EEP after reset.	High	LIO.TXIFIFORST
TICKIN_DONE	Output	Asserted for one cycle when the time-code has been transmitted resulting from an assertion of TICKIN. When TICKIN_DONE is asserted the TICKIN should be deasserted the same cycle if it is not desired that another time-code should be deasserted.	High	LIO.TICKIN_DONE
TICKOUT	Output	Time-code tickout output. Asserted for one clock cycle (synchronous to CLK) when a time-code has been received. When asserted the TIMEOUT output is valid.	High	LIO.TICKOUT

Table 1443. Signal descriptions

Signal name	Type	Function	Active	GRLIB signal name
TIMEOUT[7:0]	Output	Time-code output. Synchronous to CLK.	-	LIO.TIMEOUT
MERROR	Output	Asserted one clock cycle when an EDAC memory error is detected.	High	LIO.MERROR
RXRST	Output	Internal reset generated by the transmitter for synchronization purpose between transmitter and both receiver channels. It shall be used to generate the asynchronous and synchronous receiver resets.	Low	LIO.RXRST

81.3 Instantiation

This example shows how the core can be instantiated. It also includes the reset generators.

Normally di, si, do and so should be connected to input and output pads configured with LVDS drivers. How this is done is technology dependent.

The core in the example is configured with non-ft technology mapped FIFOs of size 64. It uses DDR sampling on the input and the combined sampling/transmitter frequency (spw_rxtxclk) is 50MHz.

The minimum amount of signals that need to be driven to put the codec in a deterministic initial state is shown.

```
library grlib;
use grlib.tech.all;
library gaisler;
use gaisler.spacewire.all;

entity spacewire_ex is
  port (
    clk           : in  std_ulogic;
    rstn          : in  std_ulogic;
    -- spacewire signals
    spw_rxdp      : in  std_ulogic;
    spw_rxdn      : in  std_ulogic;
    spw_rxsp      : in  std_ulogic;
    spw_rxsn      : in  std_ulogic;
    spw_txdp      : out std_ulogic;
    spw_txdn      : out std_ulogic;
    spw_txsp      : out std_ulogic;
    spw_txsn      : out std_ulogic;
    spw_rxtxclk   : in  std_ulogic;
    spw_rxclkn    : in  std_ulogic;
    testen        : in  std_ulogic;
    testrst       : in  std_ulogic
    );
end;

architecture rtl of spacewire_ex is

  signal di               : std_ulogic;
  signal si               : std_ulogic;
  signal d                : std_logic_vector(3 downto 0);
  signal dv               : std_logic_vector(3 downto 0);
  signal dconnect         : std_logic_vector(3 downto 0);
  signal dconnect2        : std_logic_vector(3 downto 0);
  signal dconnect3        : std_logic_vector(3 downto 0);
  signal do               : std_logic_vector(3 downto 0);
  signal so               : std_logic_vector(3 downto 0);
  signal rxclko           : std_ulogic;
  signal linkdisabled     : std_ulogic;
  signal linkstart        : std_ulogic;
  signal autostart        : std_ulogic;
```

```
  signal portsel          : std_ulogic;
  signal noportforce      : std_ulogic;
  signal rdivisor         : std_logic_vector(7 downto 0);
  signal idivisor         : std_logic_vector(7 downto 0);
  signal rxiread          : std_ulogic;
  signal rxififorst       : std_ulogic;
  signal txiwrite         : std_ulogic;
  signal txichar          : std_logic_vector(8 downto 0);
  signal txififorst       : std_ulogic;
  signal tickin           : std_ulogic;
  signal timein           : std_logic_vector(7 downto 0);
  signal do               : std_logic_vector(3 downto 0);
  signal so               : std_logic_vector(3 downto 0);
  signal state            : std_logic_vector(2 downto 0);
  signal actport          : std_ulogic;
  signal dconnecterr      : std_ulogic;
  signal crederr          : std_ulogic;
  signal escerr           : std_ulogic;
  signal parerr           : std_ulogic;
  signal rxicharav        : std_ulogic;
  signal rxicharcnt       : std_logic_vector(11 downto 0);
  signal rxichar          : std_logic_vector(8 downto 0);
  signal txicharcnt       : std_logic_vector(11 downto 0);
  signal txifull          : std_ulogic;
  signal txiempty         : std_ulogic;
  signal tickin_done      : std_ulogic;
  signal tickout          : std_ulogic;
  signal timeout          : std_logic_vector(7 downto 0);
  signal merror           : std_ulogic;

  -- Internal resets
  signal mrst      : std_ulogic;
  signal rxasyncrst: std_ulogic;
  signal rxsyncrst0: std_ulogic;
  signal rxsyncrst1: std_ulogic;
  signal txsyncrst : std_ulogic;

begin

  spw_rxd_pad : inpad_ds generic map (tech => padtech, level => lvds)
                            port map (spw_rxd, spw_rxdn, di);
  spw_rxs_pad : inpad_ds generic map (tech => padtech, level => lvds)
                            port map (spw_rxs, spw_rxsn, si);

  spw_phy0 : grspw2_phy
    generic map(
      scantest   => 0,
      tech       => memtech,
      input_type => 3)
    port map(
      rstn       => rstn,
      rxclki     => spw_rxtxclk,
      rxclkin    => spw_rxclkn,
      nrxclki    => spw_rxtxclk,
      di         => di,
      si         => si,
      do         => d(1 downto 0),
      dov        => dv(1 downto 0),
      dconnect   => dconnect(1 downto 0),
      dconnect2  => dconnect2(1 downto 0),
      dconnect3  => dconnect3(1 downto 0),
      rxclko     => rxclk0);

  d(3 downto 2)         <= "00"; -- For second port
  dv(3 downto 2)        <= "00"; -- For second port
  dconnect(3 downto 2)  <= "00"; -- For second port
  dconnect2(3 downto 2) <= "00"; -- For second port
  dconnect3(3 downto 2) <= "00"; -- For second port

  -- Internal reset generators (TX, RX0 and RX1 clock domains)
```

```
-- CLK domain (synchronous reset)
AMBA_rst : rstgen
port map (rstn, clk, vcc, mrst, open);

-- TX domain (synchronous reset)
txrst : rstgen
port map (rstn, spw_rxtxclk, vcc, txsyncrst, open);

-- RX domain (asynchronous reset)
rxasyncrst <= rxrst;

-- RX domain (synchronous reset)
rxsyncrst0 : rstgen
port map (rxrst, rxclk0, vcc, rxsyncrst0, open);
rxsyncrst1 : rstgen
port map (rxrst, rxclk1, vcc, rxsyncrst1, open);

codec : grspw_codec_gen
  generic map(
    ports        => 1,
    input_type   => 0,
    output_type  => 3,
    rxtx_sameclk => 1,
    fifosize     => 64,
    tech         => 11,
    scantest     => 0,
    techfifo     => 1,
    ft           => 0)
  port map(
    rst           => mrst,
    clk           => clk,
    rxasyncrst    => rxasyncrst,
    rxsyncrst0    => rxsyncrst0,
    rxclk0        => rxclk0,
    rxsyncrst1    => rxsyncrst1,
    rxclk1        => rxclk1,
    txsyncrst     => txsyncrst,
    txclk         => spw_rxtxclk,
    txclkn        => spw_rxtxclk,
    testen        => testen,
    testrst       => testrst,
    d             => d,
    dv            => dv,
    dconnect      => dconnect,
    dconnect2     => dconnect2,
    dconnect3     => dconnect3,
    do            => do,
    so            => so,
    linkdisabled  => linkdisabled,
    linkstart     => linkstart,
    autostart     => autostart,
    portsel       => portsel,
    noportforce   => noportforce,
    rdivisor      => rdivisor,
    idivisor      => idivisor,
    state         => state,
    actport       => actport,
    dconnecterr   => dconnecterr,
    crederr       => crederr,
    escerr        => escerr,
    parerr        => parerr,
    rxicharav     => rxicharav,
    rxicharcnt    => rxicharcnt,
    rxichar       => rxichar,
    rxiread       => rxiread,
    rxififorst    => rxififorst,
    txicharcnt    => txicharcnt,
    txifull       => txifull,
    txiempty      => txiempty,
    txiwrite      => txiwrite,
    txichar       => txichar,
```

```
    txififorst    => txififorst,
    txififorstact => txififorstact,
    tickin        => tickin,
    timein        => timein,
    tickin_done   => tickin_done,
    tickout       => tickout,
    timeout       => timeout,
    merror        => merror,
    rxrst         => rxrst);

linkdisabled <= '1';
linkstart    <= '0';
autostart    <= '0';
portsel      <= '0';
noportforce  <= '0';
idivisor     <= conv_std_logic_vector(4, 8);
rdivisor     <= conv_std_logic_vector(0, 8);

rxiread      <= '0';
txiwrite     <= '0';
txichar      <= (others => '0');
txififorst   <= '0';
tickin       <= '0';
timein       <= (others => '0');

spw_rxd_pad : inpad_ds generic map (padtech, lvds, x25v)
  port map (spw_rxdp, spw_rxdn, di);
spw_rxs_pad : inpad_ds generic map (padtech, lvds, x25v)
  port map (spw_rxsp, spw_rxsn, si);
spw_txd_pad : outpad_ds generic map (padtech, lvds, x25v)
  port map (spw_txdp, spw_txdn, do(0), gnd(0));
spw_txs_pad : outpad_ds generic map (padtech, lvds, x25v)
  port map (spw_txsp, spw_txsn, so(0), gnd(0));
...
```

82 GRSPWROUTER - SpaceWire router

82.1 Overview

The SpaceWire router core implements a SpaceWire routing switch as defined in the ECSS-E-ST-50-12C standard. It supports from 2 to 31 ports in addition to the mandatory configuration port where each port (except the configuration port) can be individually configured to be SpaceWire links, FIFO interfaces or AMBA interfaces. The AMBA ports are limited to a maximum of 16 in a single router. The configuration port provides an RMAP target, and an optional AMBA AHB slave interface, both used for accessing internal configuration and status registers.

Among the features supported by the router are: group adaptive routing, packet distribution, system time-distribution, distributed interrupts, port timers to recover from deadlock situations, and SpaceWire-D [SPWD] packet truncation based time-slot violations. The router is designed to be technology independent and configurable. This enables it to be implemented in all technologies supported by the GRLIB IP library and also to remove optional features to save area.

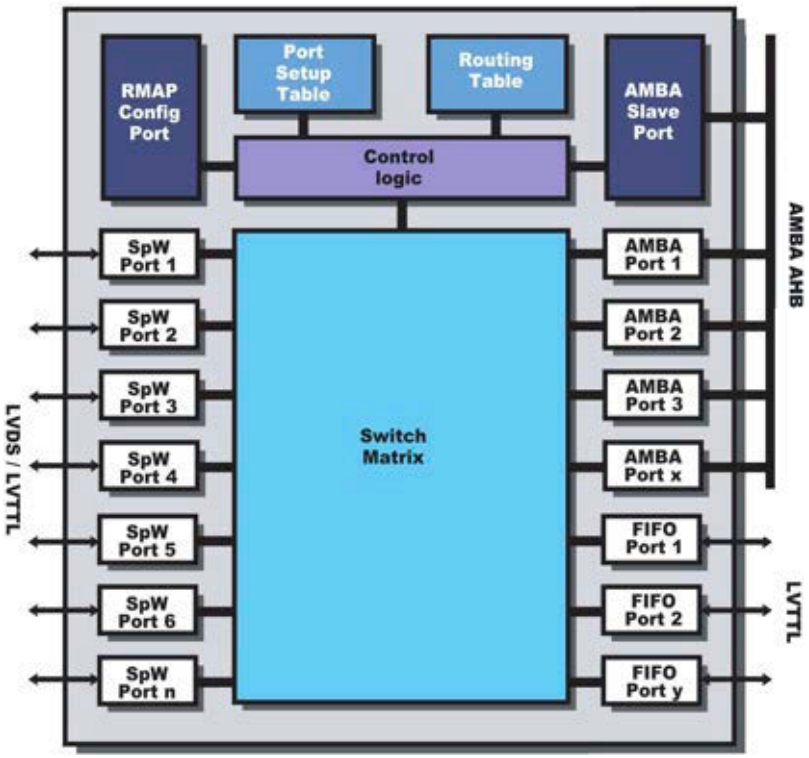


Figure 214. Block diagram

82.2 Operation

The switch matrix can connect any input port to any output port. Access to each output port is arbitrated using a round-robin arbitration scheme based on the address of the incoming packet. A single routing-table is used for the whole router, where access to the table is arbitrated using a round-robin scheme based on the input port number.

All the ports regardless of their type have the same interface to the switch matrix and behave in the same manner. The difference in behavior is on the external side of the port. The SpaceWire ports provide standard SpaceWire link interfaces using either on- or off-chip LVDS. The FIFO interfaces store characters in two FIFOs which are accessed using 9-bit wide data paths with read/write signals. Lastly

the AMBA ports transfer characters from and to an AHB bus using DMA. The four different port types are described in further detail in sections 82.3, 82.4, 82.5 and 82.6.

82.2.1 Endianness

The core is designed for big-endian systems.

82.2.2 Port numbering

The ports are numbered in the following order: configuration port, SpW ports, AMBA ports, FIFO ports. The configuration port is always present and has number 0. If SpW ports are present in the router they are numbered starting from number 1. If AMBA ports are present they are numbered starting from the last SpW ports. If no SpW ports are present the AMBA ports start at number 1. Lastly, the FIFO ports are numbered starting after the last AMBA port, SpW port or at number 1 depending on if AMBA ports and SpW ports are present respectively.

For example if 7 SpW ports and 4 FIFO ports are included in the router they will have port numbers 1-7 and 8-11. If 16 SpW ports, 2 AMBA ports and 7 FIFO ports are included they have port numbers 1-16, 17-18 and 19-25 respectively.

82.2.3 Routing table

A single routing table is provided. The access to this routing table is arbitrated using a round-robin arbiter with each port being of equal priority. The operation is pipelined and one lookup can be done each cycle. This way the maximum latency is equal to the number of ports in the router minus one. The impact on throughput should be negligible provided that packets are not incoming at the same time. The probability for this is higher when the traffic only consist of very small packets sent continuously (the average size being about the same as the number of ports). This should be a very uncommon case. Latency is still bounded and probably negligible in comparison to other latencies in most systems. The benefit is a reduced area enabling the router to be implemented with a higher number of ports on many FPGA technologies.

Since the latency for the lookup is very small and deterministic there is not much to gain by having configurable priorities for this. Priorities are instead used for arbitrating packets contending for an output port as described in the next section.

The routing table and all the configuration registers are configured through an RMAP target or an optional AHB slave interface which use the same routing table as the logic handling packet traffic. They do not introduce any extra latency for packets, since the configuration accesses have lower priority than the packet traffic and thus are only allowed access on cycles when no lookup is needed for packets. This can slow down configuration accesses but they are probably mostly done before packet traffic starts and very seldom afterwards. The routing table is split into two parts, one which controls the port mapping for the address (RTR.RTPMAP registers), and one which controls properties for the address, such as priority and header deletion (RTR.RTACTRL registers).

82.2.3.1 Port mapping

For both physical and logical addresses it is possible to configure which port(s) the incoming packet should be routed to. This is done by programming the corresponding RTR.RTPMAP register. The RTR.RTPMAP registers also controls whether or not group adaptive routing or packet distribution should be used for the incoming packet. The RTR.RTPMAP registers are not initialized after reset / power-up. For physical addresses this has the effect that the incoming packet is routed to the port that matches the address in the packet, without any group adaptive routing or packet distribution. For logical addresses, an uninitialized RTR.RTPMAP register (or if the RTR.RTPMAP.PE field has been written with all zeros) has the effect that the incoming packet is spilled. See table 1479 in section 82.7 for more details.

82.2.3.2 Address control

For both physical and logical addresses it is possible to configure the priority, and to enable the spill-if-not-ready feature (explained in section 82.2.9). For logical addresses it is also possible to enable / disable the address, and enable / disable header deletion. Physical addresses are always enabled, and always have header deletion enabled, as specified by ECSS-E-ST-50-12C [SPW]. This configuration for an address is done by programming the corresponding RTR.RTACTRL register. Logical addresses are disabled after reset / power-up. An incoming packet with a disabled logical address is spilled. See table 1480 in section 82.7 for details.

82.2.4 Output port arbitration

Each output port is arbitrated individually based on the address of the incoming packet, using two priority levels, with round-robin at each level. Each physical address and logical address can be configured in the routing table (RTR.RTACTRL register) to be either high or low priority. Priority assignments can have large impact on the delays for packets, because packets can be large and the speed of the data consumer and link itself may not be known. This should therefore be considered when assigning priorities.

82.2.5 Group adaptive routing

Group adaptive routing can be used to allow incoming packets to be sent on different output ports depending on which of the output ports that are currently ready. It can be enabled for both physical and logical addresses, and is configured by programming the corresponding RTR.RTPMAP register.

When a packet arrives, and group adaptive routing is enabled for the packet's address, the router looks at the group of ports selected by the corresponding RTR.RTPMAP register and transmits the packet on the port with the lowest index that is currently ready. Ready in this context means that the port's link interface is in run-state and currently not sending any other packet. If none of the selected output ports are ready, the incoming packet will either be spilled or transmitted on the first port that becomes ready. The action taken depends on the setting of the input port's data character timer (see section 82.2.14), the spill-if-not-ready feature for the address (see section 82.2.9), and the link-start-on-request feature for the output ports (see section 82.2.12). See table 1479 in section 82.7 for details on how to enable and configure group adaptive routing.

82.2.6 Packet distribution

Packet distribution can be used to implement multicast and broadcast addresses, and can be enabled for both physical and logical addresses. Packet distribution is enabled and configured by programming the corresponding RTR.RTPMAP register.

When a packet arrives, and packet distribution is enabled for the packet's address, the router looks at the group of ports selected by the corresponding RTR.RTPMAP register. If all of the selected ports are ready, the packet is transmitted on all the ports. Ready in this context means that the port's link interface is in run-state and currently not sending any other packet. If one or more of the selected ports are not ready, the incoming packet will either be spilled or transmitted once all ports are ready. The action taken depends on the setting of the input port's data character timer (section 82.2.14), the spill-if-not-ready feature for the address (section 82.2.9), and the link-start-on-request feature for the output ports (section 82.2.12). See table 1479 in section 82.7 for details on how to enable and configure packet distribution.

82.2.7 Port disable

A port can be disabled for data traffic by setting the corresponding RTR.PCTRL.DI bit. Incoming packets on a disabled port are silently spilled, and no packets are routed to a disabled port. A disabled port will not be included in any group used for group adaptive routing or packet distribution, even if the corresponding bit in that address' RTR.RTPMAP.PE field is set. When routing packets that are

incoming on other ports, the router will simply behave as if the disabled port did not exist. The RTR.PCTRL.DI bit only affects routing of data, thus the transmission and reception of time-codes and distributed interrupt codes are not affected.

The link interface for a SpaceWire port is not affected solely by the RTR.PCTRL.DI bit. However, note that the RTR.PCTRL.DI bit, together with the RTR.PCTRL.LD bit, is used to clock-gate a SpaceWire port (see section 82.2.24).

82.2.8 Static routing

The router supports a feature called static routing, which can be enabled individually per port. When enabled, all incoming packets on the port are routed based on the physical address specified in the port's RTR.PCTRL2.SC field, and the setting of the corresponding RTR.PCTRL2.SC bit, instead of the address in the packets. Header deletion is not used for the incoming packets when static routing is enabled, which means that the first byte of the packets are always sent to the output port as well. Static routing to port 0 is not allowed, and will generate an invalid address error if attempted.

The STATICROUTEEN signal works as a global enable / disable for the static routing feature during reset. The feature is enabled if the signal is high during reset, and disabled if the signal is low during reset. The RTR.RTRCFG.SR bit shows if the static routing feature is globally enabled or disabled.

Note that when static routing is enabled for a port, it is not possible to access the configuration port from the same port.

82.2.9 Spill-if-not-ready

The spill-if-not-ready feature can be enabled individually for each physical and logical address by configuring the corresponding RTR.RTACTRL.SR bit. When enabled, an incoming packet is spilled if the selected output port's link interface is not in run-state. If group adaptive routing is used for the incoming packet then the packet is only spilled if none of the ports in the group is in run-state. If packet distribution is used for the incoming packet then the packet is spilled unless the link interfaces for all selected output ports are in run-state. The spill-if-not-ready feature has priority over the incoming port's data character timer (section 82.2.14) and the output port's link-start-on-request feature (section 82.2.12). This means that if the spill-if-not-ready feature is enabled, the packet is spilled before the timer starts, and the link-start-on-request feature will never be activated.

82.2.10 Self addressing

Self addressing occurs when a selected output port for a packet is the same port as the input port. Whether or not this is allowed is controlled by the RTR.RTRCFG.SA bit. If self addressing is not allowed, the incoming packet is spilled and an invalid address error occurs.

When group adaptive routing is used, and self addressing is not allowed, the input port is still allowed to be in the group of ports configured for the packet. The packet is not spilled until the router actually selects the input port as output port. If the router selects one of the other ports in the group, the packet is not spilled.

When packet distribution is used, and self addressing is not allowed, the input port is not allowed to be in the group of ports configured for the packet, since the packet should be sent to all ports in the group.

82.2.11 Invalid address error

An invalid address error occurs under the conditions listed below.

- When an incoming packet's address corresponds to a non-existing port (physical addresses 20-31).
- When an incoming packet's address is a logical address that is not enabled (RTR.RTACTRL.EN = 0).

- When an incoming packet's address is a logical address for which the corresponding RTR.RTPMAP register is not initialized, or the corresponding RTR.RTPMAP.PE field set to all zeroes.
- When only one output port is selected for an incoming packet, and that port is disabled (RTR.PCTRL.DI = 1).
- When self addressing occurs, and the router is configured not to allow self addressing (RTR.RTRCFG.SA = 0).
- When a packet is routed with the static routing feature, and the physical address programmed in RTR.PCTRL2.SD is 0 (static routing to port 0 is not allowed).

For all the invalid address cases above, the incoming packet is spilled, and the RTR.PSTS.IA bit corresponding to the input port will be set to 1.

82.2.12 Link-start-on-request

The link-start-on-request feature gives the possibility to automatically start a SpaceWire port's link interface when a packet is routed to the port (i.e, port is selected as output port). Each port can have the feature individually enabled by setting the corresponding RTR.PCTRL.LR bit to 1.

If a packet arrives, and the link interface of the selected output port is not in run-state, and the port has the link-start-on-request feature enabled, the router will try to start the link interface under the following conditions:

1. The link interface is not already trying to start (RTR.PCTRL.LS = 0).

2. The link is not disabled (RTR.PCTRL.LD = 0).

3. The spill-if-not-ready feature is not enabled for the packet being routed.

The link will continue to be started until either the RTR.PCTRL.LD bit is set to 1, or until the link is disabled through the auto-disconnect feature, described in section 82.2.13.

The link-start-on-request feature is only available for the SpaceWire ports, since the configuration port, FIFO ports, and AMBA ports does not have a link interface FSM, and are therefore always considered to be in run-state.

82.2.13 Auto-disconnect

The auto-disconnect feature gives the possibility to automatically disable the link interface of a SpaceWire port if the port has been inactive for a long enough period of time. Each port can have the feature individually enabled by setting their corresponding RTR.PCTRL.AD bit to 1. The amount of time the port needs to be inactive for is decided by the settings of the global prescaler register (RTR.PRESCALER), and the port's individual timer register (RTR.PTIMER). This time period is the same as the timeout period used by the port's data character timer when recovering from deadlock situations (see section 82.2.14). If the auto-disconnect feature is enabled, then a SpaceWire port will automatically disable its link interface under the following conditions:

1. The link interface entered run-state because it was started by the link-start-on-request feature, described in section 82.2.12.

2. The packet that caused the link interface to start has finished (either sent or spilled).

3. Nothing has been transmitted or received on the port for the duration of the time period specified by the RTR.PRESCALER register, and the corresponding RTR.PTIMER register.

4. The port's corresponding RTR.PCTRL.LS bit has not been set to 1.

The auto-disconnect feature is only available for the SpaceWire ports, since the configuration port and FIFO ports, and AMBA ports does not have a link interface FSM, and are therefore always considered to be in run-state.

82.2.14 Port data character timers

Timer functionality can be optionally implemented by setting the timerbits VHDL generic to a non-zero value.

Each port has an individual data character timer, which can be used to timeout an ongoing data transfer in order to recover from a deadlock situation. There are two different timeouts defined: overrun timeout, and underrun timeout. An overrun timeout is when the input port has data available, but the output port(s) can not accept data fast enough. An underrun timeout is when the output port(s) can accept more data, but the input port can not provide data fast enough.

The timeout period for a port is set in its corresponding RTR.PTIMER register, and the timer is enabled through the corresponding RTR.PCTRL.TR bit. Timeouts due to overrun and underrun can also be individually enabled / disabled through the corresponding RTR.PCTRL2.OR and RTR.PCTRL2.UR bits.

It is always the input port's data character timer that is used for timing data transfers. When the timer is enabled, it counts down on every tick from the global prescaler (RTR.PRESCALER register). If a data character is transmitted from the input port to the output port(s), then the timer is restarted. If the timer expires, the ongoing packet is spilled, and an EEP is written to the transmit FIFO of the output port(s).

The range of the timeout period depends on the system clock frequency, and is calculated with the following formula:

<timeout period> = (<clock period> x (RTR.PRESCALER+1)) x RTR.PTIMER

Sub-sections 82.2.14.1 through 82.2.14.4 clarifies the behaviour of the timers for different scenarios that can occur when a packet arrives.

82.2.14.1 Timer disabled

If the data character timer for an input port is disabled, the incoming packet will wait indefinitely for the output port(s) to be ready, unless the spill-if-not-ready feature is enabled for the packet's address (see section 82.2.9).

82.2.14.2 Timer enabled, but output port(s) not in run state

If the spill-if-not-ready feature (see section 82.2.9) is disabled for the incoming packet's address, the input port's data character timer is started when the packet arrives, and if the output port's link interface has not entered run-state when the timer expires, the packet is spilled. When group adaptive routing is used for the incoming packet, it is enough for one of the possible output ports to enter run-state before the timer expires for the packet to be transmitted. If packet distribution is used, all the output ports must enter run-state before the timer expires, otherwise the packet is spilled.

If the link-start-on-request feature is enabled for an output port, that port will try to enter run-state when the packet arrives. However, the input port's timer is unaffected of this, and will still only wait for its configured timeout period, before spilling the packet.

A timeout due to the output port not being in run-state is classified as a overrun timeout, which means that the RTR.PCFG2.OR bit for the input port must be set in order for the packet to be spilled. If the RTR.PCFG2.OR bit is not set, the packet will wait indefinitely (unless spill-if-not-ready is enabled).

82.2.14.3 Timer enabled, output port(s) in run-state but busy with other transmission

The input port's data character timer will not start, and the incoming packet will wait indefinitely until the output port either becomes free or leaves run-state. When group adaptive routing is used for the incoming packet, it is enough for one of the possible output ports to be in run-state to prevent the timer from starting. If packet distribution is used, all the output ports must be in run-state to prevent the timer from starting.

82.2.14.4 Timer functionality when accessing the configuration port

The timer functionality is basically the same for the configuration port as for the other ports. When the command is being received, the configuration port is the output port of the data transfer, and when the reply is being sent, the configuration port is the input port of the data transfer. The differences between the configuration port and the other ports are:

- The configuration port can always accept data fast enough, which means that an overrun timeout will never occur when a command is being received.
- The configuration port can always send data fast enough, which means that an underrun timeout will never occur when a reply is being sent.

82.2.15 Packet length truncation

Packet length truncation monitors the length of an incoming packet, and increases a counter for each received data character. If the counter reaches a value larger than the input port's RTR.MAXPLEN register, and truncation is enabled for the input port (RTR.PCTRL.PL = 1), the rest of the packet is spilled, and an EEP is written to the FIFO of the output port(s). Each port has its own RTR.MAX-PLEN register and counter in order to allow different maximum lengths for different ports.

Packet length truncation can also be enabled for port 0. In that case, it is the length of the RMAP / SpaceWire Plug-and-Play reply packet that is monitored.

82.2.16 System time-distribution

The router supports system time distribution through time-codes, as defined in ECSS-E-ST-50-12C [SPW]. It contains a global time-counter register (RTR.TC) where the latest received time-code can be read. All ports support time-code transmission and reception. All incoming time-codes update the RTR.TC register. If the incoming time-code has a time value which is plus one (modulo 64) compared to the old RTR.TC value, the time-code is forwarded to all the other ports. The time-code is not sent out onto the port on which it arrived.

Time-codes can be globally enabled / disabled through the RTR.TC register, as well as individually enabled / disabled per port through respective RTR.PCTRL.TE bit. When time-codes are disabled for a port, all incoming time-codes on that port are discarded, and no time-codes will be forwarded to the port.

The router can be configured to either filter out all incoming time-codes that does not have the two control flags (bit 7:6) set to "00", or to discard the control flags and allow them to have any value. This configuration is done through the RTR.RTRCFG.TF bit. The value of the control flags for the last received time-code can also be read from the RTR.TC register. Note that if interrupt distribution is globally enabled (RTR.RTRCTRL.IE = 1), only control flags "00" are considered as time-codes, no matter the value of the RTR.RTRCFG.TF bit.

82.2.17 SpaceWire distributed interrupt support

The router supports SpaceWire distributed interrupts. It can be configured to operate in two modes, interrupt with acknowledgment mode and extended interrupt mode. In the interrupt with acknowledg-ment mode, 32 interrupt numbers are supported, whereas the extended interrupt mode supports 64 interrupt numbers. The operation mode is configured through the RTR.RTRCFG.EE bit.

A distributed interrupt code is a control code that has the control flags (bits 7:6) always set to "10".

- *Interrupt with acknowledgment mode:* A distributed interrupt code that has bit 5 set to 0 is called an interrupt code, and bits 4:0 specify an interrupt number between 0 and 31. When operating in the interrupt with acknowledgment mode, a distributed interrupt code with bit 5 set to 1 is called an interrupt acknowledgment code, which is used to acknowledge the interrupt with the interrupt number specified by bits 4:0.

- *Extended interrupt mode:* When operating in the extended interrupt mode, a distributed interrupt code with bit 5 set to 1 is called an extended interrupt code, and bits 4:0 specify an interrupt number between 32 and 63. If bit 5 is set to 0, bits 4:0 specify an interrupt number between 0 and 31.

The interrupt codes and extended interrupt codes are generated by the source of the interrupt event, while the interrupt acknowledgment code is sent by the interrupt handler for the corresponding interrupt number.

The router has two 32-bit ISR register (RTR.ISR0 and RTR.ISR1) where each bit corresponds to one interrupt number. A bit in the ISR registers is set to 1 when an interrupt code or extended interrupt code with the corresponding interrupt number is received. A bit in the ISR registers is set to 0 when an interrupt acknowledgment code with the corresponding interrupt number is received. Therefore, the ISR registers reflect the status of all interrupt numbers. Each interrupt number has also its own timer which is used to clear the ISR register bit if an interrupt acknowledgment code is not received before the timer expires (for example if operating in the extended interrupt mode), as well as an optional timer which is used to control how fast a bit in the RTR.ISR register is allowed to toggle. See section 82.2.17.2 for more details on the ISR timers. Note that although it is possible to clear the bits in the ISR registers, these registers should normally only be used for diagnostics and FDIR.

82.2.17.1 Receiving and transmitting distributed interrupt codes

When a distributed interrupt code is received on a port, or the auxiliary time-code / distributed interrupt code interface, the following requirements must be fulfilled in order for the code to be distributed:

1. Interrupt distribution is globally enabled (RTR.RTRCTRL.IE = 1), and enabled for the port that received the code (corresponding RTR.PCTRL.IC = 1).

2. If the received code is an interrupt code, the RTR.PCTRL2.IR bit for the port must be set to 1. If the received code is an interrupt acknowledgment code or extended interrupt code, the RTR.PCTRL2.AR bit for the port must be set to 1.

3. If the code is an interrupt code or extended interrupt code, the interrupt number's corresponding bit in RTR.ISR0 or RTR.ISR1 must be 0. If the code is an interrupt acknowledgment code, the corresponding bit in RTR.ISR0 must be 1.

4. No previous distributed interrupt code with the same interrupt number is waiting to be distributed.

5. The ISR change timers (see section 82.2.17.2) are either globally disabled (RTR.RTRCFG.IC = 0) or the interrupt number's corresponding ISR change timer has expired.

If one of the requirements above is not fulfilled, then the received code is discarded. If all of the requirements above are fulfilled, then the received code is placed in a queue. The queue is then serviced in one of the four following ways, depending on the settings of the RTR.RTRCFG.IS and RTR.RTRCFG.IP bits:

1. All interrupt codes have priority over all interrupt acknowledgment codes / extended interrupt codes (RTR.RTRCFG.IP = 0), and the interrupt numbers are serviced through a round-robin scheme (RTR.RTRCFG.IS = 0). This is the default service scheme after reset / power-up.

2. All interrupt codes have priority over all interrupt acknowledgment codes / extended interrupt codes (RTR.RTRCFG.IP = 0), and the interrupt numbers are serviced with priority to lower interrupt numbers (RTR.RTRCFG.IS = 1).

3. All interrupt acknowledgment codes / extended interrupt codes have priority over all interrupt codes (RTR.RTRCFG.IP = 1), and the interrupt numbers are serviced through a round-robin scheme (RTR.RTRCFG.IS = 0).

4. All interrupt acknowledgment codes / extended interrupt codes have priority over all interrupt codes (RTR.RTRCFG.IP = 1), and the interrupt numbers are serviced with priority to lower interrupt numbers (RTR.RTRCFG.IS = 1).

When a distributed interrupt code has been selected from the queue, it is forwarded to all ports (except the port it was received on) that has interrupt distribution enabled (RTR.PCTRL.IC = 1), and that has enabled transmission of interrupt codes or interrupt acknowledgment codes / extended interrupt codes (RTR.PCTRL2.IT and RTR.PCTRL2.AT respectively).

82.2.17.2 Interrupt distribution timers

Each interrupt number has two corresponding timers, called the ISR timer, and ISR change timer:

The ISR timer is started and reloaded with the value from the RTR.ISRTIMER register each time a received interrupt code / extended interrupt code sets the corresponding RTR.ISR0 / RTR.ISR1 bit to 1. If an interrupt acknowledgment code is received, the corresponding ISR timer is stopped. If the ISR timer expires before an interrupt acknowledgment code is received, the corresponding bit in the RTR.ISR0 or RTR.ISR1 register is cleared. The use of ISR timers is always enabled. In the interrupt with acknowledgment mode, the purpose of the timers is to recover from situations where an interrupt acknowledgment code is lost. In the extended interrupt mode, the purpose of the ISR timers is to limit the rate of which interrupt codes are forwarded. It is important to configure the reload value for the ISR timer correctly. In the interrupt with acknowledgment mode, the reload value must not be less than the worst propagation delay for the interrupt code, plus the maximum delay in the interrupt handler, plus the worst propagation delay for the interrupt acknowledgment code. In the extended interrupt mode, the reload value must not be less than the worst propagation delay for the interrupt code / extended interrupt code.

The ISR change timers are timers that optionally can be used to control the minimum delay between two consecutive changes to the same RTR.ISR0 / RTR.ISR1 bit. The purpose of the timers is to protect against unexpected code occurrences that could occur, for example, due to a network malfunction or a babbling idiot. If the use of ISR change timers is enabled (RTR.RTRCFG.IC = 1), then the ISR change timer for an RTR.ISR0 / RTR.ISR1 bit is started and reloaded with the value from the RTR.ISRCTIMER register each time a received distributed interrupt code makes the RTR.ISR0 / RTR.ISR1 bit change value. Until the timer has expired, the corresponding RTR.ISR0 / RTR.ISR1 bit is not allowed to change value, and any received distributed interrupt code with that interrupt number is discarded. In the extended interrupt mode, the ISR change timers are not used, and should be disabled.

82.2.17.3 Interrupt code generation

In addition to distributing interrupt codes received on the ports, the router can also generate an interrupt code / extended interrupt code when an internal error event occurs, such that the IRQ pin is set to 1. See section 82.2.21 for information about how to control which errors that set the IRQ pin. In addition to the errors described in 82.2.21 the IRQ pin can also be configured to be set when a SpaceWire port's link interface enters run-state.

Everything in sections 82.2.17.1 and 82.2.17.2 also applies when the distributed interrupt code is generated by the router. The only difference is that a distributed interrupt code generated by the router will not be discarded if it is not allowed to be distributed. Instead, the distributed interrupt code will be distributed later, as soon as it is allowed. The only time a distributed interrupt code generated by the router is not distributed is if the bits in RTR.PIP are cleared by software before the interrupt code is allowed to be sent.

The interrupt code generation is controlled through the RTR.ICODEGEN register, and in addition to the enable / disable bit (RTR.ICODEGEN.EN), the following features are available:

- The interrupt number to use for a generated distributed interrupt code is programmable through the RTR.ICODEGEN.IN field.
- The generated distributed interrupt code can be configured to be either level type, or edge type. Level type means that a new distributed interrupt code will be sent as long as the IRQ pin is set (i.e. as long as any bit in the RTR.PIP register is set). Edge type means that a new distributed

interrupt code will only be sent when a new error event occurs (an RTR.PIP bit toggles from 0 to 1). The type is selected by the RTR.ICODEGEN.IT bit.

- A timer can be enabled through the RTR.ICODEGEN.TE bit. This timer controls the minimum time between a received interrupt acknowledgment code and the distribution of a new generated interrupt code. The timer is started and reloaded with the value from the RTR.AITIMER register when an interrupt acknowledgment code is received, if the router was the source of the corresponding interrupt code. Until the timer has expired, a new generated interrupt code will not be distributed. The reload value should not be less than the worst propagation delay for the interrupt acknowledgment code. This timer is unused when operating in the extended interrupt mode.
- Through the RTR.ICODEGEN.AH and RTR.ICODEGEN.UA bits the router can be configured to, upon receiving an interrupt acknowledgment code, or the when the ISR timer expires, automatically clear the RTR.PIP bits that were set when the distributed interrupt code was generated.

82.2.18 Auxiliary time-code / distributed interrupt code interface

There is an auxiliary time-code / distributed interrupt code interface that consists of the following signals AUXTICKIN, AUXTICKOUT, AUXTIMEOUT[7:0], AUXTIMEIN[7:0], and AUXTIMEINEN.

The AUXTICKOUT is set high for one CLK cycle when a valid time-code or distributed interrupt code is received on any of the ports, and the value of the time-code / distributed interrupt code is presented on the AUXTIMEOUT[7:0] signals. The rules that determine whether or not a received time-code / distributed interrupt code will be sent out on the auxiliary interface are the same as for all the other ports.

The AUXTICKIN allows for transmission of time-codes and distributed interrupt codes from the auxiliary interface to the other ports. The AUXTICKIN signal can be either synchronous or asynchronous to CLK. In the synchronous mode, a tick is detected each clock cycle that AUXTICKIN is high. In the asynchronous mode, a tick is detected each time a rising edge is seen on AUXTICKIN.

If AUXTIMEINEN is not asserted when a tick i detected, the router's internal time-count value (RTR.TC.TC) is used to send a new time-code out onto the ports. If AUXTIMEINEN is asserted when a tick is detected, the router checks the value of the AUXTIMEIN[7:0] signals to determine if a time-code or distributed interrupt code should be sent. Note that, in the asynchronous mode, AUXTIMEIN[7:0] must hold their value for at least two CLK cycles after the rising edge on AUXTICKIN.

The rules that determine whether a distributed interrupt code should be forwarded to the ports are the same as when a distributed interrupt code is received from any other port. However, for time-codes, the value on AUXTIMEIN[7:0] is always forwarded, and the router's internal time-count value is updated. Note that time-codes / distributed interrupt codes received through the auxiliary interface is not forwarded back out onto the interface itself.

Just as for the router ports, the auxiliary interface has enable / disable bits for time-codes (RTR.RTRCFG.AT) and distributed interrupt codes (RTR.RTRCFG.AI), which need to be set high in order for respective code to be transmitted / received.

82.2.19 SpaceWire-D support

82.2.19.1 Time-code / distributed interrupt code truncation

The router supports truncation of packets when it receives a valid time-code / distributed interrupt code (time-code or distributed interrupt code). A time-code is considered valid when the value equals the internal time count plus one (modulo 64). An distributed interrupt code is considered valid if the corresponding ISR bit is flipped due to reception of the code. The feature can be enabled individually for each port by setting the corresponding RTR.PCTRL.TS bit to 1. A filter, allowing only certain time-codes / distributed interrupt codes to spill packets, can also be configured individually for each port (see RTR.PCTRL2 register).

If a packet transfer is ongoing when a valid time-code / distributed interrupt code is received, and the code matches the filter in RTR.PCTRL2, the rest of the packet is spilled, and an EEP is written to the FIFO of the output port(s).

Time-code / distributed interrupt code truncation can also be enabled for port 0. In that case, it is the RMAP / SpaceWire Plug-and-Play reply packet that is spilled.

82.2.20 Character and packet counters

Each port, except port 0, has counters for incoming and outgoing characters and packets. For the character counters (RTR.ICHARCNT / RTR.OCHARCNT registers), only SpaceWire data characters are counted (not EOP/EEP). Characters deleted due to header deletion are counted on the incoming port but not on the outgoing port. For the packet counters (RTR.IPKTCNT / RTR.OPKTCNT registers), each EOP/EEP that is preceded by at least one data character is counted as a packet. The counters wrap around, and signal an overflow, when they reach the maximum value. The counters are accessed through the configuration port. See section 82.7 for details.

82.2.21 Error detection and reporting

The router can detect and report the following errors:

- SpaceWire link errors: parity error, disconnect error, escape error, credit error (see ECSS-E-ST-50-12C [SPW] for definition of these errors).
- Packet spill due to: timeout (see section 82.2.14), packet length truncation (see section 82.2.15), time-code / distributed interrupt code truncation (see section 82.2.19.1), and spill-if-not-ready feature (see section 82.2.9)
- Invalid address error (see section 82.2.11)
- RMAP errors (see section 82.6.1)
- Memory errors in the memory used for the routing table, RMAP command buffers, and port transmit and receive FIFO (see section 82.2.25)
- SpaceWire Plug-and-Play errors (see section 82.7.1)

Each error type has corresponding status bits in respective RTR.PSTS register (RTR.PSTSCFG for the configuration port). Common for all the status bits is that they are set when the error is detected, and stay set until they are cleared manually.

Another way the router can report an error is to asserted the IRQ pin. Whether or not the IRQ pin is set when one of the above mentioned errors occur is controlled by the two mask registers, RTR.IMASK and RTR.IPMASK. When the error occur and both the port's corresponding bit in RTR.IPMASK as well as the error type's corresponding bit in RTR.IMASK, are set, then the port's corresponding bit in the Port interrupt pending register (RTR.PIP) is set. The IRQ pin is high as long as any bit in the RTR.PIP register is set.

The router can also be configured to generate a distributed interrupt code when the IRQ pin gets set. See section 82.2.17 for more details.

82.2.22 Setting link-rate for the SpaceWire ports

The initialization divisor register (RTR.IDIV) determines the link-rate during initialization (all states up to and including the connecting-state) for all SpaceWire ports. The register is also used to calculate the link interface FSM timeouts for all SpaceWire ports (6.4 us and 12.8 us, as defined in the SpaceWire standard). The RTR.IDIV register should always be set so that a 10 Mbit/s link-rate is achieved during initialization. In that case the timeout values will also be calculated correctly. The reset value of the RTR.IDIV.ID field is taken from the IDIVISOR[7:0] signal, see Table 1491 for details.

To achieve a 10 Mbit/s link-rate, the RTR.IDIV register should be set according to the following formula:

RTR.IDIV = (<frequency in MHz of internal SpaceWire clock> / 10) - 1

The link-rate in run-state can be controlled individually per SpaceWire port with the run-state divisor located in each port's control register (RTR.PCTRL.RD field). The link-rate in run-state is calculated according to the following formula:

<link-rate in Mbits/s> = <frequency in MHz of internal SpaceWire clock> / (RTR.PCTRL.RD+1)

The value in RTR.PCTRL.RD only affects the link-rate in run-state, and does not affect the 6.4 us or 12.8 us timeouts values. The reset value of the RTR.PCTRL.RD field is taken from the IDIVISOR[7:0] signal, see Table 1482 for details.

82.2.23 SpaceWire transceiver enable signals

There enable signals SPWEN, one for each SpaceWire port, intended for controlling external transceivers. Each index corresponds to the port with the same number. SPWEN can be used for enabling / disabling external LVDS transceivers.

The enable signal is active as long as either the corresponding LVDS driver or corresponding LVDS receiver is powered on.

82.2.24 Power saving features

The router supports two power saving features: signals for LVDS driver / receiver power down, and signals for clock gating of the ports.

When a SpaceWire port is inactive, the LVDS driver power-down signals is asserted. A SpaceWire port is considered inactive if one of the following conditions are true:

1. Link interface is disabled, either because RTR.PCTRL.LD is set to 1, or because of the automatic-disconnect feature (see section 82.2.13).
2. Link interface is not disabled, but the transmitter has not been enabled since the last time the link interface was disabled.
3. After reset, until the LVDS driver becomes enabled. The LVDS driver is enabled in the started-state, connecting-state, and run-state.

The signal for LVDS receiver is only asserted when the link interface is disabled.

For the clock gate signal for a SpaceWire port to be asserted, both the link disable bit (RTR.PCTRL.LD) and the port disable bit (RTR.PCTRL.DI) should be set. The port stays clock gated until either of the two bits are cleared.

82.2.25 On-chip memories

The router can have several memories on-chip. They are usually selected from technology specific RAM blocks but generic models can also be used which either make the synthesis tool infer the memories or use registers instead.

There are two memory blocks in the routing table, one for the port setup registers and one for the routing table. The port setup memory bit width is equal to the number of ports including the configuration port with depth 256. The routing table is 256 locations deep and 3 bits wide.

Each port excluding the configuration port also have FIFO memories. The SpaceWire ports have one FIFO per direction (rx, tx) which are 10-bit respective 9-bit wide. The depth is controlled with the fifosize VHDL generic. The FIFO ports have the exact same FIFO configuration as the SpaceWire ports controlled by the same generic.

The AMBA ports have one 9-bit wide receiver FIFO controlled by the fifosize generic and two 32-bit wide AHB FIFOs. The depth of the latter two is controlled with the fifosize2 generic (see the AMBA port section for more information).

All of the memories are instantiated using either the syncram_2p of syncram_2pft components from GRLIB (see the syncram_2p section in the GRIP manual for more information) depending on the value of the ft generic. If ft is set to 0 the syncram_2p is used and there is no fault-tolerance support. If ft is nonzero the syncram_2pft component is chosen and there are two variants of fault-tolerance depending on the value. Setting ft=1 selects parity and ft=2 selects TMR.

Parity is used to protect the memories and up to four bits per word can be corrected and there is a signal indicating an uncorrectable error. TMR is used to protect the memories and a voter determines the correct result. No errors are detectable for TMR and thus the error signal is not used.

If a memory error occurs in the port setup table or the routing table the memory error (ME) bit in the router configuration/status register is set and remains set until cleared by the user. If a memory error is detected in any of the ports FIFO memories the memory error (ME) bit in the respective port status register is set and remains set until cleared by the user. The ME bits are only set for uncorrectable errors.

When an uncorrectable error is detected in the port setup or routing table when a packet is being routed it will be discarded. Unocrrectable errors in the FIFO memories are not handled since they only affect the contents of the routed packet not the operation of the router itself. These type of errors should be caught by CRC checks if used in the packet.

The ME bit for the ports is only usable for detecting errors and statistics since there is no need to correct the error manually since the packet has already been routed when it is detected. The ME indication for the routing table and port setup registers can be used for starting a scrubbing operation if detected. There is also an option of having automatic scrubbing (see section 82.2.25.1)

If one or more of the memory error bits described above are set the signal ro.merror will be asserted.

82.2.25.1 Autoscrub

An autoscrubbing feature can be optionally enabled using the autoscrub VHDL generic.

With autoscrubbing enabled, the routing table and port setup registers are periodically read and rewritten. This is done to prevent a buildup of SEUs, which can cause an uncorrectable error in the memories. It runs in the background and has no timing impact on SpaceWire traffic but can delay configuration accesses by two cycles.

The scrubber starts at address 0 and simultaneously writes one location in the port setup memory and the routing table memory. It then waits for a timeout period before writing the next word. Eventually, the last location is reached and the process starts over from address 0.

The period between each word refresh is approximately 2^{26} system clock cycles (system clock used for AMBA system). The scrubber uses a free time slot, in which data traffic does not perform a table lookup to read and write the memories that causes a small nondeterminism in the period.

82.3 SpaceWire ports

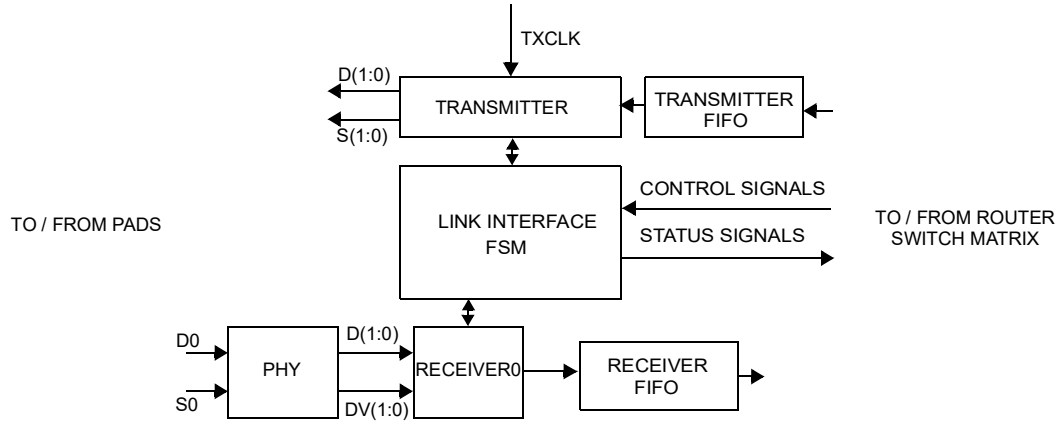


Figure 215. Block diagram. Represents a configuration with *dualport*=0 (see sections 82.3.1 and 82.10).

Each SpaceWire port comprises a SpaceWire codec that implements an encoder-decoder compliant to ECSS-E-ST-50-12C [SPW]. This is the same SpaceWire codec core (GRSPW_CODEC) provided in GRLIB as a separate core. It is a very versatile core providing several external interface types. The receiver can be configured as self-clocking (XOR gate based), SDR-, DDR sampling or with an interface to an Frontgrade SpW transceiver (UT200SpWPHY01). The transmitter can also be configured with SDR, DDR or SpWPHY outputs. Both internal and external LVDS transceivers are supported. The interface to the router's switch matrix consists of a transmit FIFO, receive FIFO, and control and status signals, see Figure 215. The size of both transmit and receive FIFOs is configurable in terms of number of N-chars that they can store at any time; the default is 64 N-chars. All the configuration parameters and status information for the ports are accessible through the router's configuration port, either through RMAP or AMBA AHB (see section 82.7). For more detailed information about the codec itself see the SpaceWire codec section in the GRLIB IP library manual.

82.3.1 Redundant ports

Redundant ports can be optionally enabled for SpaceWire ports. This is configured using the *dualport* VHDL generic. The redundant port is part of the codec implementation and provides an extra set of SpaceWire transmit and receive signals. This means each SpaceWire router port will be connected to two external SpaceWire ports. Only one of the two external ports can have an active SpaceWire link at any given time and data transferred on that router port will be received and transmitted on the active link. The external port to use for the active link can be forced via the RTR.PCTRL register, or be determined automatically based on the current activity on primary and redundant port links.

82.3.2 Link-interface FSM

The link-interface FSM controls the link interface (a more detailed description is found in ECSS-E-ST-50-12C [SPW]). The low-level protocol handling (the signal and character level) is handled by the transmitter and receiver while the FSM handles the exchange level.

The link-interface FSM is controlled through the control signals provided in the RTR.PCTRL registers. The link can be disabled through the RTR.PCTRL.LD bit, which depending on the current state, either prevents the link-interface from reaching the started-state, or forces it to the error-reset state. When the link is not disabled, the link interface FSM is allowed to enter the started-state when either the RTR.PCTRL.LS bit is set, or the link-start-on-request feature described in section 82.2.12 is trying to start the port, or when a NULL character has been received and the RTR.PCTRL.AS bit is set.

The current state of the link-interface determines which type of characters are allowed to be transmitted, which, together with the requests made from the host interface, determine what character will be sent.

When the link-interface is in the connecting- or run-state it is allowed to send FCTs. FCTs are sent automatically by the link-interface when possible. This is done based on the maximum value of 56 for the outstanding credit counter and the currently free space in the receive FIFO. FCTs are sent as long as the outstanding counter is less than or equal to 48, and there are at least 8 more empty FIFO entries than the counter value.

N-Chars are sent in the run-state when they are available from the transmit FIFO, and there are credits available. NULLs are sent when no other character transmission is requested, or when the FSM is in a state where no other transmission is allowed.

The credit counter (incoming credits) is automatically increased when FCTs are received, and decreased when N-Chars are transmitted. The credit counter for a SpaceWire port can be read in the corresponding RTR.CREDCNT register.

82.3.3 Transmitter

The state of the FSM, credit counters, possible request to send a time-code / distributed interrupt code, and requests from the transmit FIFO are used to decide the next character to be transmitted. The type of character and the character itself (for N-Chars and time-codes / distributed interrupt codes) to be transmitted are presented to the low-level transmitter, which run on the internal SpaceWire clock. For information on how to change the transmission rate, please see section 82.2.22.

The state of the FSM, credit counters, requests from the time-interface and requests from the transmitter FIFO are used to decide the next character to be transmitted. The type of character and the character itself (for N-Chars and Time-codes) to be transmitted are presented to the low-level transmitter which is located in a separate clock-domain.

82.3.4 Receiver

The receiver detects connections from other nodes and receives characters as a bit stream recovered from the data and strobe signals by the PHY module, which presents it as a data and data-valid signal.

The receiver is activated as soon as the link-interface leaves the error reset state. Then after a NULL is received it can start receiving any characters. It detects parity, escape and credit errors, which causes the link interface to enter the error-reset state.

Received L-Chars are the handled automatically by link-interface, while all N-Chars are stored in the receive FIFO.

82.3.5 Setting link-rate

The router's Initialization divisor register (RTR.IDIV) determines the link-rate during initialization (all states up to and including the connecting-state). The register is also used to calculate the link interface FSM timeouts (6.4 µs and 12.8 µs, as defined in the SpaceWire standard). The register's ID field should always be set so that a 10 Mbit/s link-rate is achieved during initialization. In that case the timeout values will also be calculated correctly.

To achieve a 10 Mbit/s link-rate, the ID field should be set according to the following formulas:

With single data rate (SDR) outputs:

ID = (<frequency in MHz of TXCLK> / 10) - 1

With double data rate (DDR) outputs, or when connected to Frontgrade SpaceWire transceiver:

ID = (2 x <frequency in MHz of TXCLK> / 10) - 1

The link-rate in run-state is controlled with the run-state divisor, which is set through the RD field of each port's Port control register. The link-rate in run-state is calculated according to the following formulas:

With SDR outputs:

<link-rate in Mbits/s> = <frequency in MHz of TXCLK> / (RD+1)

With DDR outputs / Frontgrade SpaceWire transceiver:

<link-rate in Mbits/s> = 2 x <frequency in MHz of TXCLK> / (RD+1)

The value of the RD field only affects the link-rate in run-state, and does not affect the 6.4 µs or 12.8 µs timeouts values.

Note that when using DDR ouputs, or when connected to Frontgrade SpaceWire transceiver, there is a limitation in the usable clock divisor values. All even values (except 0) will result in the same bitrate as the one higher odd number.

An example of clock divisor and resulting link-rate, with a TXCLK frequency of 50 MHz, is shown in the table 1444.

Table 1444. SpaceWire link-rate example with 50 MHz TXCLK

	Link-rate in Mbit/s	
Clock divisor value	**SDR outputs**	**DDR outputs / Frontgrade SpaceWire transceiver**
0	50	100
1	25	50
2	16.67	25
3	12.5	25
4	10	16.67
5	8.33	16.67
6	7.14	12.5
7	6.25	12.5
8	5.56	10
9	5	10

82.4 FIFO ports

A port configured as a FIFO port contains one FIFO in each direction to/from the switch matrix. The fifosize can be configured to 8, 16, 32 and 64 using a VHDL generic (note that the same generic is used for the fifosizes in the SpaceWire links and the AHB interfaces).

82.4.1 Transmitter

The transmitter FIFO interface consists of the following signals: txfull, txafull, txwrite, txchar, txcharcnt. Figure 216 illustrates the write operation. Note that txfull would only be asserted as illustrated in the figure when txcharcnt is 4 if the FIFO size is 4 (which is not the case typically).

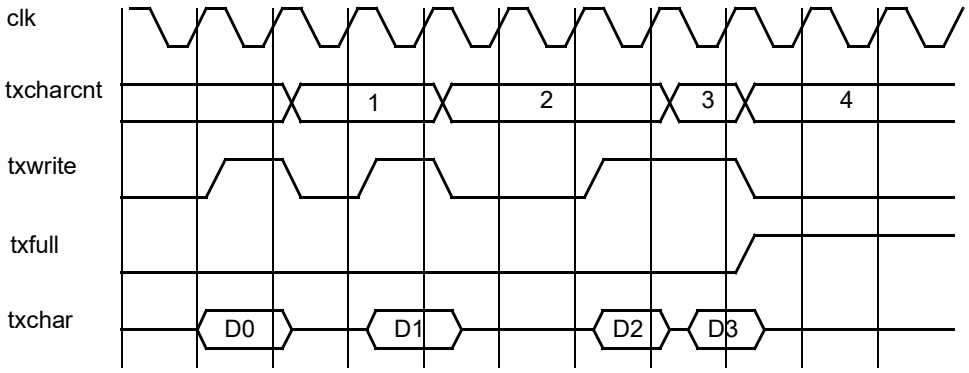


Figure 216. Transmitter FIFO interface write cycle.

Txwrite is the write signal and each time when asserted on the rising edge of the clock the value on the txchar signal will be written into the transmitter FIFO if it is not full. If it is full the character will be dropped. Txcharcnt indicates the number of characters currently in the FIFO. Txfull is asserted when the FIFO is full and txafull is asserted when the FIFO is almost full. The almost full signal is configurable through the almostsize VHDL generic and will be asserted when the FIFO contains a number of characters more than or equal to fifosize-almostsize.

The transmitter FIFO can be reset through the port's control register using the TF bit.

82.4.2 Receiver

The receiver FIFO interface consists of the following signals: rxread, rxchar, rxcharav and rxaempty. Figure 217 illustrates the read operation. Note that rxcharav would only be deasserted as illustrated in the figure if the FIFO contained 4 characters.

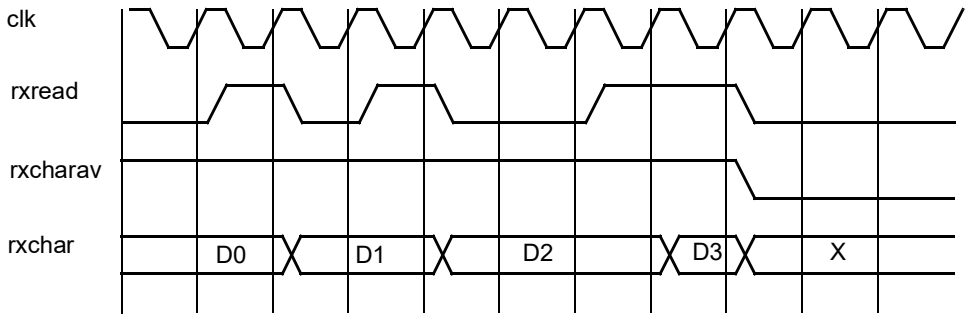


Figure 217. Receiver FIFO interface read cycle.

Each time rxread is asserted on the rising edge of the clock a new character will be available on the rxchar output the next cycle if available. If the FIFO is empty the value is undefined. Rxcharav is asserted when the FIFO contains at least one character. Rxaempty is asserted when the FIFO is almost empty and is defined as when the FIFO contains a number of characters less than or equal to the almostsize VHDL generic.

The receiver FIFO can be reset using the RF bit in the port's control register.

82.4.3 Time-code transmit

The time-code transmit interface consists of the following signals: tickin, timein. Figure 218 illustrates the tickin operation.

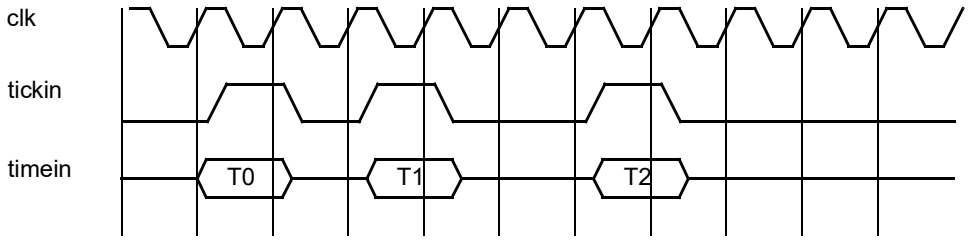


Figure 218. Time interface tickin operation.

82.4.4 Time-code receive

The time-code receive interface consists of the following signals: tickout, timeout. Figure 219 illustrates the tickout operation.

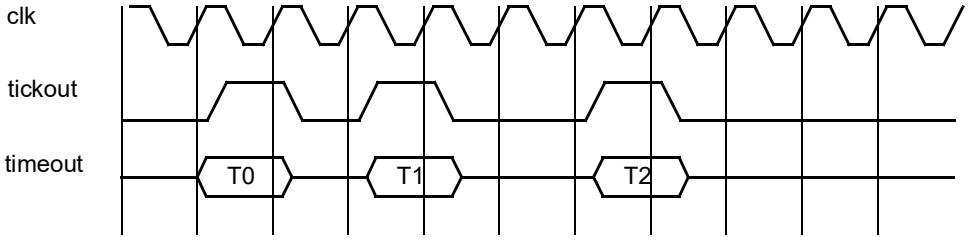


Figure 219. Time interface tickout operation.

The clock that all the interface signals are synchronized to is the same as the core clock (the clock that everything except the SpaceWire links' transmitters and receivers are running on). It can run on any frequency but to support the maximum throughput it has to be at least one eighth of the maximum link bitrate.

82.4.5 Bridge mode

The FIFO ports normally operate in standard mode which has been described so far in this section. But they can also be set in bridge mode through the bridge enable (BE) bit in the port's control register. The reset value of this signal is set through an input signal so this mode can be enabled per default after reset.

In bridge mode two FIFO ports can be connected together with automatic packet and time-code transfer without any glue logic. Table 1445 shows how the signals should be mapped. Rxaempty and txafull are unused in this mode.

Table 1445. Signal mappings of FIFO port in bridge mode.

Port 0	Port 1
rxchar	txchar
rxread	txfull
txwrite	rxcharav
txchar	rxchar
txfull	rxread
rxcharav	txwrite
tickin	tickout
timein	timeout
tickout	tickin
timeout	timein

82.5 AMBA ports

The AMBA ports are Frontgrade Gaisler's GRSPW2 controller with the SpaceWire codec removed. Thus, the same drivers that are provided for the GRSPW2 can also be used for an AMBA port of the router. Only one additional driver is needed, which handles the setup of the registers within the configuration port.

82.5.1 Overview

The router's AMBA ports are configured by register accesses through an APB interface. Data is transferred through one to four DMA channels using an AHB master interface. The number of DMA channels is configurable from one to four. Only byte addressed 32-bit big-endian AHB buses are supported.

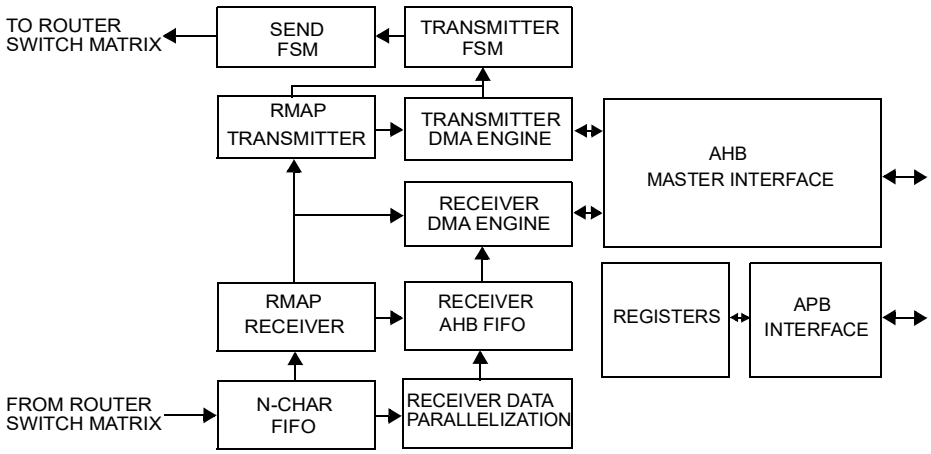


Figure 220. Block diagram of the Router DMA port

82.5.2 Operation

The main sub-blocks of the router AHB interfaces are the DMA engines, the RMAP target and the AMBA interface. A block diagram of the internal structure can be found in figure 220.

The AMBA interface is divided into the AHB master interface and the APB interface. The DMA engines have FIFO interfaces to the router switch matrix. These FIFOs are used to transfer N-Chars between the AMBA bus and the other ports in the router.

The RMAP target is an optional part of the DMA port which can be enabled with a VHDL generic. The RMAP target handles incoming packets which are determined to be RMAP commands instead of the receiver DMA engine. The RMAP command is decoded and if it is valid, the operation is performed on the AHB bus. If a reply was requested it is automatically transmitted back to the source by the RMAP transmitter.

The AMBA ports are controlled by writing to a set of user registers through the APB interface and a set of signals. The different sub-modules are discussed in further detail in later sections.

82.5.2.1 Protocol support

The AMBA port only accepts packets with a valid destination address in the first received byte. Packets with address mismatch will be silently discarded (except in promiscuous mode which is covered in section 82.5.4.10).

The second byte is sometimes interpreted as a protocol ID a described hereafter. The RMAP protocol (ID=0x1) is the only protocol handled separately in hardware while other packets are stored to a DMA channel. If the RMAP target is present and enabled all RMAP commands will be processed, executed and replied automatically in hardware. Otherwise RMAP commands are stored to a DMA channel in the same way as other packets. RMAP replies are always stored to a DMA channel. More information on the RMAP protocol support is found in section 82.5.6 (note that this RMAP target is different from the one in the configuration port). When the RMAP target is not present or disabled, there is no need to include a protocol ID in the packets and the data can start immediately after the address.

All packets arriving with the extended protocol ID (0x00) are stored to a DMA channel. This means that the hardware RMAP target will not work if the incoming RMAP packets use the extended protocol ID. Note also that packets with the reserved extended protocol identifier (ID = 0x000000) are not ignored by the AMBA port. It is up to the client receiving the packets to ignore them.

When transmitting packets, the address and protocol-ID fields must be included in the buffers from where data is fetched. They are *not* automatically added by the AMBA port DMA engine.

Figure 221 shows the packet types accepted by the port. The port also allows reception and transmission with extended protocol identifiers but without support for RMAP CRC calculations and the RMAP target.

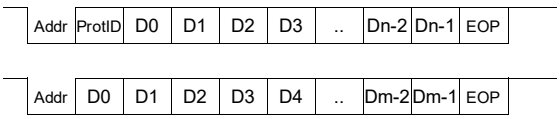


Figure 221. The SpaceWire packet types supported by the port.

82.5.3 Time-Code / distributed interrupt interface

82.5.3.1 Sending and receiving Time-Codes

To transmit a Time-Code through the register interface of an AMBA port, the RTR.AMBACTRL.TI bit should be written to 1. When the bit is written, the AMBA port's current time value (RTR.AMBATC.TIMECNT field) is incremented and a Time-Code consisting of the new time value together with the current control flags (RTR.AMBATC.TCTRL field) is sent. The RTR.AMBACTRL.TI bit will stay high until the Time-Code has been transmitted. Note that whether or not a Time-Code is forwarded to any other port after it has been sent by an AMBA port depends on the settings explained in 82.2.16.

A Time-Code that is received by an AMBA port will be stored in the port's RTR.AMBATC register. There is also a possibility to generate AMBA interrupts and tick-out pulses. This is controlled through the RTR.AMBACTRL and RTR.AMBATC registers. See section 82.5.9 for details about these registers.

82.5.3.2 Sending and receiving distributed interrupts

To transmit a distributed interrupt code through the register interface of an AMBA port, the RTR.AMBAINTCTRL.II bit in should be written to 1. When the bit is written, the value of the RTR.AMBAINTCTRL.TXINT field determine which distributed interrupt code that will be sent. Note that whether or not a distributed interrupt code is forwarded to any other port after it has been sent by an AMBA port depends on the settings of the router and the state of that specific interrupt in the network. See 82.2.17 for details.

A distributed interrupt can also be configured to be generated automatically by the AMBA port upon certain events. This is controlled through the RTR.AMBAINTCTRL and RTR.AMBADMACTRL registers. See section 82.5.9 for details about these registers and features.

Distributed interrupt codes that are received by an AMBA port can be programmed to generate AMBA interrupts as well as tick out-pulses. See section 82.5.9 for details about these features.

82.5.3.3 Sending and receiving Time-Codes and distributed interrupts via signal interface

For all received control codes, Time-Codes or not, the control flags together with the time value are outputted on the TIMEOUT[7:0] signals, and the TICKOUT signal is asserted for one system clock cycle.

Control codes can be sent by asserting the TICKIN signal and place the value of the Time-Code / distributed interrupt to be sent on the TIMEIN[7:0] signals.

82.5.4 Receiver DMA channels

The receiver DMA engine handles reception of data from the SpaceWire network to different DMA channels.

82.5.4.1 Address comparison and channel selection

Packets are received to different channels based on the address and whether a channel is enabled or not. When the receiver N-Char FIFO contains one or more characters, N-Chars are read by the receiver DMA engine. The first character is interpreted as the logical address and is compared with the addresses of each channel starting from 0. The packet will be stored to the first channel with an matching address. The complete packet including address and protocol ID but excluding EOP/EEP is stored to the memory address pointed to by the descriptors (explained later in this section) of the channel.

Each SpaceWire address register has a corresponding mask register. Only bits at an index containing a zero in the corresponding mask register are compared. This way a DMA channel can accept a range of addresses. There is a default address register which is used for address checking in all implemented DMA channels that do not have separate addressing enabled and for RMAP commands in the RMAP target. With separate addressing enabled the DMA channels' own address/mask register pair is used instead.

If an RMAP command is received it is only handled by the target if the default address register (including mask) matches the received address. Otherwise the packet will be stored to a DMA channel if one or more of them has a matching address. If the address does not match neither the default address nor one of the DMA channels' separate register, the packet is still handled by the RMAP target if enabled since it has to return the invalid address error code. The packet is only discarded (up to and including the next EOP/EEP) if an address match cannot be found and the RMAP target is disabled.

Packets, other than RMAP commands, that do not match neither the default address register nor the DMA channels' address register will be discarded. Figure 222 shows a flowchart of packet reception.

At least 2 non EOP/EEP N-Chars needs to be received for a packet to be stored to the DMA channel unless the promiscuous mode is enabled in which case 1 N-Char is enough. If it is an RMAP packet with hardware RMAP enabled 3 N-Chars are needed since the command byte determines where the packet is processed. Packets smaller than these sizes are discarded.

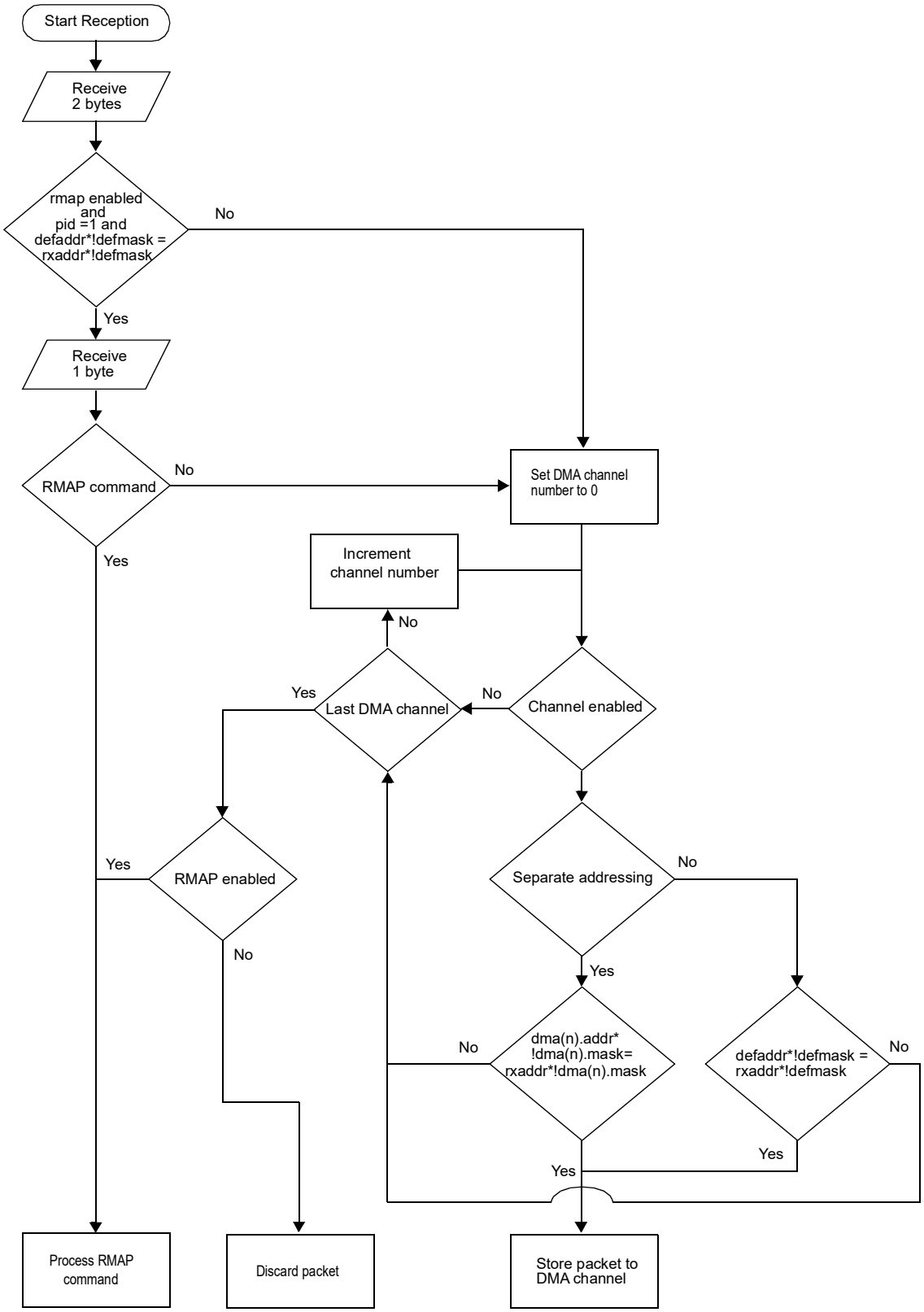


Figure 222. Flow chart of packet reception (promiscuous mode disabled).

82.5.4.2 Basic functionality of a channel

Reception is based on descriptors located in a consecutive area in memory that hold pointers to buffers where packets should be stored. When a packet arrives at the port the channel which should receive it is first determined as described in the previous section. A descriptor is then read from the channels' descriptor area and the packet is stored to the memory area pointed to by the descriptor. Lastly, status is stored to the same descriptor and increments the descriptor pointer to the next one. The following sections will describe DMA channel reception in more detail.

82.5.4.3 Setting up the port for reception

A few registers need to be initialized before reception to a channel can take place. The DMA channel has a maximum length register which sets the maximum packet size in bytes that can be received to this channel. Larger packets are truncated and the excessive part is spilled. If this happens an indication will be given in the status field of the descriptor. The minimum value for the receiver maximum length field is 4 and the value can only be incremented in steps of four bytes up to the maximum value 33554428. If the maximum length is set to zero the receiver will *not* function correctly.

Either the default address register or the channel specific address register (the accompanying mask register must also be set) needs to be set to hold the address used by the channel. A control bit in the DMA channel control register determines whether the channel should use default address and mask registers for address comparison or the channel's own registers. Using the default register the same address range is accepted as for other channels with default addressing and the RMAP target while the separate address provides the channel its own range. If all channels use the default registers they will accept the same address range and the enabled channel with the lowest number will receive the packet.

Finally, the descriptor table and control register must be initialized. This will be described in the two following sections.

82.5.4.4 Setting up the descriptor table address

The port reads descriptors from an area in memory pointed to by the receiver descriptor table address register. The register consists of a base address and a descriptor selector. The base address points to the beginning of the area and must start on a 1024 bytes aligned address. It is also limited to be 1024 bytes in size which means the maximum number of descriptors is 128 since the descriptor size is 8 bytes.

The descriptor selector points to individual descriptors and is increased by 1 when a descriptor has been used. When the selector reaches the upper limit of the area it wraps to the beginning automatically. It can also be set to wrap at a specific descriptor before the upper limit by setting the wrap bit in the descriptor. The idea is that the selector should be initialized to 0 (start of the descriptor area) but it can also be written with another 8 bytes aligned value to start somewhere in the middle of the area. It will still wrap to the beginning of the area.

If one wants to use a new descriptor table the receiver enable bit has to be cleared first. When the rxactive bit for the channel is cleared it is safe to update the descriptor table register. When this is finished and descriptors are enabled the receiver enable bit can be set again.

82.5.4.5 Enabling descriptors

As mentioned earlier one or more descriptors must be enabled before reception can take place. Each descriptor is 8 byte in size and the layout can be found in the tables below. The descriptors should be written to the memory area pointed to by the receiver descriptor table address register. When new descriptors are added they must always be placed after the previous one written to the area. Otherwise they will not be noticed.

A descriptor is enabled by setting the address pointer to point at a location where data can be stored and then setting the enable bit. The WR bit can be set to cause the selector to be set to zero when

reception has finished to this descriptor. IE should be set if an interrupt is wanted when the reception has finished. The DMA control register interrupt enable bit must also be set for an interrupt to be generated. The descriptor packet address should be word aligned. All accesses on the bus are word accesses so complete words will always be overwritten regardless of whether all 32-bit contain received data. Also if the packet does not end on a word boundary the complete word containing the last data byte will be overwritten. If the rxunaligned or rmap VHDL generic is set to 1 this restriction is removed and any number of bytes can be received to any packet address without excessive bytes being overwritten.

Table 1446. RXDMA receive descriptor word 0 (address offset 0x0)

31	30	29	28	27	26	25	24	0
TR	DC	HC	EP	IE	WR	EN	PACKETLENGTH	

31	Truncated (TR) - Packet was truncated due to maximum length violation.
30	Data CRC (DC) - 1 if a CRC error was detected for the data and 0 otherwise.
29	Header CRC (HC) - 1 if a CRC error was detected for the header and 0 otherwise.
28	EEP termination (EP) - This packet ended with an Error End of Packet character.
27	Interrupt enable (IE) - If set, an interrupt will be generated when a packet has been received if the receive interrupt enable bit in the DMA channel control register is set.
26	Wrap (WR) - If set, the next descriptor used by the GRSPW will be the first one in the descriptor table (at the base address). Otherwise the descriptor pointer will be increased with 0x8 to use the descriptor at the next higher memory location. The descriptor table is limited to 1 kbytes in size and the pointer will be automatically wrap back to the base address when it reaches the 1 kbytes boundary.
25	Enable (EN) - Set to one to activate this descriptor. This means that the descriptor contains valid control values and the memory area pointed to by the packet address field can be used to store a packet.
24: 0	Packet length (PACKETLENGTH) - The number of bytes received to this buffer. Only valid after EN has been set to 0 by the GRSPW.

Table 1447. RXDMA receive descriptor word 1 (address offset 0x4)

31	0
PACKETADDRESS	

31: 0	Packet address (PACKETADDRESS) - The address pointing at the buffer which will be used to store the received packet. If the rxunaligned and rmap VHDL generics are both set to zero only bit 31 to 2 are used.

82.5.4.6 Setting up the DMA control register

The final step to receive packets is to set the control register in the following steps: The receiver must be enabled by setting the rxen bit in the DMA control register (see section 82.7). This can be done anytime and before this bit is set nothing will happen. The rxdescav bit in the DMA control register is then set to indicate that there are new active descriptors. This must always be done after the descriptors have been enabled or the port might not notice the new descriptors. More descriptors can be activated when reception has already started by enabling the descriptors and writing the rxdescav bit. When these bits are set reception will start immediately when data is arriving.

82.5.4.7 The effect to the control bits during reception

When the receiver is disabled all packets going to the DMA-channel are discarded if the packet's address does not fall into the range of another DMA channel. If the receiver is enabled and the address falls into the accepted address range, the next state is entered where the rxdescav bit is checked. This bit indicates whether there are active descriptors or not and should be set by the external application using the DMA channel each time descriptors are enabled as mentioned above. If the rxdescav bit is '0' and the nospill bit is '0' the packets will be discarded. If nospill is one the grspw waits until rxdes-

cav is set and the characters are kept in the N-Char fifo during this time. If the fifo becomes full further N-char transmissions are inhibited by stopping the transmission of FCTs.

When rxdescav is set the next descriptor is read and if enabled the packet is received to the buffer. If the read descriptor is not enabled, rxdescav is set to '0' and the packet is spilled depending on the value of nospill.

The receiver can be disabled at any time and will stop packets from being received to this channel. If a packet is currently received when the receiver is disabled the reception will still be finished. The rxdescav bit can also be cleared at any time. It will not affect any ongoing receptions but no more descriptors will be read until it is set again. Rxdescav is also cleared by the port when it reads a disabled descriptor.

82.5.4.8 Status bits

When the reception of a packet is finished the enable bit in the current descriptor is set to zero. When enable is zero, the status bits are also valid and the number of received bytes is indicated in the length field. The DMA control register contains a status bit which is set each time a packet has been received. The port can also be made to generate an interrupt for this event.

RMAP CRC logic is included in the implementation if the rmapcrc or rmap VHDL generic set to 1. The RMAP CRC calculation is always active for all received packets and all bytes except the EOP/EEP are included. The packet is always assumed to be a RMAP packet and the length of the header is determined by checking byte 3 which should be the command field. The calculated CRC value is then checked when the header has been received (according to the calculated number of bytes) and if it is non-zero the HC bit is set indicating a header CRC error.

The CRC value is not set to zero after the header has been received, instead the calculation continues in the same way until the complete packet has been received. Then if the CRC value is non-zero the DC bit is set indicating a data CRC error. This means that the port can indicate a data CRC error even if the data field was correct when the header CRC was incorrect. However, the data should not be used when the header is corrupt and therefore the DC bit is unimportant in this case. When the header is not corrupted the CRC value will always be zero when the calculation continues with the data field and the behaviour will be as if the CRC calculation was restarted

If the received packet is not of RMAP type the header CRC error indication bit cannot be used. It is still possible to use the DC bit if the complete packet is covered by a CRC calculated using the RMAP CRC definition. This is because the port does not restart the calculation after the header has been received but instead calculates a complete CRC over the packet. Thus any packet format with one CRC at the end of the packet calculated according to RMAP standard can be checked using the DC bit.

If the packet is neither of RMAP type nor of the type above with RMAP CRC at the end, then both the HC and DC bits should be ignored.

82.5.4.9 Error handling

If an AHB error occurs during reception the current packet is spilled up to and including the next EEP/EOP and then the currently active channel is disabled and the receiver enters the idle state. A bit in the channels control/status register is set to indicate this condition.

82.5.4.10 Promiscuous mode

The port supports a promiscuous mode where all received data (excluding EOPs/EEPs) is stored to the first enabled DMA channel regardless of the node address and possible early EOPs/EEPs. The AMBA port DMA RX maximum length register is still checked and packets exceeding this size are truncated.

RMAP commands are still handled by the RMAP hardware target when promiscuous mode is enabled, if the RMAP enable bit in the AMBA port DMA control/status register is set. If the RMAP enable bit is cleared, RMAP commands are stored to a DMA channel instead.

82.5.5 Transmitter DMA channels

The transmitter DMA engine handles transmission of data from the DMA channels to the SpaceWire network. Each receive channel has a corresponding transmit channel which means there can be up to 4 transmit channels. It is however only necessary to use a separate transmit channel for each receive channel if there are also separate entities controlling the transmissions. The use of a single channel with multiple controlling entities would cause them to corrupt each other's transmissions. A single channel is more efficient and should be used when possible.

Multiple transmit channels with pending transmissions are arbitrated in a round-robin fashion.

82.5.5.1 Basic functionality of a channel

A transmit DMA channel reads data from the AHB bus and stores them in the transmitter FIFO for transmission on the SpaceWire network. Transmission is based on the same type of descriptors as for the receiver and the descriptor table has the same alignment and size restrictions. When there are new descriptors enabled the port reads them and transfer the amount data indicated.

82.5.5.2 Setting up the core for transmission

Four steps need to be performed before transmissions can be done with the port. First the link interface must be enabled and started by writing the appropriate value to the ctrl register. Then the address to the descriptor table needs to be written to the transmitter descriptor table address register and one or more descriptors must also be enabled in the table. Finally, the txen bit in the DMA control register is written with a one which triggers the transmission. These steps will be covered in more detail in the next sections.

82.5.5.3 Enabling descriptors

The descriptor table address register works in the same way as the receiver's corresponding register which was covered in section 82.5.4. The maximum size is 1024 bytes as for the receiver but since the descriptor size is 16 bytes the number of descriptors is 64.

To transmit packets one or more descriptors have to be initialized in memory which is done in the following way: The number of bytes to be transmitted and a pointer to the data has to be set. There are two different length and address fields in the transmit descriptors because there are separate pointers for header and data. If a length field is zero the corresponding part of a packet is skipped and if both are zero no packet is sent. The maximum header length is 255 bytes and the maximum data length is 16 Mbyte - 1. When the pointer and length fields have been set the enable bit should be set to enable the descriptor. This must always be done last. The other control bits must also be set before enabling the descriptor.

The transmit descriptors are 16 bytes in size so the maximum number in a single table is 64. The different fields of the descriptor together with the memory offsets are shown in the tables below.

The HC bit should be set if RMAP CRC should be calculated and inserted for the header field and correspondingly the DC bit should be set for the data field. This field is only used by the GRSPW when the CRC logic is available (*rmap* or *rmapcrc* VHDL generic set to 1). The header CRC will be calculated from the data fetched from the header pointer and the data CRC is generated from data fetched from the data pointer. The CRCs are appended after the corresponding fields. The NON-CRC bytes field is set to the number of bytes in the beginning of the header field that should not be included in the CRC calculation.

The CRCs are sent even if the corresponding length is zero, but when both lengths are zero no packet is sent not even an EOP.

82.5.5.4 Starting transmissions

When the descriptors have been initialized, the transmit enable bit in the DMA control register has to be set to tell the port to start transmitting. New descriptors can be activated in the table on the fly (while transmission is active). Each time a set of descriptors is added the transmit enable register bit should be set. This has to be done because each time the core encounters a disabled descriptor this register bit is set to 0.

Table 1448. TXDMA transmit descriptor word 0 (address offset 0x0)

31 18	17	16	15	14	13	12	11 8	7 0
RESERVED	DC	HC	LE	IE	WR	EN	NONCRCLEN	HEADERLEN

31: 18	RESERVED
17	Append data CRC (DC) - Append CRC calculated according to the RMAP specification after the data sent from the data pointer. The CRC covers all the bytes from this pointer. A null CRC will be sent if the length of the data field is zero.
16	Append header CRC (HC) - Append CRC calculated according to the RMAP specification after the data sent from the header pointer. The CRC covers all bytes from this pointer except a number of bytes in the beginning specified by the non-crc bytes field. The CRC will not be sent if the header length field is zero.
15	Link Error (LE) - An error occurred during the transmission of this packet. This bit is also set if the transmission was aborted while processing this packet.
14	Interrupt enable (IE) - If set, an interrupt will be generated when the packet has been transmitted and the transmitter interrupt enable bit in the DMA control register is set.
13	Wrap (WR) - If set, the descriptor pointer will wrap and the next descriptor read will be the first one in the table (at the base address). Otherwise the pointer is increased with 0x10 to use the descriptor at the next higher memory location.
12	Enable (EN) - Enable transmitter descriptor. When all control fields (address, length, wrap and crc) are set, this bit should be set. While the bit is set the descriptor should not be touched since this might corrupt the transmission. The GRSPW clears this bit when the transmission has finished.
11: 8	Non-CRC bytes (NONCRCLEN)- Sets the number of bytes in the beginning of the header which should not be included in the CRC calculation. This is necessary when using path addressing since one or more bytes in the beginning of the packet might be discarded before the packet reaches its destination.
7: 0	Header length (HEADERLEN) - Header Length in bytes. If set to zero, the header is skipped.

Table 1449. TXDMA transmit descriptor word 1 (address offset 0x4)

31 0
HEADERADDRESS

31: 0	Header address (HEADERADDRESS) - Address from where the packet header is fetched. Does not need to be word aligned.

Table 1450. TXDMA transmit descriptor word 2 (address offset 0x8)

31 24	23 0
RESERVED	DATALEN

31: 24	RESERVED
23: 0	Data length (DATALEN) - Length of data part of packet. If set to zero, no data will be sent. If both data- and header-lengths are set to zero no packet will be sent.

Table 1451. TXDMA transmit descriptor word 3(address offset 0xC)

31	0
DATAADDRESS	

31: 0 Data address (DATAADDRESS) - Address from where data is read. Does not need to be word aligned.

82.5.5.5 The transmission process

When the txen bit is set the port starts reading descriptors immediately. The number of bytes indicated are read and transmitted. When a transmission has finished, status will be written to the first field of the descriptor and a packet sent bit is set in the DMA control register. If an interrupt was requested it will also be generated. Then a new descriptor is read and if enabled a new transmission starts, otherwise the transmit enable bit is cleared and nothing will happen until it is enabled again.

82.5.5.6 The descriptor table address register

The internal pointer which is used to keep the current position in the descriptor table can be read and written through the APB interface. This pointer is set to zero during reset and is incremented each time a descriptor is used. It wraps automatically when the 1024 bytes limit for the descriptor table is reached or it can be set to wrap earlier by setting a bit in the current descriptor.

The descriptor table register can be updated with a new table anytime when no transmission is active. No transmission is active if the transmit enable bit is zero and the complete table has been sent or if the table is aborted (explained below). If the table is aborted one has to wait until the transmit enable bit is zero before updating the table pointer.

82.5.5.7 Error handling

82.5.5.7.1Abort Tx

The DMA control register contains a bit called Abort TX which if set causes the current transmission to be aborted, the packet is truncated and an EEP is inserted. This is only useful if the packet needs to be aborted because of congestion on the SpaceWire network. If the congestion is on the AHB bus this will not help (This should not be a problem since AHB slaves should have a maximum of 16 wait-states). The aborted packet will have its LE bit set in the descriptor. The transmit enable register bit is also cleared and no new transmissions will be done until the transmitter is enabled again.

82.5.5.7.2AHB error

When an AHB error is encountered during transmission the currently active DMA channel is disabled and the transmitter goes to the idle mode. A bit in the DMA channel's control/status register is set to indicate this error condition and, if enabled, an interrupt will also be generated. Further error handling depends on what state the transmitter DMA engine was in when the AHB error occurred. If the descriptor was being read the packet transmission had not been started yet and no more actions need to be taken.

If the AHB error occurs during packet transmission the packet is truncated and an EEP is inserted. Lastly, if it occurs when status is written to the descriptor the packet has been successfully transmitted but the descriptor is not written and will continue to be enabled (this also means that no error bits are set in the descriptor for AHB errors).

The client using the channel has to correct the AHB error condition and enable the channel again. No more AHB transfers are done again from the same unit (receiver or transmitter) which was active during the AHB error until the error state is cleared and the unit is enabled again.

82.5.6 RMAP target

The Remote Memory Access Protocol (RMAP) is used to implement access to resources on the AHB bus via the SpaceWire Link. Some common operations are reading and writing to memory, registers and FIFOs. The port has an optional hardware RMAP target which is enabled with a VHDL generic. This section describes the target implementation.

82.5.6.1 Fundamentals of the protocol

RMAP is a protocol which is designed to provide remote access via a SpaceWire network to memory mapped resources on a SpaceWire node. It has been assigned protocol ID 0x01. It provides three operations write, read and read-modify-write. These operations are posted operations which means that a source does not wait for an acknowledge or reply. It also implies that any number of operations can be outstanding at any time and that no timeout mechanism is implemented in the protocol. Time-outs must be implemented in the user application which sends the commands. Data payloads of up to 16 Mb - 1 is supported in the protocol. A destination can be requested to send replies and to verify data before executing an operation. A complete description of the protocol is found in the RMAP standard.

82.5.6.2 Implementation

The port includes a target for RMAP commands which processes all incoming packets with protocol ID = 0x01, type field (bit 7 and 6 of the 3rd byte in the packet) equal to 01b and an address falling in the range set by the default address and mask register. When such a packet is detected it is not stored to the DMA channel, instead it is passed to the RMAP receiver.

The target implements all three commands defined in the standard with some restrictions. Support is only provided for 32-bit big-endian systems. This means that the first byte received is the msb in a word. The target will not receive RMAP packets using the extended protocol ID which are always dumped to the DMA channel.

The RMAP receiver processes commands. If they are correct and accepted the operation is performed on the AHB bus and a reply is formatted. If an acknowledge is requested the RMAP transmitter automatically send the reply. RMAP transmissions have priority over DMA channel transmissions.

There is a user accessible destination key register which is compared to destination key field in incoming packets. If there is a mismatch and a reply has been requested the error code in the reply is set to 3. Replies are sent if and only if the ack field is set to ‘1’.

When a failure occurs during a bus access the error code is set to 1 (General Error). There is predetermined order in which error-codes are set in the case of multiple errors in the core. It is shown in table 1452.

Table 1452. The order of error detection in case of multiple errors in the AMBA port. The error detected first has number 1.

Detection Order	Error Code	Error
1	12	Invalid destination logical address
2	2	Unused RMAP packet type or command code
3	3	Invalid destination key
4	9	Verify buffer overrun
5	11	RMW data length error
6	10	Authorization failure
7*	1	General Error (AHB errors during non-verified writes)
8	5/7	Early EOP / EEP (if early)
9	4	Invalid Data CRC
10	1	General Error (AHB errors during verified writes or RMW)
11	7	EEP
12	6	Too Much Data
*The AHB error is not guaranteed to be detected before Early EOP/EEP or Invalid Data CRC. For very long accesses the AHB error detection might be delayed causing the other two errors to appear first.		

Read accesses are performed on the fly, that is they are not stored in a temporary buffer before transmitting. This means that the error code 1 will never be seen in a read reply since the header has already been sent when the data is read. If the AHB error occurs the packet will be truncated and ended with an EEP.

Errors up to and including Invalid Data CRC (number 9) are checked before verified commands. The other errors do not prevent verified operations from being performed.

The details of the support for the different commands are now presented. All defined commands which are received but have an option set which is not supported in this specific implementation will not be executed and a possible reply is sent with error code 10.

82.5.6.3 Write commands

The write commands are divided into two subcategories when examining their capabilities: verified writes and non-verified writes. Verified writes have a length restriction of 4 bytes and the address must be aligned to the size. That is 1 byte writes can be done to any address, 2 bytes must be halfword aligned, 3 bytes are not allowed and 4 bytes writes must be word aligned. Since there will always be only on AHB operation performed for each RMAP verified write command the incrementing address bit can be set to any value.

Non-verified writes have no restrictions when the incrementing bit is set to 1. If it is set to 0 the number of bytes must be a multiple of 4 and the address word aligned. There is no guarantee how many words will be written when early EOP/EEP is detected for non-verified writes.

82.5.6.4 Read commands

Read commands are performed on the fly when the reply is sent. Thus if an AHB error occurs the packet will be truncated and ended with an EEP. There are no restrictions for incrementing reads but non-incrementing reads have the same alignment restrictions as non-verified writes. Note that the “Authorization failure” error code will be sent in the reply if a violation was detected even if the length field was zero. Also note that no data is sent in the reply if an error was detected i.e. if the status field is non-zero.

82.5.6.5 RMW commands

All read-modify-write sizes are supported except 6 which would have caused 3 B being read and written on the bus. The RMW bus accesses have the same restrictions as the verified writes. As in the verified write case, the incrementing bit can be set to any value since only one AHB bus operation will be performed for each RMW command. Cargo too large is detected after the bus accesses so this error will not prevent the operation from being performed. No data is sent in a reply if an error is detected i.e. the status field is non-zero.

82.5.6.6 Control

The RMAP target mostly runs in the background without any external intervention, but there are a few control possibilities.

There is an enable bit in the control register of the core which can be used to completely disable the RMAP target. When it is set to '0' no RMAP packets will be handled in hardware, instead they are all stored to the DMA channel.

There is a possibility that RMAP commands will not be performed in the order they arrive. This can happen if a read arrives before one or more writes. Since the target stores replies in a buffer with more than one entry several commands can be processed even if no replies are sent. Data for read replies is read when the reply is sent and thus writes coming after the read might have been performed already if there was congestion in the transmitter. To avoid this the RMAP buffer disable bit can be set to force the target to only use one buffer which prevents this situation.

The last control option for the target is the possibility to set the destination key which is found in a separate register.

Table 1453. AMBA port hardware RMAP handling of different packet type and command fields.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Command	Action
Reserved	Command / Response	Write / Read	Verify data before write	Acknow-ledge	Increment Address		
0	0	-	-	-	-	Response	Stored to DMA-channel.
0	1	0	0	0	0	Not used	Does nothing. No reply is sent.
0	1	0	0	0	1	Not used	Does nothing. No reply is sent.
0	1	0	0	1	0	Read single address	Executed normally. Address has to be word aligned and data size a multiple of four. Reply is sent. If alignment restrictions are violated error code is set to 10.
0	1	0	0	1	1	Read incre-menting address.	Executed normally. No restrictions. Reply is sent.
0	1	0	1	0	0	Not used	Does nothing. No reply is sent.
0	1	0	1	0	1	Not used	Does nothing. No reply is sent.
0	1	0	1	1	0	Not used	Does nothing. Reply is sent with error code 2.

Table 1453. AMBA port hardware RMAP handling of different packet type and command fields.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Command	Action
Reserved	Command / Response	Write / Read	Verify data before write	Acknow-ledge	Increment Address		
0	1	0	1	1	1	Read-Mod-ify-Write increment-ing address	Executed normally. If length is not one of the allowed rmw val-ues nothing is done and error code is set to 11. If the length was correct, alignment restric-tions are checked next. 1 byte can be rmw to any address. 2 bytes must be halfword aligned. 3 bytes are not allowed. 4 bytes must be word aligned. If these restrictions are violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	0	0	0	Write, sin-gle-address, do not verify before writ-ing, no acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done. No reply is sent.
0	1	1	0	0	1	Write, incre-menting address, do not verify before writ-ing, no acknowledge	Executed normally. No restric-tions. No reply is sent.
0	1	1	0	1	0	Write, sin-gle-address, do not verify before writ-ing, send acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	0	1	1	Write, incre-menting address, do not verify before writ-ing, send acknowledge	Executed normally. No restric-tions. If AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	0	0	Write, single address, ver-ify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restric-tions apply as for rmw. No reply is sent.

Table 1453. AMBA port hardware RMAP handling of different packet type and command fields.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Command	Action
Reserved	Command / Response	Write / Read	Verify data before write	Acknow-ledge	Increment Address		
0	1	1	1	0	1	Write, incre-menting address, ver-ify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restric-tions apply as for rmw. If they are violated nothing is done. No reply is sent.
0	1	1	1	1	0	Write, single address, ver-ify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are vio-lated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	1	1	Write, incre-menting address, ver-ify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are vio-lated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
1	0	-	-	-	-	Unused	Stored to DMA-channel.
1	1	-	-	-	-	Unused	Stored to DMA-channel.

82.5.7 AMBA interface

The AMBA interface consists of an APB interface, an AHB master interface and DMA FIFOs. The APB interface provides access to the user registers which are described in section 82.7. The DMA engines have 32-bit wide FIFOs to the AHB master interface which are used when reading and writing to the bus.

The transmitter DMA engine reads data from the bus in bursts which are half the FIFO size in length. A burst is always started when the FIFO is half-empty or if it can hold the last data for the packet. The burst containing the last data might have shorter length if the packet is not an even number of bursts in size.

The receiver DMA works in the same way except that it checks if the FIFO is half-full and then performs a burst write to the bus which is half the FIFO size in length. The last burst might be shorter. If the rmap or rxunaligned VHDL generics are set to 1 the interface also handles byte accesses. Byte accesses are used for non word-aligned buffers and/or packet lengths that are not a multiple of four bytes. There might be 1 to 3 single byte writes when writing the beginning and end of the received packets.

82.5.7.1 APB slave interface

As mentioned above, the APB interface provides access to the user registers which are 32-bits in width. The accesses to this interface are required to be aligned word accesses. The result is undefined if this restriction is violated.

82.5.7.2 AHB master interface

The port contains a single master interface which is used by both the transmitter and receiver DMA engines. The arbitration algorithm between the channels is done so that if the current owner requests the interface again it will always acquire it. This will not lead to starvation problems since the DMA engines always deassert their requests between accesses.

The AHB accesses are always word accesses (HSIZE = 0x010) of type incremental burst with unspecified length (HBURST = 0x001) if VHDL generics rmap and rxunaligned are disabled. The AHB accesses can be of size byte, halfword and word (HSIZE = 0x000, 0x001, 0x010) otherwise. Byte and halfword accesses are always NONSEQ. Note that read accesses are always word accesses (HSIZE = 0x010), which can result in destructive read.

The burst length will be half the AHB FIFO size except for the last transfer for a packet which might be smaller. Shorter accesses are also done during descriptor reads and status writes.

The AHB master also supports non-incrementing accesses where the address will be constant for several consecutive accesses. HTRANS will always be NONSEQ in this case while for incrementing accesses it is set to SEQ after the first access. This feature is included to support non-incrementing reads and writes for RMAP.

If the core does not need the bus after a burst has finished there will be one wasted cycle (HTRANS = IDLE).

BUSY transfer types are never requested and the port provides full support for ERROR, RETRY and SPLIT responses.

82.5.8 Synthesis and hardware

82.5.8.1 Clocking

The AMBA ports run on the same clock as the router switch matrix.

82.5.8.2 Fault-tolerance

The ports can optionally be implemented with fault-tolerance against SEU errors in the FIFO memories. The fault-tolerance is enabled through the *ft* VHDL generic. Possible options are byte parity protection (*ft = 1*) or TMR (*ft = 2*).

82.5.8.3 Technology mapping

The core has three generics for technology mapping: *tech*, *techfifo* and *memtech*. *Tech* selects the technology used for the clock buffers and also adds reset to some registers for technologies where they would otherwise cause problems with gate-level simulations. *Techfifo* selects whether *memtech* should be used to select the technology for the FIFO memories (the RMAP buffer is not affected by the this generic) or if they should be inferred. *Tech* and *memtech* can be set to any value from 0 to NTECH as defined in the GRLIB.TECH package.

82.5.8.4 RAM usage

The core maps all RAM memories on the syncram_2p component if the *ft* generic is 0 and to the syncram_2pft component for other values. The syncrams are located in the technology mapping library (TECHMAP). The organization of the different memories are described below. If techfifo and/or memtech is set to 0 the synthesis tool will infer the memories. Either RAM blocks or flip-flops will be used depending on the tool and technology. The number of flip-flops used is *syncram depth x syncram width* for all the different memories. The receiver AHB FIFO with fifosize 32 will for example use 1024 flips-flops.

82.5.8.4.1Receiver AHB FIFO

The receiver AHB FIFO consists of one syncram_2p block with a width of 32-bits. The depth is determined by the configured FIFO depth. Table 1454 shows the syncram organization for the allowed configurations.

*Table 1454.*syncram_2p sizes for receiver AHB FIFO.

Fifosize	Syncram_2p organization
4	4x32
8	8x32
16	16x32
32	32x32

82.5.8.4.2Transmitter AHB FIFO

The transmitter AHB FIFO consists of one syncram_2p block with a width of 32-bits. The depth is determined by the configured FIFO depth. Table 1455 shows the syncram organization for the allowed configurations.

*Table 1455.*syncram_2p sizes for transmitter AHB FIFO.

Fifosize	Syncram_2p organization
4	4x32
8	8x32
16	16x32
32	32x32

82.5.8.4.3Receiver N-Char FIFO

The receiver N-Char FIFO consists of one syncram_2p block with a width of 10-bits. The depth is determined by the configured FIFO depth. Table 1456 shows the syncram organization for the allowed configurations.

*Table 1456.*syncram_2p sizes for the receiver N-Char FIFO.

Fifosize	Syncram_2p organization
16	16x10
32	32x10
64	64x10

82.5.8.4.4RMAP buffer

The RMAP buffer consists of one syncram_2p block with a width of 8-bits. The depth is determined by the number of configured RMAP buffers. Table 1457 shows the syncram organization for the allowed configurations.

*Table 1457.*syncram_2p sizes for RMAP buffer memory.

RMAP buffers	Syncram_2p organization
2	64x8
4	128x8
8	256x8

82.5.9 Registers

The port is programmed through registers mapped into APB address space. The addresses in the table below are offsets from each port's base address. The actual AMBA AHB address used to access the port is determined as follows: The AMBA ports' registers are accessed through an APB interface which resides on the APB bus. The APB bus is connected to the AHB bus using an APB controller whose AHB address determines the first base address for the AMBA ports.

See the grlib manual and APB controller manual on how to determine this base address. The starting base address on the APB bus for each AMBA port APB interface is set by the paddr and pmask VHDL generics. The starting paddr value is then incremented by one for each port. Finally a registers address is determined by adding the APB controller's AHB base address, the ports APB base address and the registers offset.

*Table 1458.*AMBA port registers

APB address offset	Register name	Acronym
0x00	AMBA port Control	RTR.AMBACTRL
0x40	AMBA port Status	RTR.AMBASTS
0x08	AMBA port Default address	RTR.AMBADEFADDR
0x0C	RESERVED	
0x10	AMBA port Destination key	RTR.AMBADKEY
0x14	AMBA port Time-code	RTR.AMBATC
0x18	RESERVED	
0x20, 0x40, 0x60, 0x80	AMBA port DMA control/status, channels 1 - 4*	RTR.AMBADMACTRL
0x24, 0x44, 0x64, 0x84	AMBA port DMA RX maximum length, channels 1 - 4*	RTR.AMBADMAMAXLEN
0x28, 0x48, 0x68, 0x88	AMBA port DMA transmit descriptor table address, channels 1 - 4 *	RTR.AMBADMATXDESC
0x2C, 0x4C, 0x6C, 0x8C	AMBA port DMA receive descriptor table address, channels 1 - 4 *	RTR.AMBADMARXDESC
0x30, 0x50, 0x70, 0x90	AMBA port DMA address, channels 1 - 4 *	RTR.AMBADMAADDR
0x34, 0x54, 0x74, 0x94	RESERVED	
0x38, 0x58, 0x78, 0x98	RESERVED	
0x3C, 0x5C, 0x7C, 0x9C	RESERVED	
0xA0	AMBA port Distributed interrupt control	RTR.AMBAINTCTRL
0xA4	AMBA port Interrupt receive	RTR.AMBAINTRX
0xA8	AMBA port Interrupt acknowledgment / extended interrupt receive	RTR.AMBAACKRX
0xAC	AMBA port Interrupt timeout, interrupt 0-31	RTR.AMBAINTTO0
0xB0	AMBA port Interrupt timeout, interrupt 32-63	RTR.AMBAINTTO1
0xB4	AMBA port Interrupt mask, interrupt 0-31	RTR.AMBAINTMSK0
0xB8	AMBA port Interrupt mask, interrupt 32-63	RTR.AMBAINTMSK1
0xBC - 0xFF	RESERVED	

* One identical register per DMA channel. Register is only described once

Table 1459. 0x00 - RTR.AMBACTRL - AMBA port Control

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RA	RX	RC	NCH		R	CC	DI	ME	RESERVED					RD	RE	RESERVED							TQ	R	RS	PM	TI	IE	RESERVED			
*	*	*	*		0	*	*	0	0x00					0	1	0x00							0	0	0	0	0	0	0x0			
r	r	r	r		r	r	r	rw	r					rw	rw	r							rw	r	rw	rw	rw	rw	r			

31 RMAP available (RA) - Set to 1 if the RMAP target is implemented.

30 RX unaligned access (RX) - Set to 1 if unaligned writes are available for the receiver.

29 RMAP CRC available (RC) - Set to 1 if RMAP CRC is enabled.

28: 27 Number of DMA channels (NCH) - The number of available DMA channels minus one.

26 RESERVED

25 CCSDS CRC available (CC) - Set to 1 if CCSDS CRC is enabled.

24 Distributed interrupt support (DI) - Set to 1 if distributed interrupts are supported.

23 Memory error truncation enable (ME) - If set to 1, a packet being transmitted will be truncated with an EEP if an error occur while reading from the AMBA port's TX FIFO.

22: 18 RESERVED

17 RMAP buffer disable (RD) - If set only one RMAP buffer is used. This ensures that all RMAP commands will be executed consecutively.

16 RMAP Enable (RE) - Enable RMAP target.

15: 9 RESERVED

8 Tick-out IRQ (TQ) - Generate interrupt when a valid time-code is received if the time-code also matches the time-code filter specified by the RTR.AMBATIME.TCMSK and RTR.AMBATIME.TCVAL fields.

7 Time-code tick-out enable (TO) - If set to 1, the internal tickout signal is set when a valid time-code is received. if the time-code also matches the time-code filter specified by the RTR.AMBATIME.TCMSK and RTR.AMBATIME.TCVAL fields.

6 Reset (RS) - Make complete reset of the SpaceWire node. Self clearing.

5 Promiscuous Mode (PM) - Enable Promiscuous mode.

4 Tick In (TI) - The host can generate a tick by writing a one to this field. This will increment the timer counter and the new value is transmitted after the current character is transferred. A tick can also be generated by asserting the tick_in signal.

3 Interrupt Enable (IE) - If set, an interrupt is generated when bit 8 is set and its corresponding event occurs.

2: 0 RESERVED

Frontgrade Gaisler AB
Kungsgatan | SE-411 19 | Goteborg | Sweden
+46 31 7758650 | frontgrade.com/gaisler

Table 1460. 0x04 - RTR.AMBASTS - AMBA port Status

31	30 28	27 26	25 24	23 13	12	11 9	8	7	7 0	
R	NIRQ	NRXD	NTXD	RESERVED	ME	RESERVED	EE	IA	RESERVED	TO
0	*	*	*	0x000	0	0x0	0	0	0x00	0
r	r	r	r	r	wc	r	wc	wc	r	wc

31 RESERVED

30: 28 Number of interrupts (NIRQ) - Shows the number of support distributed interrupt, according to the formula 2^{NIRQ}.

27: 26 Number of receive descriptors (NRXD) - Shows the size of the DMA receive descriptor table.

25: 24 Number of transmit descriptors (NTXD) - Shows the size of the DMA transmit descriptor table.

23: 13 RESERVED

12 Memory error packet truncation (ME) - This bit is set to one when a transmitted packet is truncated with an EEP due to a memory error in the AMBA ports' TX FIFO.

11: 9 RESERVED

8 Early EOP/EEP (EE) - Set to one when a packet is received with an EOP after the first byte for a non-RMAP packet and after the second byte for an RMAP packet.

7 Invalid Address (IA) - Set to one when a packet is received with an invalid destination address field, i.e it does not match the nodeaddr register.

6: 1 RESERVED

0 Tick Out (TO) - A new time count value was received and is stored in the time counter field.

Table 1461. 0x08 - RTR.AMBADEFADDR - AMBA port Default address

31 16	15 8	7 0
RESERVED	DEFMASK	DEFADDR
0x0000	0x00	0xFE
r	rw	rw

31: 8 RESERVED

15: 8 Default mask (DEFMASK) - Default mask used for node identification on the SpaceWire network. This field is used for masking the address before comparison. Both the received address and the DEFADDR field are anded with the inverse of DEFMASK before the address check.

7: 0 Default address (DEFADDR) - Default address used for node identification on the SpaceWire network. Reset value: 254.

Table 1462. 0x10 - RTR.AMBADKEY - AMBA port Destination key

31 8	7 0
RESERVED	DESTKEY
0x000000	0x00
r	rw

31: 8 RESERVED

7: 0 Destination key (DESTKEY) - RMAP destination key.

Table 1463. 0x14 - RTR.AMBATC - AMBA port Time-code

31 24	23 16	15 8	7 6	5 0
TCMSK	TCVAL	RESERVED	TCTRL	TIMECNT

Table 1463. 0x14 - RTR.AMBATC - AMBA port Time-code

0x00	0x00	0x00	0x0	0x00
rw	rw	r	rw	rw

31: 24 Time-code filter mask (TCMSK) - If a bit in this field is set to 1 then the corresponding bit in the RTR.AMBATIME.TCVAL field must match the value of the same bit in a received time-code in order for that time-code to generate an AMBA IRQ or a pulse on the internal tickout signals connected to the general purpose timers.

23: 16 Time-code filter value (TCVAL) - For each bit set to 1 in the RTR.AMBATIME.TCMSK, the corresponding bit in this field must match the value of the same bit of a received time-code in order to that time-code to generate and AMBA IRQ, or a pulse on the internal tickout signals connected to the general purpose timers.

15: 8 RESERVED

7: 6 Time control flags (TCTRL) - The current value of the time control flags. Sent with time-code resulting from a tick-in. Received control flags are also stored in this register.

5: 0 Time counter (TIMECNT) - The current value of the system time counter. It is incremented for each tick-in and the incremented value is transmitted. The register can also be written directly but the written value will not be transmitted. Received time-counter values are also stored in this register

Table 1464. 0x20,0x40,0x60,0x80 - RTR.AMBADMACTRL - AMBA port DMA control/status

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTNUM						RES		EP	TR	IE	IT	RP	TP	RES		SP	SA	EN	NS	RD	RX	AT	RA	TA	PR	PS	AI	RI	TI	RE	TE
0x00						0x0		0	0	0	0	0	0	0x0		0	0	0	0	0	0	0	0	0	0	0	NR	NR	NR	0	0
rw						r		wc	wc	rw	rw	wc	wc	r		rw	rw	rw	rw	rw	r	rw	wc	wc	wc	wc	rw	rw	rw	rw	rw

31: 26 Interrupt number (INTNUM) - The interrupt number used for this DMA channel when sending a distributed interrupt code that was generated due to any of the events maskable by the RTR.AMBADMACTRL.IE and RTR.AMBADMACTRL.IT bits.

25: 24 RESERVED

23 EEP termination (EP) - Set to 1 when a received packet for the corresponding DMA channel ended with an Error End of Packet (EEP) character.

22 Truncated (TR) - Set to 1 when a received packet for the corresponding DMA channel is truncated due to a maximum length violation.

21 Interrupt transmit enable on EEP (IE) - When set to 1, the distributed interrupt code specified in the RTR.AMBADMACTRL.INTNUM field is generated when a received packet on this DMA channel ended with an Error End of Packet (EEP) character.

Table 1464. 0x20,0x40,0x60,0x80 - RTR.AMBADMACTRL - AMBA port DMA control/status

20	Interrupt-code transmit enable on truncation (IT) - When set to 1, the distributed interrupt code specified in the RTR.AMBADMACTRL.INTNUM field is generated when a received packet on this DMA channel is truncated due to a maximum length violation.
19	Receive packet IRQ (RP) - This bit is set to 1 when an AMBA interrupt was generated due to the fact that a packet was received for the corresponding DMA channel.
18	Transmit packet IRQ (TP) - This bit is set to 1 when an AMBA interrupt was generated due to the fact that a packet was transmitted for the corresponding DMA channel.
17: 16	RESERVED
15	Strip pid (SP) - Remove the pid byte (second byte) of each packet. The address byte (first byte) will also be removed when this bit is set independent of the SA bit.
14	Strip addr (SA) - Remove the addr byte (first byte) of each packet.
13	Enable addr (EN) - Enable separate node address for this channel.
12	No spill (NS) - If cleared, packets will be discarded when a packet is arriving and there are no active descriptors. If set, the GRSPWROUTER will wait for a descriptor to be activated.
11	Rx descriptors available (RD) - Set to one, to indicate to the GRSPWROUTER that there are enabled descriptors in the descriptor table. Cleared by the GRSPWROUTER when it encounters a disabled descriptor:
10	RX active (RX) - Is set to ‘1’ if a reception to the DMA channel is currently active otherwise it is ‘0’.
9	Abort TX (AT) - Set to one to abort the currently transmitting packet and disable transmissions. If no transmission is active the only effect is to disable transmissions. Self clearing.
8	RX AHB error (RA) - An error response was detected on the AHB bus while this receive DMA channel was accessing the bus. The AHB error interrupt (AI) field must be set to ’1’ for the RA field to be set.
7	TX AHB error (TA) - An error response was detected on the AHB bus while this transmit DMA channel was accessing the bus.
6	Packet received (PR) - This bit is set each time a packet has been received. Never cleared by the DMA channel.
5	Packet sent (PS) - This bit is set each time a packet has been sent. Never cleared by the DMA channel.
4	AHB error interrupt (AI) - If set, an interrupt will be generated each time an AHB error occurs when this DMA channel is accessing the bus.
3	Receive interrupt (RI) - If set, an interrupt will be generated each time a packet has been received. This happens both if the packet is terminated by an EEP or EOP.
2	Transmit interrupt (TI) - If set, an interrupt will be generated each time a packet is transmitted. The interrupt is generated regardless of whether the transmission was successful or not.
1	Receiver enable (RE) - Set to one when packets are allowed to be received to this channel.
0	Transmitter enable (TE) - Write a one to this bit each time new descriptors are activated in the table. Writing a one will cause the DMA channel to read a new descriptor and try to transmit the packet it points to. This bit is automatically cleared when the DMA channel encounters a descriptor which is disabled.

Table 1465. 0x24,0x44,0x64,0x84 - RTR.AMBADMAMAXLEN - AMBA port DMA RX maximum length

31 25	24 2	1 0
RESERVED	RXMAXLEN	RES
0x00	N/R	0x0
r	rw	r

31: 25	RESERVED
24: 2	RX maximum length (RXMAXLEN) - Receiver packet maximum length in 32-bit words.
1: 0	RESERVED

Table 1466. 0x28,0x48,0x68,0x88 - RTR.AMBADMATXDESC - AMBA port DMA transmit descriptor table address

31 N+1	N 4	3 0
DESCBASEADDR	DESCSEL	RESERVED

Table 1466. 0x28,0x48,0x68,0x88 - RTR.AMBADMATXDESC - AMBA port DMA transmit descriptor table address

N/R	0x00	0x0
rw	rw	r

31: *N+1* Descriptor table base address (DESCBASEADDR) - Sets the base address of the descriptor table. The size of this field is determined by the value of the VHDL generic *num_txdesc*.

N: 4 Descriptor selector (DESCSEL) - Offset into the descriptor table. Shows which descriptor is currently used by the GRSPWROUTER For each new descriptor read, the selector will increase with 16 and eventually wrap to zero again. The width of this field is log2(*num_txdesc*). N is set to log2(*num_txdesc*) + 3.

3: 0 RESERVED

Table 1467. 0x2C,0x4C,0x6C,0x8C - RTR.AMBADMARXDESC - AMBA port DMA receive descriptor table address

31 *N+1*	*N* 3	2 0
DESCBASEADDR	DESCSEL	RESERVED
N/R	0x00	0x0
rw	rw	r

31: *N+1* Descriptor table base address (DESCBASEADDR) - Sets the base address of the descriptor table. Not reset. The size of this field is determined by the value of the VHDL generic *num_rxdesc*.

N: 3 Descriptor selector (DESCSEL) - Offset into the descriptor table. Shows which descriptor is currently used by the GRSPWROUTER. For each new descriptor read, the selector will increase with 8 and eventually wrap to zero again. Reset value: 0. The width of this field is log2(*num_rxdesc*). N is set to log2(*num_rxdesc*) + 2.

2: 0 RESERVED

Table 1468. 0x30,0x50,0x70,0x90 - RTR.AMBADMAADDR - AMBA port DMA address

31 16	15 8	7 0
RESERVED	MASK	ADDR
0x0000	N/R	N/R
r	rw	rw

31: 8 RESERVED

15: 8 Mask (MASK) - Mask used for node identification on the SpaceWire network. This field is used for masking the address before comparison. Both the received address and the ADDR field are anded with the inverse of MASK before the address check.

7: 0 Address (ADDR) - Address used for node identification on the SpaceWire network for the corresponding dma channel when the EN bit in the DMA control register is set.

Table 1469. 0xA0 - RTR.AMBAINTCTRL - AMBA port Distributed interrupt control

31 26	25	24	23	22 21	20	19	18	17 16	15	14	13	12 8	7	6	5 0
INTNUM	R	EE	IA	RES	TQ	AQ	IQ	RES	AA	AT	IT	RESERVED	ID	II	TXINT
0x00	0	0	0	0x0	0	0	0	0x0	0	1	1	0x00	0	0	0x00
rw	r	rw	rw	r	rw	rw	rw	r	rw	rw	rw	r	wc	rw*	rw

31: 26 Interrupt number (INTNUM) - The interrupt-number used when sending an interrupt code that was generated due to any of the events maskable by the RTR.AMBAINCTRL.ER and RTR.AMBAINTCTRL.IA bits. Note that when RTR.RTRCFG.EE = 0 (interrupt with acknowledgment mode), this field must no be set to a value greater than 31.

25 RESERVED

Table 1469. 0xA0 - RTR.AMBAINTCTRL - AMBA port Distributed interrupt control

24	Interrupt transmit on early EOP/EEP (EE) - If set to 1, a distributed interrupt code with the interrupt number specified in the RTR.AMBAINTCTRL.INTNUM field is sent each time an event occurs such that the STS.EE bit is set to 1 (even if the bit was already set when the event occurred).
23	Interrupt transmit on invalid address (IA) - If set to 1, a distributed interrupt code with the interrupt number specified in the RTR.AMBAINTCTRL.INTNUM field is sent each time an event occurs such that the STS.IA bit is set to 1 (even if the bit was already set when the event occurred).
22: 21	RESERVED
20	Interrupt timeout IRQ enable (TQ) - When set to 1, an AMBA interrupt is generated when a bit in the RTR.AMBAINTTO0 or RTR.AMBAINTTO1 registers is set. Note that the RTR.AMBACTRL.IE bit also must be set for this bit to have any effect.
19	Interrupt acknowledgment / extended interrupt receive IRQ enable (AQ) - When set to 1, an AMBA interrupt is generated when an interrupt acknowledgment code or extended interrupt code is received such that a bit in the RTR.AMBAACKRX register is set to 1 (even if the bit was already set when the code was received).
18	Interrupt-code receive IRQ enable (IQ) - When set to 1, an AMBA interrupt is generated when an interrupt code is received such that a bit in the RTR.AMBAINTRX register is set to 1 (even if the bit was already set when the code was received).
17: 16	RESERVED
15	Handle all interrupt acknowledgment codes (AA) - Is set to 0, only those received interrupt acknowledgment codes that match an interrupt code sent by software are handled. If set to 1, all received interrupt acknowledgment codes are handled.
14	Interrupt acknowledgment / extended interrupt tickout enable (AT) - When set to 1, the internal tickout signal from this AMBA port is set when an interrupt acknowledgment code or extended interrupt code is received such that a bit in the RTR.AMBAACKRX register is set to 1 (even if the bit was already set when the code was received).
13	Interrupt tickout enable (IT) - When set to 1, the internal tickout signal from this AMBA port is set when an interrupt code is received such that a bit in the RTR.AMBAINTRX register is set to 1 (even if the bit was already set when the code was received).
12: 8	RESERVED
7	Interrupt discarded (ID) - This bit is set to 1 when a distributed interrupt code that software tried to send by writing the RTR.AMBAINTCTRL.II bit was discarded by the routers switch matrix. There is a maximum of ten clock cycle delay between the RTR.AMBAINTCTRL.II bit being written and this bit being set.
6	Interrupt-code tick-in (II) - When this field is written to 1 the distributed interrupt code specified by the RTR.AMBAINTCTRL.TXINT field will be sent out from the AMBA port to the routers switch matrix. This bit is automatically cleared and always reads '0'. Writing a '0' has no effect.
5: 0	Transmit distributed interrupt code (TXINT) - The distributed interrupt code that will be sent when the register RTR.AMBAINTCTRL.II is written with 1.

Table 1470. 0xA4 - RTR.AMBAINTRX - AMBA port Interrupt receive

31	0
RXIRQ	
0x00000000	
wc	

31: 0	Received interrupt code (RXIRQ) - Each bit corresponds to the interrupt number with the same number as the bit index. A position is set to 1 when an interrupt code is received for which the corresponding bit in the RTR.AMBAINTMSK0 register is set to 1.

Table 1471. 0xA8 - RTR.AMBAACKRX - AMBA port Interrupt acknowledgment / extended interrupt receive

31	0
RXACK	
0x00000000	
wc	

Table 1471. 0xA8 - RTR.AMBAACKRX - AMBA port Interrupt acknowledgment / extended interrupt receive

31: 0 Received interrupt acknowledgment code / extended interrupt code (RXACK) - When operating in extended interrupt mode (RTR.RTRCFG.EE = 1) then each bit corresponds to the interrupt number with the same number as the bit index plus 32, i.e bit 0 corresponds to interrupt number 32, bit 1 to interrupt number 33 etc. This bit gets set to 1 when an extended interrupt code is received for which the corresponding bit in the RTR.AMBAINTMSK1 register is set to 1.

When operating in interrupt with acknowledgment mode (RTR.RTRCFG.EE = 0) then each bit corresponds to the interrupt number with the same number as the bit index. This bit gets set to 1 an interrupt acknowledgment code is received for which the corresponding bit in the RTR.AMBAINTMSK0 register is set, and either if RTR.AMBAINTCTRL.AA is set to 1 or for which the matching interrupt code was sent by software.

Table 1472. 0xAC - RTR.AMBAINTTO0 - AMBA port Interrupt timeout, interrupt 0-31

31	0
INTTO	
0x00000000	
wc	

31: 0 Interrupt code timeout (INTTO) - Each bit corresponds to the interrupt number with the same number as the bit index. This bit is set to 1 when an interrupt code that was sent by software doesn't receive an interrupt acknowledgment code for the duration of a timeout period (specified in the RTR.ISRTIMER register), and if the corresponding bit in the RTR.AMBAINTMSK0 register is set.

Table 1473. 0xB0 - RTR.AMBAINTTO1 - AMBA port Interrupt timeout, interrupt 32-63

31	0
INTTO	
0x00000000	
wc	

31: 0 Extended interrupt code timeout (INTTO) - Each bit corresponds to the interrupt number with the same number as the bit index plus 32, i.e bit 0 corresponds to interrupt number 32, bit 1 to interrupt number 33 etc.. This bit is set to 1 when an extended interrupt code that was sent by software time out, i.e after the duration of a timeout period (specified in the RTR.ISRTIMER register), and if the corresponding bit in the RTR.AMBAINTMSK1 register is set.

Table 1474. 0xB4 - RTR.AMBAINTMSK0 - AMBA port Interrupt mask, interrupt 0-31

31	0
MASK	
0x00000000	
rw	

31: 0 Interrupt mask (MASK) - Each bit corresponds to the interrupt number with the same value as the bit index. If a bit is set to 0, all received interrupt codes and interrupt acknowledgment codes with the interrupt identifier corresponding to that bit is ignored. If a bit is set to 1, then the matching distributed interrupt code is handled.

Table 1475. 0xB8 - RTR.AMBAINTMSK1 - AMBA port Interrupt mask, interrupt 32-63

31	0
MASK	

Table 1475. 0xB8 - RTR.AMBAINTMSK1 - AMBA port Interrupt mask, interrupt 32-63

0x00000000
rw

31: 0 Interrupt mask (MASK) - Each bit corresponds to the interrupt number with the same value as the bit index. plus 32, i.e bit 0 corresponds to interrupt number 32, bit 1 to interrupt number 33 etc. If a bit is set to 0, all received extended interrupt codes with the interrupt identifier corresponding to that bit is ignored. If a bit is set to 1, then the matching distributed interrupt code is handled.

82.5.10 Vendor and device identifiers

The port has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x8A. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

82.6 Configuration port

The configuration port has port number 0. It consists of an RMAP target, AMBA AHB slave interface, SpaceWire Plug-and-Play interface, and a set of configuration and status registers.

82.6.1 RMAP target

82.6.1.1 Overview

The configuration port's RMAP target implements the RMAP protocol, as defined in the RMAP standard [RMAP]. Verified writes and reads up to 128 B, and read-modify-writes of 4 B (8 B if the mask field is included in the count) are supported. Replies from the configuration port are always sent to the port they arrived on, regardless of the values of the RMAP command's Initiator Logical Address field, and Reply Address field. The address space of the configuration port is specified in section 82.7.

Additional requirements on the RMAP commands imposed by the configuration port's RMAP target are:

- The Target Logical Address field must be 0xFE.
- The Address fields must contain a 4 B aligned address.
- The Extended Address field must be 0x00.
- Key field must be 0x00.
- For write and read commands the Data Length fields must contain a value that is a multiple of 4, ranging from 0 to 128 B.
- For read-modify-write commands the Data Length fields must contain a value of 0 or 8.
- For write commands the Verify Data Before Write bit in the Instruction field must be set to 1.

How the RMAP target handles commands that does not meet the above requirement is detailed in sections 82.6.1.2 and 82.6.1.4.

When an RMAP write command larger than 4 bytes is processed (i.e. more than one register written by the same command), the registers will not change value simultaneously. The command is buffered locally, since only verified write commands are allowed, and then read out from the buffer and written to the registers, with one CLK cycle delay between each register write. This needs to be considered when for example updating large parts of the routing table. Data traffic that arrives while the RMAP command is being processed may or may not be routed according to the new values depending on the address of the packet and how much of the RMAP command that has been processed. Note that the RMAP target is blocked while the RMAP write command is being processed, which means that data returned in an RMAP read command is always either the value before or after the complete RMAP write, never in between.

82.6.1.2 RMAP command support

Table 1476 lists all possible RMAP commands and shows how the configuration port's RMAP target handles them. An RMAP command will always have bits 7:6 of the command's Instruction field set to "01", and those bits are therefore left out of the table. Bits 1:0 of the command's Instruction field determines the length of the command's Reply Address Field, and does not affect the action taken, so they have been left out of the table as well. The action taken assumes that no errors were detected in the RMAP packet. For handling of RMAP packet error, see section 82.6.1.4.

*Table 1476.*RMAP command decoding and handling.

Bit 5	Bit 4	Bit 3	Bit 2		
Write / Read	Verify Data Before Write	Reply	Increment Addr	Function	Action taken
0	0	0	0	Invalid	No operation performed. Error code 0x02 is saved in the RTR.PSTSCFG.EC field. No reply is sent.
0	0	0	1	Invalid	No operation performed. Error code 0x02 is saved in the RTR.PSTSCFG.EC field. No reply is sent.
0	0	1	0	Read single address	Read operation performed, if the requirements in section 82.6.1.1 are met.
0	0	1	1	Read incrementing address	Read operation performed, if the requirements in section 82.6.1.1 are met.
0	1	0	0	Invalid	No operation performed. Error code 0x02 is saved in the RTR.PSTSCFG.EC field. No reply is sent.
0	1	0	1	Invalid	No operation performed. Error code 0x02 is saved in the RTR.PSTSCFG.EC field. No reply is sent.
0	1	1	0	Invalid	No operation performed. Reply is sent with error code 0x02. Error code is also saved in the RTR.PSTSCFG.EC field.
0	1	1	1	Read-modify-write incrementing address	Read-modify-write operation performed if the requirements in section 82.6.1.1 are met.
1	0	0	0	Write, single address, don't verify before writing, no reply	No operation performed. Error code 0x0A is saved in the RTR.PSTSCFG.EC field. No reply sent.
1	0	0	1	Write, incrementing address, don't verify before writing, no reply	No operation performed. Error code 0x0A is saved in the RTR.PSTSCFG.EC field. No reply sent.
1	0	1	0	Write, single address, don't verify before write, send reply	No operation performed. Reply is sent with error code 0x0A. Error code is also saved in the RTR.PSTSCFG.EC field.
1	0	1	1	Write, incrementing address, don't verify before write, send reply	No operation performed. Reply is sent with error code 0x0A. Error code is also saved in the RTR.PSTSCFG.EC field.
1	1	0	0	Write, single address, verify before writing, no reply	Write operation performed if the requirements in section 82.6.1.1 are met.

*Table 1476.*RMAP command decoding and handling.

Bit 5	Bit 4	Bit 3	Bit 2		
Write / Read	Verify Data Before Write	Reply	Incre-ment Addr	Function	Action taken
1	1	0	1	Write, incre-menting address, verify before writing, no reply	Write operation performed if the requirements in section 82.6.1.1 are met.
1	1	1	0	Write, single address, verify before writing, send reply	Write operation performed if the requirements in section 82.6.1.1 are met.
1	1	1	1	Write, incre-menting address, verify before writing, send reply	Write operation performed if the requirements in section 82.6.1.1 are met.

82.6.1.3 Access control

After reset / power-up the configuration port's address space can be accessed from all the ports. Configuration port accesses can be individually disabled per port by clearing the corresponding RTR.PCTRL.CE bit. Write commands, and read-modify-write commands to the configuration area can be globally disabled by writing a 0 to the RTR.CFGWE.WE bit.

There is also a CFGLOCK pin which can be used to disable configuration accesses from all ports except port 1 and 2. This signal overrides the setting of the RTR.PCTRL.CE for all ports, enabling configuration port accesses for port 1 and 2, and disabling them for all other ports. The RTR.CFGWE register still affects ports 1 and 2 in this case.

When a correct RMAP command is received but not allowed due to one or more of the access control features being enabled, a reply with Status field set to 0x0A (Authorization failure) is sent (if requested), and the RTR.PSTSCFG.EC field is updated to reflect the error. If a reply is not requested, the RTR.PSTSCFG.EC field is still set. In both cases, the operation is not performed.

82.6.1.4 RMAP Error handling

Table 1477 shows the order in which errors in an RMAP command are detected. As soon as an error is detected, the command is discarded. If a reply should be sent, to a command that included an error, the reply is sent as soon as possible after the error is detected. This means that the reply might be sent out before the complete incoming RMAP command has been received. Note that since the complete RMAP command is buffered before it is executed, a command that contains an error is never executed.

*Table 1477.*RMAP target error detection order

Detection Order	**Error type**	**RMAP error code**	**Action taken**
1	Wrong Protocol Identifier	N/A	The RTR.PSTSCFG.PT bit is set in order to indicate that the error occurred. No reply is sent.
2	EOP / EEP before com-pleted header	N/A	The RTR.PSTSCFG.EO / RTR.PSTSCFG.EE bit is set in order to indi-cate that the error occurred. No reply is sent.

*Table 1477.*RMAP target error detection order

Detection Order	Error type	RMAP error code	Action taken
3	Header CRC error	N/A	The RTR.PSTSCFG.HC bit is set in order to indicate that the error occurred. No reply is sent.
4	Unused RMAP packet type	N/A	If the packet type (bit 7:6 of the packet's Instruction field) is "10" or "11" then the bit RTR.PSTSCFG.PT is set. For the value "00" (indicating a reply), no bit in RTR.PSTSCFG is set, since the RMAP standard [RMAP] does not specify that such an event should be recorded.
5	EEP immediately after header	N/A	The RTR.PSTSCFG.EE bit is set in order to indicate that the error occurred. No reply is sent.
6	Unused RMAP command code	0x02	RMAP error code is saved in the RTR.PSTSCFG.EC field. Reply is sent if the Reply bit in the command's Instruction field was set to 1.
7	Invalid Target Logical Address	0x0C	RMAP error code is saved in the RTR.PSTSCFG.EC field. Reply is sent if the Reply bit in the command's Instruction field was set to 1.
8	Invalid Key	0x03	RMAP error code is saved in the RTR.PSTSCFG.EC field. Reply is sent if the Reply bit in the command's Instruction field was set to 1.
9	Verify buffer overrun	0x09	RMAP error code is saved in the RTR.PSTSCFG.EC field. Reply is sent if the Reply bit in the command's Instruction field was set to 1.
10	RMW data length error	0x0B	RMAP error code is saved in the RTR.PSTSCFG.EC field. Reply is sent if the Reply bit in the command's Instruction field was set to 1.
11	RMAP command not implemented or not authorized.	0x0A	RMAP error code is saved in the RTR.PSTSCFG.EC field. Reply is sent if the Reply bit in the command's Instruction field was set to 1.
12	Early EOP / early EEP (not immediately after header)	0x05 / 0x07	RMAP error code is saved in the RTR.PSTSCFG.EC field. Reply is sent if the Reply bit in the command's Instruction field was set to 1.
13	Invalid Data CRC	0x04	RMAP error code is saved in the RTR.PSTSCFG.EC field. Reply is sent if the Reply bit in the command's Instruction field was set to 1.
14	EEP	0x07	RMAP error code is saved in the RTR.PSTSCFG.EC field. Reply is sent if the Reply bit in the command's Instruction field was set to 1.
15	Too much data	0x06	RMAP error code is saved in the RTR.PSTSCFG.EC field. Reply is sent if the Reply bit in the command's Instruction field was set to 1.

Most of the errors listed in table 1477 are errors that only occur in one specific way, and they are also explained in the RMAP standard [RMAP]. Authorization failure (error code 0x0A) is however an exception. All the cases that lead to an authorization failure are listed below:

- A read command's Data Length field exceed 128 B.
- A command's (read, write, or read-modify-write) Address field does not contain a 4 B aligned address.
- The access control features described in section 82.6.1.3 prevented the port from accessing the RMAP target.
- The Address field of a command (read, write, or read-modify-write) contains an address that is outside of the configuration port's memory space.
- The Address field of the command (read, write, or read-modify-write) combined with the Length field would generate an access outside of the configuration port's memory space.
- The Length field of a command (read, write, or rea-modify write) is not a multiple of 4.
- A non-verified write command was received.

82.6.2 AMBA AHB slave interface

The configuration port provides an AMBA AHB slave interface, which makes the entire address space of the configuration port accessible from the AHB bus. The address offsets are the same as when accessing through RMAP but the base address is different. The slave consists of an AHB I/O bank: the 32-bit AHB base address is calculated by setting bits 31 downto 20 to 0xFFF and bits 19 downto 8 to the VHDL generics cfghaddr and cfghmask anded. So if, for example, cfghaddr is 0xC00 and cfghmask is 0xFFF, the router AHB memory area will occupy 256 bytes starting from 0xFFFC0000.

The routing table is shared between the ports, RMAP target and AHB slave, so accesses from the AHB slave might be stalled because of accesses from the other sources. The priority order when accessing the routing table, starting from the highest, is: router ports, AHB slave, RMAP target. Note that since the AHB slave has higher priority than the RMAP target, it is possible to read and write to the configuration port's registers in the middle of an RMAP write command. This needs to be considered in order to avoid a mismatch between the expected written value and actual written value.

None of the access control mechanisms mentioned in section 82.6.1.3 have any effect on the AHB slave interface.

82.7 Registers

The configuration port's registers listed in this section can be accessed either through the RMAP target, or the AMBA AHB slave interface. The RMAP addresses specified in table 1478. The AHB addresses are determined by adding the RAMP addresses table 1478 to the AHB base address determined by the cfghaddr and cfghmask VHDL generics. Registers that exist in several identical copies, corresponding to different addresses or ports, for example the RTR.RTPMAP registers, are only described once.

Only 32-bit single-accesses to the registers through AHB are supported.

Table 1478. GRSPWROUTER registers

RMAP address	Register name	Acronym
0x00000000	RESERVED *	
0x00000004 - 0x0000007C	Routing table port mapping, physical addresses 1-31 *	RTR.RTPMAP
0x00000080 - 0x000003FC	Routing table port mapping, logical addresses 32-255	RTR.RTPMAP
0x00000400	RESERVED *	
0x00000404 - 0x0000047C	Routing table address control, physical addresses 1-31 *	RTR.RTACTRL
0x00000480 - 0x000007FC	Routing table address control, logical addresses 32-255	RTR.RTACTRL
0x00000800	Port control, port 0 (configuration port)	RTR.PCTRLCFG
0x00000804 - 0x0000087C	Port control, port 1-31	RTR.PCTRL
0x00000880	Port status, port 0 (configuration port)	RTR.PSTSCFG
0x00000884 - 0x000008FC	Port status, ports 1-31	RTR.PSTS
0x00000900 - 0x0000097C	Port timer reload, ports 0-31	RTR.PTIMER
0x00000980	Port control 2, ports 0 (configuration port)	RTR.PCTRL2CFG
0x00000984 - 0x000009FC	Port control 2, ports 1-31	RTR.PCTRL2
0x00000A00	Router configuration / status	RTR.RTRCFG
0x00000A04	Time-code	RTR.TC
0x00000A08	Version / instance ID	RTR.VER
0x00000A0C	Initialization divisor	RTR.IDIV
0x00000A10	Configuration write enable	RTR.CFGWE
0x00000A14	Timer prescaler reload	RTR.PRESCALER
0x00000A18	Interrupt mask	RTR.IMASK
0x00000A1C	Interrupt port mask	RTR.IPMASK
0x00000A20	Port interrupt pending	RTR.PIP
0x00000A24	Interrupt code generation	RTR.ICODEGEN
0x00000A28	Interrupt code distribution ISR, interrupt 0-31	RTR.ISR0
0x00000A2C	Interrupt code distribution ISR, interrupt 32-63	RTR.ISR1
0x00000A30	Interrupt code distribution ISR timer reload	RTR.ISRTIMER
0x00000A34	Interrupt code distribution ACK-to-INT timer reload	RTR.AITIMER
0x00000A38	Interrupt code distribution ISR change timer reload	RTR.ISRCTIMER
0x00000A3C	RESERVED	

Table 1478. GRSPWROUTER registers

RMAP address	Register name	Acronym
0x00000A40	SpaceWire link running status	RTR.LRUNSTAT
0x00000A44	Capability	RTR.CAP
0x00000A48 - 0x00000A4C	RESERVED	
0x00000A50	SpaceWire Plug-and-Play - Device Vendor and Product ID	RTR.PNPVEND
0x00000A54	SpaceWire Plug-and-Play - Unit Vendor and Product ID	RTR.PNPUVEND
0x00000A58	SpaceWire Plug-and-Play - Unit Serial Number	RTR.PNPUSN
0x00000A5C	SpaceWire Plug-and-Play - Port network discovery enable	RTR.PNPNET-DISC
0x00000A60 - 0x00000C0C	RESERVED	
0x00000C10, 0x00000C20 ... 0x00000DF0	Outgoing character counter, ports [1, 2 ... 31] ***	RTR.OCHARCNT
0x00000C14, 0x00000C24 ... 0x00000DF4	Incoming character counter, ports [1, 2 ... 31] ***	RTR.ICHARCNT
0x00000C18, 0x00000C28 ... 0x00000DF8	Outgoing packet counter, ports [1, 2 ... 31] ***	RTR.OPKTCNT
0x00000C1C, 0x00000C2C ... 0x00000DFC	Incoming packet counter, ports [1, 2 ... 31] ***	RTR.IPKTCNT
0x00000E00 - 0x00000E7C	Maximum packet length, ports 0-31	RTR.MAXPLEN
0x00000E84 - 0x00000EFC	Credit counter, ports 1-31	RTR.CREDCNT
0x00000F00	General purpose out, bits 0-31	RTR.GPO0
0x00000F04	General purpose out, bits 32-63	RTR.GPO1
0x00000F08	General purpose out, bits 64-95	RTR.GPO2
0x00000F0C	General purpose out, bits 96-127	RTR.GPO3

Table 1478. GRSPWROUTER registers

RMAP address	Register name	Acronym
0x00000F10	General purpose in, bits 0-31	RTR.GPI0
0x00000F14	General purpose in, bits 32-63	RTR.GPI1
0x00000F18	General purpose in, bits 64-95	RTR.GPI2
0x00000F1C	General purpose in, bits 96-127	RTR.GPI3
0x00001000	RESERVED**	
0x00001004 - 0x0000107C	Routing table, combined port mapping and address control, addresses 1-31 ****	RTR.RTCOMB
0x00001080 - 0x000013FC	Routing table, combined port mapping and address control, addresses 32-255 ****	RTR.RTCOMB
0x00001400 - 0x00001FFC	RESERVED	
0x00002000 - 0x00002FFC	APB address area	RTR.APBAREA

* Physical address 0 (configuration port), and non existing ports does not have an RTR.RTPMAP or RTR.RTACTRL register, and are therefore RESERVED.

** Physical address 0 (configuration port), and non existing ports does not have an RTR.RTCOMB register, and are therefore RESERVED.

*** Physical address 0 (configuration port), and non existing ports does not have an RTR.OCHARCNT, RTR.ICHARCNT, RTR.IPKTCNT or RTR.OPKTCNT register, and are therefore RESERVED.

**** RTR.RTCOMB is only implemented if the total number of ports is less than 28.

Table 1479. 0x00000004-0x0000007C, 0x00000080-0x000003FC - RTR.RTPMAP - Routing table port mapping, addresses 1-31 and 32-255

31	1	0
PE		PD
*		0
rw*		rw

31: 1 Port enable bits (PE) - When set to 1, each bit enables packets with the physical / logical address corresponding to this RTR.RTPMAP register to be sent on the port with the same number as the bit index. For physical addresses, the bit index corresponding to the port with the same number as the physical address itself is always 1 (not possible to write to 0). For logical addresses, at least one bit must be set to 1 in order for a packet with the corresponding address to be routed; otherwise the packet is spilled and an invalid address error generated. Reset value for physical addresses is all zeroes (except for the bit index corresponding to the port with the same number as the address). Reset value for logical addresses is zero. Only bits corresponding to existing ports, as defined by the sum of the VHDL generics *spwports*, *ambaports*, *fifoports* and *customport*, are implemented.

0 Packet distribution (PD) - When set to 1, packet distribution is used for the physical / logical address corresponding to this RTR.RTPMAP register. When set to 0, group adaptive routing is used. See section 82.2.5 and 82.2.6 for more information.

Table 1480. 0x00000404-0x0000047C, 0x00000480-0x000007FC - RTR.RTACTRL - Routing table address control, addresses 1-31 and 32-255

31	4	3	2	1	0
RESERVED		SR	EN	PR	HD
0x0000000		*	*	*	*
r		rw	rw	rw	rw

31: 4 RESERVED

3 Spill-if-not-ready (SR) - When set to 1, an incoming packet with the corresponding physical / logical address is immediately spilled if the selected output port's link interface is not in run-state. If packet distribution is used for the incoming packet, and this bit is set, the packet is spilled unless all output ports' link interfaces are in run state. For physical addresses, this bit is double mapped in the RTR.PCTRL.SR field. Reset value for physical addresses are taken from the SPILLIFNOTREADY pin. Reset value for logical addresses is N/R.

2 Enable (EN) - Enables the routing table address control entry. Address control entries for physical addresses are always enabled, and this field is constant 1. For logical addresses, this bit must be set to 1 in order for packets with the corresponding logical address to be routed. Reset value for logical addresses is 0.

1 Priority (PR) - Sets the arbitration priority of this physical / logical address. 0 = low priority, 1 = high priority. Used when more than one packet is competing for the same output port. For physical addresses, this bit is double mapped in the RTR.PCTRL.PR field. Reset value for physical addresses is 0. Reset value for logical addresses is N/R.

0 Header deletion (HD) - Enables / disabled header deletion for the corresponding logical address. For physical addresses, header deletion is always enabled, and this bit is constant 1. Reset value for logical addresses is N/R.

Table 1481. 0x00000800 - RTR.PCTRLCFG - Port control, port 0 (configuration port)

31 18	17	16	15 10	9	8 0
RESERVED	PL	TS	RESERVED	TR	RESERVED
0x0000	0	0	0x00	*	0x000
r	rw	rw	r	rw	r

31: 18 RESERVED

17 Packet length truncation (PL) - When set to 1, an RMAP / SpaceWire Plug-and-Play reply is spilled, and an EEP written to the transmit FIFO of the output port, if the total length of the reply packet exceeds the maximum length specified in the RTR.MAXPLEN register for port 0. See section 82.2.15 for more information on packet length truncation.

16 Time-code / distributed interrupt code truncation (TS) - When set to 1, an RMAP / SpaceWire Plug-and-Play reply is spilled, and an EEP written to the transmit FIFO of the output port, if a valid time-code / distributed interrupt code is received and if the code matches the codes selected by the RTR.PCTRL2CFG.SV and RTR.PCTRL2CFG.SM fields. See section 82.2.19 for more information.

15: 10 RESERVED

9 Timer enable (TR) - Enable data character timer for port 0. See section 82.2.14 for details. Reset value set from TIMEEN signal.

8: 0 RESERVED

Table 1482. 0x00000804-0x0000084C - RTR.PCTRL - Port control, ports > 0

31 30 29 24	23 22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RD	RES	ST	SR	AD	LR	PL	TS	IC	ET	NF	PS	BE	DI	TR	PR	TF	RS	TE	R	CE	AS	LS	LD
0x0 *	0x0	0	*	*	*	0	0	*	0	0	0	*	*	*	0	0	0	1	0	*	1	0	0
rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	rw	rw	rw	rw

31: 24 Run-state clock divisor (RD) - Clock divisor value used for the corresponding port's link interface when in run-state. Field is only available for the SpaceWire ports. Bits 31:30 have reset value 0x0, while bits 29:24 get their reset value from the IDIVISOR[7:0] pins. For more information about setting the link-rate for SpaceWire ports during run-state see section 82.2.22.

23: 22 RESERVED

21 Static routing enable (ST) - When set to 1, incoming packets on this port are routed based on the physical address specified in the corresponding RTR.PCTRL2.SD field, and the setting of the corresponding RTR.PCTRL2.SC bit, instead of the packet's first byte. Header deletion is not used when static routing is enabled, which means that the first byte of the packet is always sent as well. This bit can only be set to 1 if the RTR.RTRCFG.SR bit is set to 1. Note that when this bit is set to 1 it is not possible to access the configuration port from this port.

20 Spill-if-not-ready (SR) - This bit is double mapping of the RTR.RTACTRL.SR bit. See table 1480.

19 Auto-disconnect (AD) - When set to 1, the auto-disconnect feature described in section 82.2.13 is enabled. Reset value taken from the AUTODCONNECT pin. This bit is only available for the SpaceWire ports.

18 Link-start-on-request (LR) - When set to 1, the link-start-on-request feature described in section 82.2.12 is enabled. Reset value taken from the LINKSTARTONREQ pin. This bit is only available for the SpaceWire ports.

17 Packet length truncation (PL) - When set to 1, packets for which this port is the input port will be spilled, and an EEP written to the transmit FIFO of the output port(s), if the packets exceed the maximum length specified in the corresponding RTR.MAXPLEN register. See section 82.2.15 for more information on packet length truncation.

16 Time-code / distributed interrupt code truncation (TS) - When set to 1, packets for which this port is the input port will be spilled, and an EEP written to the transmit FIFO of the output port(s), if a valid time-code / distributed interrupt code is received, and if the code also matches the codes selected by the RTR.PCTRL2.SV and RTR.PCTRL2.SM fields. See section 82.2.19 for more information.

Table 1482. 0x00000804-0x0000084C - RTR.PCTRL - Port control, ports > 0

15	Distributed interrupt code enable (IC) - When set to 0, all incoming distributed interrupt codes on this port are discarded, and no distributed interrupt codes are sent out on the port. When set to 1, the four bits RTR.PCTRL2.IR, RTR.PCTRL2.IT, RTR.PCTRL2.AR, RTR.PCTRL2.AT are used to enable / disable distributed interrupt code transmit and receive. Note that the global distributed interrupt code enable bit, RTRTCFG.IE, also must be set to 1 for distributed interrupt codes to be sent / received. See section 82.2.17 for a description of distributed interrupts. Reset value set from INTERRUPTFWD signal.
14	Enable external time (ET) - When a time-code is received on the port and this bit is set to 0, the router discards the received time-code value and instead increments its internal time-counter value (RTR.TC.TC), and forwards a time-code with the new value to the other ports. If this bit is set to 1 when the time-code is received, the time-code is processed according to the rules described in section 82.2.16. This bit is only available for AMBA and FIFO ports.
13	No port force (NF) - Disables forced port selection, if the SpaceWire ports are implemented with redundant ports through the VHDL generic *dualport*, using the PS bit and lets the core automatically select the active link. This bit is only available for SpaceWire ports.
12	Port select (PS) - Select between nominal and redundant SpaceWire port if enabled through the VHDL generic *dualport*. This bit is only available for SpaceWire ports.
11	Bridge enable (BE) - Enable bridge mode for FIFO ports. See section 82.4.5 for more information. This bit is only available for FIFO ports. Reset value set from ENBRIDGE signal.
10	Disable port (DI) - When set to 1, data transfers to and from this port are disabled. See section 82.2.7 for details. NOTE: If this bit is set in combination with the RTR.PCTRL.LD bit, the link interface for this port will be clock gated, and the LVDS drivers for this port will be powered-down.
9	Packet timer enable (TR) - Enable the data character timer for incoming packets. See section 82.2.14 for details. Reset value set from TIMEREN signal.
8	Priority (PR) - This bit is double mapping of the RTR.RTACTRL.PR bit. See table 1480.
7	Transmit FIFO reset (TF) - Resets the transmit FIFO on this port. This means that the FIFO is emptied (counters and pointers set to 0), and an EEP is written to the FIFO to ensure that any incomplete packet is detected by the receiver. If a packet transmission is active (another port is using this port as output port) when this bit is set, the remainder of that packet will be spilled before the EEP is inserted. This bit is self-clearing, and should not be written with 0 while it is 1, since that could abort the ongoing transmit FIFO reset.
6	Receive FIFO spill (RS) - Spills the receive FIFO for this port, meaning that the packet currently being received is spilled. The output port(s) used for the packet will have an EEP written to the transmit FIFO to indicate that the packet was ended prematurely. If no packet is received, setting this bit has no effect. This bit is self-clearing, and should not be written with 0 while it is 1, since that could abort the ongoing receive FIFO spill.
5	Time-code enable (TE) - Enables time-codes to be received and transmitted on this port. When set to 1, received time-codes are processed according to the rules described in section 82.2.16. If this bit is set to 0, all received time-codes on this port are ignored.
4	RESERVED
3	Configuration port access enable (CE) - Enable accesses to the configuration port from this port. If set to 0, incoming packets with physical address 0 will be spilled.
2	Autostart (AS) - Enable the link interface FSM's autostart feature, as defined in ECSS-E-ST-50-12C [SPW]. This bit is only available for the SpaceWire ports.
1	Link start (LS) - Start the link interface FSM. This bit is only available for the SpaceWire ports.
0	Link disabled (LD) - Disable the link interface FSM. This bit is only available for the SpaceWire ports. NOTE: When this bit is set to 1, the LVDS driver for this SpaceWire port will be powered down. If this bit is set in combination with the RTR.PCTR.DI bit, the link interface will be clock gated.

Table 1483. 0x00000880 - RTR.PSTSCFG - Port status, port 0 (configuration port)

31	30	29	28	27	26	25	24	23 20	19	18	17	16 12	11 7	6 5	4	3 0
EO	EE	PL	TT	PT	HC	PI	CE	EC	R	TS	ME	RESERVED	IP	RES	CP	PC
0	0	0	0	0	0	0	0	0x0	0	0	0	0x00	0x0	0x0	0	0x0
wc	wc	wc	wc	wc	wc	wc	rw*	r	r	wc	wc	r	r	r	rw*	r

31 Early EOP (EO) - Set to 1 when an RMAP / SpaceWire Plug-and-Play command with an early EOP was received by the configuration port. See section 82.6.1.4 for error detection order.

30 Early EEP (EE) - Set to one when an RMAP / SpaceWire Plug-and-Play command with an early EEP was received by the configuration port. See section 82.6.1.4 for error detection order.

29 Packet length truncation (PL) - Set to 1 when an RMAP / SpaceWire Plug-and-Play reply packet has been spilled due to a maximum length violation. See section 82.2.15 for details.

28 Time-code / distributed interrupt code tick truncation (TT) - Set to one when an RMAP / SpaceWire Plug-and-Play reply packet has been spilled due to a time-code / distributed interrupt code. See section 82.2.19 for details.

27 Packet type error (PT) - Set to one if an RMAP / SpaceWire Plug-and-Play packet with correct header CRC, but with the packet type bits set to the reserved values “10” or “11”, was received by the configuration port. See section 82.6.1.4 for error detection order.

26 Header CRC Error (HC) - Set to one if a Header CRC error is detected in an RMAP / SpaceWire Plug-and-Play command received by the configuration port. See section 82.6.1.4 for error detection order.

25 Protocol ID Error (PI) - Set to one if a packet received by the configuration port had the wrong protocol ID. Supported protocol ID:s are 0x01 (RMAP), and 0x03 (SpaceWire Plug-and-Play). See section 82.6.1.4 for error detection order.

24 Clear error code (CE) - Write with a 1 to clear the RTR.PSTSCFG.EC field. This bit is self clearing and always reads 0. Writing 0 has no effect.

23: 20 Error code (EC) - Shows the four least significant bits of the latest non-zero RMAP status code. If zero, no error has occurred.

19 RESERVED

18 Timeout spill (TS) - Set to one when an RMAP reply was spilled due to a packet timeout. See section 82.2.14 for details.

17 Uncorrectable port buffer error (ME) - This bit is set for detected errors in port RX and TX buffers.

16: 12 RESERVED

11: 7 Input port (IP) - The number of the last port from which a packet was routed to the configuration port. This field is updated even if an operation is not performed, for example due to an incorrect RMAP packet.

6: 5 RESERVED

4 Clear SpaceWire Plug-and-Play error code (CP) - Write with a 1 to clear the RTR.PSTSCFG.PC field. This bit is self clearing and always reads 0. Writing 0 has no effect.

3: 0 SpaceWire Plug-and-Play Error code (PC) - Shows the four least significant bits of the latest non-zero SpaceWire Plug-and-Play status code. If zero, no error has occurred.

Table 1484. 0x00000884-0x000008CC - RTR.PSTS - Port status, ports > 0

31	30	29	28	27	26	25 23	22	21	20	19	18	17	16	15	14 12	11 7	6	5	4	3	2	1	0
PT		PL	TT	RS	SR	RESERVED	LR	SP	AC	AP	TS	ME	TF	RE	LS	IP	PR	PB	IA	CE	ER	DE	PE
*		0	0	0	0	0x0	0	0	0	0	0	0	0	1	000	00000	0	0	0	0	0	0	0
r		wc	wc	wc	wc	r	r	r	r	r	wc	wc	r	r	r	r	r	r	wc	wc	wc	wc	wc

31: 30 Port type (PT) - The type of this port. Constant value of “00” for the SpaceWire ports, “01” for AMBA ports, “10” for FIFO ports, and “11” for custom ports.

29 Packet length truncation (PL) - Set to 1 when a packet for which this port was the input port has been spilled due to the packet length truncation feature. See section 82.2.15 for details.

28 Time-code / distributed interrupt code tick truncation (TT) - Set to 1 when a packet for which this port was the input port has been spilled due to the time-code / distributed interrupt code truncation feature. See section 82.2.19 for details.

Table 1484. 0x00000884-0x000008CC - RTR.PSTS - Port status, ports > 0

27	RMAP / SpaceWire Plug-and-Play spill (RS) - Set to 1 when an RMAP / SpaceWire Plug-and-Play command received on this port was spilled by the configuration port.
26	Spill-if-not-ready spill (SR) - Set to 1 when a packet received on this port was spilled due to the spill-if-not-ready feature. See section 82.2.9.
25: 23	RESERVED
22	Link-start-on-request status (LR) - Set to 1 when this port either was started, or currently is trying to start, due to the link-start-on-request feature, described in section 82.2.12. This bit is only available for the SpaceWire ports.
21	Spill status (SP) - This bit is 1 when a packet that is incoming on this port currently is being spilled. Otherwise, this bit is 0.
20	Active status (AC) - Set to 1 when a packet arrives at this port and the port has been given access to the routing table. Cleared when the packet has been transmitted or spilled.
19	Active port (AP) - Indicates the active SpaceWire ports if dual ports is implemented. This bit is only available for SpaceWire ports if *dualport* is set.
18	Timeout spill (TS) - Set to 1 when a packet for which this port was the input port was spilled due to a packet timeout. See section 82.2.14 for details.
17	Uncorrectable port buffer error (ME) - This bit is set for detected errors in port RX and TX buffers.
16	Transmit FIFO full (TF) - Set to 1 when the transmit FIFO on this port is full.
15	Receive FIFO empty (RE) - Set to 1 when the receive FIFO on this port is empty.
14: 12	Link state (LS) - Current link state. 000 = Error reset. 001 = Error wait, 010 = Ready, 011 = Started, 100 = Connecting, 101 = Run state. This field is only available for the SpaceWire ports.
11: 7	Input port (IP) - This field shows the number of the input port for either the currently ongoing packet transfer on this port (if RTR.PSTS.PB = 1), or for the last packet transfer on this port (if RTR.PSTS.PB = 0).
6	Port receive busy (PR) - Set to 1 when this port is the input port of an ongoing packet transfer.
5	Port transmit busy (PB) - Set to 1 when this port is the output port of an ongoing packet transfer.
4	Invalid address (IA) - Set to 1 when an invalid address error occurred on this port. See section 82.2.11 for details.
3	Credit error (CE) - Set to 1 when a credit error has occurred. This bit is only available for the SpaceWire ports.
2	Escape error (ER) - Set to 1 when an escape error has occurred. This bit is only available for the SpaceWire ports.
1	Disconnect error (DE) - Set to 1 when a disconnect error has occurred. This bit is only available for the SpaceWire ports.
0	Parity error (PE) - Set to 1 when a parity error has occurred on. This bit is only available for the SpaceWire ports.

Table 1485. 0x00000900-0x0000094C - RTR.PTIMER - Port timer reload

31	0
RL	
*	
rw*	

9: 0	Timer reload (RL) - Port timer reload value, counted in prescaler ticks. This value is used to reload the corresponding port timer used for packet transfer timeouts, and auto-disconnect. The minimum value of this field is 1. Trying to write 0 will result in 1 being written. Reset value set from RELOADN[31:0] signal. The width of this field is determined by the VHDL generic *porttimerbits*.

Table 1486. 0x00000980 - RTR.PCTRL2CFG - Port control 2, port 0 (configuration port)

31 24	23 16	15	14 13 12 11 10 9 8 6 5 1	0
SM	SV	OR	RESERVED	
0xC0	0x00	1	0x0000	
rw	rw	rw	r	

31: 24 Time-code / distributed interrupt code truncation mask (SM) - Defines which bits of a time-code / distributed interrupt code that must match the value specified in RTR.PCTRL2CFG.SV in order for an RMAP / SpaceWire Plug-and-Play reply packet to be spilled. If a bit in this field is set to 1, the corresponding bit in RTR.PCTRL2.SV must match the time-code / distributed interrupt code. If a bit in this field is set to 0, the corresponding bit in RTR.PCTRL2.SV does not have to match the time-code / distributed interrupt code.

23: 16 Time-code / distributed interrupt code truncation value (SV) - Defines the value to use together with the RTR.PCTRL2CFG.SM field when checking if a received time-code / distributed interrupt code should spill an ongoing RMAP / SpaceWire Plug-and-Play reply.

15 Overrun timeout enable (OR) - Enables spilling due to overrun timeouts for RMAP / SpaceWire Plug-and-Play replies. See section 82.2.14 for details.

14: 0 RESERVED

Table 1487. 0x00000984-0x000009CC - RTR.PCTRL2 - Port control 2, ports > 0

31 24	23 16	15	14	13	12	11	10	9	8 6	5 1	0
SM	SV	OR	UR	R	AT	AR	IT	IR	RESERVED	SD	SC
0xC0	0x00	1	1	0	1	1	1	1	0x0	0x00	0
rw	rw	rw	rw	r	rw	rw	rw	rw	r	rw	rw

31: 24 Time-code / distributed interrupt code truncation mask (SM) - Defines which bits of a time-code / distributed interrupt code that must match the value specified in RTR.PCTRL2.SV in order for a packet, for which this port is the input port, to be spilled. If a bit in this field is set to 1, the corresponding bit in RTR.PCTRL2.SV must match the time-code / distributed interrupt code. If a bit in this field is set to 0, the corresponding bit in RTR.PCTRL2.SV does not have to match the time-code / distributed interrupt code.

23: 16 Time-code / distributed interrupt code truncation value (SV) - Defines the value to use together with the RTR.PCTRL2.SM field when checking if a time-code / distributed interrupt code should spill a packet for which this port is the input port.

15 Overrun timeout enable (OR) - Enables spilling due to overrun timeouts for packets for which this port is the input port. See section 82.2.14 for details.

14 Underrun timeout enable (UR) - Enables spilling due to unerrun timeouts for packets for which this port is the input port. See section 82.2.14 for details.

13 RESERVED

12 Interrupt acknowledgment code / extended interrupt code transmit enable (AT) - Enables the transmission of interrupt acknowledgment codes / extended interrupt codes on this port. If set to 0, no interrupt acknowledgment codes / extended interrupt codes will be forwarded to this port.

11 Interrupt acknowledgment code / extended interrupt code receive enable (AR) - Enabled the reception of interrupt acknowledgment codes / extended interrupt codes on this port. If set to 0, all received interrupt acknowledgment codes / extended interrupt codes on this port will be silently discarded.

10 Interrupt code transmit enable (IT) - Enables the transmission of interrupt codes on this port. If set to 0, no interrupt codes will be forwarded to this port.

9 Interrupt code receive enable (IR) - Enabled the reception of interrupt codes on this port. If set to 0, all received interrupt codes on this port will be silently discarded.

8: 6 RESERVED

5: 1 Static route destination (SD) - When RTR.PCTRL.ST is set to 1, incoming packets on this port will be routed based on the value of this field, and the setting of RTR.PCTRL2.SC, instead of the packet's first byte.

0 Static route configuration (SC) - When this bit is set to 1, the RTR.RTPMAP register corresponding to the physical address specified by the RTR.PCTRL2.SD field will be used when routing packets, if RTR.PCTRL.ST is set to 1.

Table 1488. 0x00000A00 - RTR.RTRCFG - Router configuration / status

31 27	26 22	21 17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SP	AP	FP	R	SR	PE	IC	IS	IP	AI	AT	IE	RE	EE	R	SA	TF	RM	TA	PP
*	*	*	0	*	*	0	0	0	*	1	*	0	*	0	1	*	0	1	1
r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw	r	rw	rw	wc	r	r

31: 27 SpaceWire ports (SP) - Set to the number of SpaceWire ports in the router.

26: 22 AMBA ports (AP) - Set to the number of AMBA ports in the router.

21: 17 FIFO ports (FP) - Set to the number of FIFO ports in the router.

16 RESERVED

15 Static routing enable (SR) - This read-only bit specifies if the router's static routing feature is enabled (1) or disabled (0). See section 82.2.8 for details. The value is set from the STATICROUTEEN signal at reset.

14 SpaceWire Plug-and-Play enable (PE) - This read-only bit specifies if the router's SpaceWire Plug-and-Play features are enabled (1) or disabled (0). See section 82.7.1 for details. The value is set from the PNPEN pin at reset.

13 ISR change timer enable (IC) - If set to 1, the router will wait for the time period specified by the RTR.IRCTIMER register after an ISR bit change value, before it allows an incoming distributed interrupt code to change the value of the same ISR bit. If set to 0, the ISR change timers are not used, and an ISR bit is allowed to change value again as soon as the previous distributed interrupt code has been distributed.

12 Distributed interrupt code selection routine (IS) - If set to 0, the router uses round-robin on the interrupt numbers when deciding which distributed interrupt code to distribute next. If set to 1, the router gives priority to lower interrupt numbers when deciding which distributed interrupt code to distribute. See section 82.2.17.1.

11 Distributed interrupt code priority (IP) - When set to 0, all interrupt codes have priority over all interrupt acknowledgment codes / extended interrupt codes, and will be distributed first. When set to 1, all interrupt acknowledgment codes have priority over all interrupt codes. See section 82.2.17.1.

10 Auxiliary distributed interrupt codes enable (AI) - If set to 1, distributed interrupt codes can be sent and received on the auxiliary time-code / distributed interrupt code interface. If set to 0, all distributed interrupt codes received on the auxiliary interface are silently discarded, and no distributed interrupt codes will be transmitted on the interface. Reset value set from INTERRUPTFWD signal.

9 Auxiliary time-code enable (AT) - If set to 1, time-codes can be sent and received on the auxiliary time-code / distributed interrupt code interface. If set to 0, all time-codes received on the auxiliary interface are silently discarded, and no time-codes will be transmitted on the interface.

8 Distributed interrupt codes enable (IE) - Global enable/disable for distributed interrupt codes. If set to 0, all received distributed interrupt codes will either be silently discarded (if RTRCFG.TF = 1), or handled as time-codes (if RTRCFG.TF = 0). When set to 1, whether or not distributed interrupt codes are received or transmitted on a port depends on the setting of the register bits RTR.PCTRL.IC, RTR.PCTRL2.IR, RTR.PCTRL2.IT, RTR.PCTRL2.AR, and RTR.PCTRL2.AT. Reset value taken from INTERRUPTCODEEN signal.

7 Reset (RE) - Resets the complete router when written with a 1. When this bit is written through RMAP, an RMAP reply will not be sent, even if the reply bit in the RMAP commands Instruction field is set to 1. This bit is self-clearing.

6 Enable extended distributed interrupts (EE) - If set to 0, all distributed interrupt codes with bit 5 set to 1 are handled as interrupt acknowledgment codes. If set to 1, all distributed interrupt codes with bit 5 set to 1 are handled as extended interrupt code. Reset value taken from INTERRUPTMODE signal. See section 82.2.17.

5 RESERVED

4 Self addressing enable (SA) - If set to 1, ports are allowed to send packets to themselves. If set to 0, packets with the same input port as output port are spilled, and an invalid address error is asserted for that port.

3 Time-code control flag mode (TF) - When set to 0, all received time-codes / distributed interrupt codes are handled as time-codes, no matter the value of the control flags (bits 7:6 of the code). When set to 1, the time-code control flags must have value "00" to be considered valid time-codes. Note that the RTRCFG.IE bit has priority over this bit, which means that if RTRCFG.IE is 1, then setting this bit to 0 has no impact. Reset value taken from TIMECODEFILT.

2 Routing table error (RM) - This bit is set for detected errors in Routing table (PORT_SETUP and ROUTE_TABLE) memory.

Table 1488. 0x00000A00 - RTR.RTRCFG - Router configuration / status

1 Timers available (TA) - Constant value 1. Indicates that the router has support for timers, as described in section 82.2.14.

0 SpaceWire Plug and Play available (PP) - Constant value 1. Indicates that the router support SpaceWire Plug and Play, as described in section 82.7.1.

Table 1489. 0x00000A04 - RTR.TC - Time-code

31 10	9	8	7 6	5 0
RESERVED	RE	EN	CF	TC
0x000000	0	*	0x0	0x00
r	rw*	rw	r	r

31: 10 RESERVED

9 Reset time-code (RE) - When this field is written to 1, the RTR.TC.CF and RTR.TC.TC fields are reset. This bit is self-clearing, and always reads 0. Writing 0 has no effect.

8 Enable time-codes (EN) - When set to 1, received time-codes are handled by the router according to the rules described in 82.2.16. When set to 0, all received time-codes are silently discarded. Reset value set through TIMECODEREGEN signal.

7: 6 Time-control flags (CF) - The current value of the router's time-code control flags (bits 7:6 of the latest valid time-code received).

5: 0 Time-counter (TC) - Current value of the router's time counter.

Table 1490. 0x00000A08 - RTR.VER - Version / instance ID

31 24	23 16	15 8	7 0
MA	MI	PA	ID
0x01	0x02	0x00	*
r	r	r	rw

31: 24 Major version (MA) - Holds the major version number of the router. Set to 0x01 for this version.

23: 16 Minor version (MI) - Holds the minor version number of the router. Set to 0x02 for this version.

15: 8 Patch (PA) - Holds the patch number of the router. Set to 0x00 for this version.

7: 0 Instance ID (ID) - Holds the instance ID number of the router. Reset value is set through INSTANVEID[7:0] signal.

Table 1491. 0x00000A0C - RTR.IDIV - Initialization divisor

31 8	7 0
RESERVED	ID
0x000000	*
r	rw

31: 8 RESERVED

7: 0 Initialization clock divisor (ID) - Clock divisor value used by all the SpaceWire links to generate the 10 Mbit/s rate during initialization. Reset value from the IDIVISOR[7:0] signal. For more information about setting the link-rate for SpaceWire ports during initialization see section 82.2.22.

Table 1492. 0x00000A10 - RTR.CFGWE - Configuration port write enable

31	1	0
RESERVED		WE
0x00000000		1
r		rw

31: 1 RESERVED

0 Configuration port write enable (WE) - When set to 1, write accesses to the configuration port area are allowed. When set to 0, write accesses are only allowed to this register. RMAP write and RMAP read-modify-write commands will be replied to with the Status field set to 0x0A (authorization failure), if a reply was requested. The value of this bit has no effect for SpaceWire Plug-and-Play commands.

Table 1493. 0x00000A14 - RTR.PRESCALER - Timer prescaler reload

31	0
RL	
*	
rw*	

31: 0 Timer prescaler reload (RL) - Global prescaler reload value used for generating a common tick for the data character timers, auto-disconnect timers, and distributed interrupt code timers. The prescaler runs on the system clock, and a tick is generated every RTR.PRESCALER.RL+1 CLK cycle. The minimum value of this field is 49. Trying to write a value less than that will result in 49 being written. Reset value is set through RELOAD[31:0] signal. The width of this field is determined by the VHDL generic *timerbits*.

Table 1494. 0x00000A18 - RTR.IMASK - Interrupt mask

31	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED		PE	SR	RS	TT	PL	TS	AC	RE	IA	LE	ME
0x00000		0	0	0	0	0	0	0	0	0	0	0
r		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31: 11 RESERVED

10 SpaceWire Plug-and-Play error (PE) - Generate an interrupt when a SpaceWire Plug and Play error has been detected in the configuration port. The different errors are described in 82.7.1.

9 Spill-if-not-ready (SR) - Generate an interrupt when a packet has been spilled because of the spill-if-not-ready feature described in section 82.2.9.

8 Run-state entry (RS) - Generate an interrupt when a SpaceWire link enters run-state.

7 Time-code / distributed interruptcode tick truncation (TT) - Generate an interrupt when a packet has been spilled because of the time-code / distributed interrupt code truncation feature described in section 82.2.19.1.

6 Packet length truncation (PL) - Generate an interrupt when a packet has been spilled due to the packet length truncation feature described in section 82.2.15.

5 Timeout spill (TS) - Generate an interrupt when a packet has been spilled due to the timeout mechanism.

4 Auxiliary configuration port error (AC) - Generate an interrupt when either a header CRC error, protocol ID error, packet type error, early EOP, or early EEP has been detected in the configuration port.

3 RMAP error (RE) - Generate an interrupt when an error has been detected in the configuration port for an RMAP command such that the PSTS.EC field is set to a non-zero value.

2 Invalid address (IA)- Generate an interrupt when an invalid address error has occurred on a port. See RTR.PSTS:IA bit and section 82.2.11 for a definition of invalid address.

1 Link error (LE) - Generate an interrupt when a link error has been detected on a SpaceWire port.

0 Memory error (ME) - Generate an interrupt when a port buffer error is detected.

Table 1495. 0x00000A1C - RTR.IPMASK - Interrupt port mask

31	0
IE	
0x00000000	
rw	

31: 0 Port interrupt enable (IE) - Set a bit to 1 to enable interrupts to be generated for an error detected in the port with the same number as the bit index. An interrupt is signaled through the IRQ pin, and optionally through a distributed interrupt code. Only bits corresponding to existing ports, as defined by the sum of the VHDL generics *spwports*, *ambaports*, *fifoports* and *customport*, are implemented.

Table 1496. 0x00000A20 - RTR.PIP - Port interrupt pending

31	0
IP	
0x00000000	
wc	

31: 0 Interrupt pending (IP) - When a bit is set to 1, the port with the same number as the bit index was the source of an interrupt. A bit in this field will only be set to 1 for a generated interrupt if the port's corresponding bit in RTR.IPMASK is set, as well as the error types corresponding bit in RTR.IMASK, are set. Only bits corresponding to existing ports, as defined by the sum of the VHDL generics *spwports*, *ambaports*, *fifoports* and *customport*, are implemented.

Table 1497. 0x00000A24 - RTR.ICODEGEN - Interrupt code generation

31 22	21	20	19	18	17	16	15 6	5 0
RESERVED	HI	UA	AH	IT	TE	EN	RESERVED	IN
0x000	0	0	0	0	1	0	0x000	0x00
r	rw	rw	rw	rw	rw	rw	r	rw

0

31: 22 RESERVED

21 AMBA interrupt enable (HI) - Enable AHB interrupt generation.

20 Interrupt code generation un-acknowledge mode (UA) - If this bit is set to 1, an ISR timeout for a distributed interrupt that was generated by the router will clear the bits in the RTR.PIP register that were set when the interrupt was generated. If this bit is set to 0, no extra handling is done on an ISR timeout event, and the bits in RTR.PIP will stay set. See section 82.2.17.

19 Interrupt acknowledgment code handling (AH) - When set to 1, and the router has generated an interrupt code, a received interrupt acknowledgment code with the interrupt number matching the RTR.ICODEGEN.IN field will clear the bits in the RTR.PIP register that were set when the interrupt code was generated. If set to 0, no extra handling of a received interrupt acknowledgment code is done and the bits in RTR.PIP will stay set. This bit is unused when the distributed interrupts are operating in the extended interrupt mode. See section 82.2.17.

18 Interrupt type (IT) - 0 = Level. 1 = Edge. When set to 0, a new interrupt code is distributed as long as RTR.PIP register is non zero. When set to 1, a new interrupt code is distributed only when a bit in RTR.PIP toggles from 0 to 1. See section 82.2.17.

Table 1497. 0x00000A24 - RTR.ICODEGEN - Interrupt code generation

17	Interrupt acknowledgment code to interrupt code timer enable (TE) - If set to 1, the router will wait for the time period specified by the RTR.AITIMER register after the reception of an interrupt acknowledgment code (for which the router generated the corresponding interrupt code) until a new interrupt code is allowed to be generated. If set to 0, the timer is not used, and a new interrupt code is allowed to be generated as soon as the interrupt acknowledgment code has been distributed. This bit is unused when the distributed interrupts are operating in the extended interrupt mode.
16	Interrupt code generation enable (EN) - When 1, distributed interrupt code generation is enabled, and an interrupt code / extended interrupt code can be generated when an internal error event occurs. See section 82.2.17.
15: 6	RESERVED
5: 0	Interrupt number (IN) - Sets the interrupt number of the distributed interrupt code that will be generated when the interrupt code generation feature is enabled (RTR.ICODEGEN.EN = 1). Note that when the distributed interrupts are operating in interrupt with acknowledgments mode, this field must not be set to a value larger than 31, since that would specify an interrupt with acknowledgment code. See section 82.2.17.

Table 1498. 0x00000A28 - RTR.ISR0 - Interrupt code distribution ISR register, interrupt 0-31

31	0
IB	
0x00000000	
wc	

31: 0	Distributed interrupt code ISR bits (IB) - The current value of the distributed interrupt code ISR register for interrupt numbers 0 to 31. Each bit index corresponds to the ISR bit value for the corresponding interrupt number. A bit value of 1 indicates that an interrupt code with the corresponding interrupt number has been received, but not yet acknowledged. A bit value of 0 indicates either that no interrupt code with the corresponding interrupt number has been received, or that the previous interrupt code was either acknowledged or timed out. This register should be normally only be used for diagnostics and / or FDIR.

Table 1499. 0x00000A2C - RTR.ISR1 - Interrupt code distribution ISR register, interrupt 32-63

31	0
IB	
0x00000000	
wc	

31: 0	Distributed interrupt code ISR bits (IB) - The current value of the distributed interrupt code ISR register for interrupt numbers 32 to 63. Each bit index + 32 corresponds to the ISR bit value for the corresponding interrupt number. A bit value of 1 indicates that an extended interrupt code with the corresponding interrupt number has been received. A bit value of 0 indicates either that no extended interrupt code with the corresponding interrupt number has been received, or that the previous interrupt cod has timed out. Note that if the distributed interrupts are operating in interrupt with acknowledgments mode, this register is unused. This register should be normally only be used for diagnostics and / or FDIR.

Table 1500. 0x00000A30 - RTR.ISRTIMER - Interrupt code distribution ISR timer reload

31	0
RL	
*	
rw	

31: 0	Interrupt code distribution ISR timer reload (RL) - Interrupt code distribution ISR timer reload value, counted in prescaler ticks. Each ISR bit has its own timer, which is started and reloaded with the value of this field when an interrupt code / extended interrupt code with the corresponding interrupt number is received (or generated by the router). Reset value is set through IRQTIMEOUTRELOAD[31:0] pins. See section 82.2.17 for details on interrupt code distribution. The width of this field is determined by the VHDL generic *irqtimerbits*.

Table 1501. 0x00000A34 - RTR.AITIMER - Interrupt code distribution ACK-to-INT timer reload

31	0
RL	
*	
rw	

31: 0 Interrupt acknowledgment code to interrupt code timer reload (RL) - Interrupt acknowledgment code to interrupt code timer reload value, counted in prescaler ticks. When an interrupt acknowledgment code is received - for which the router generated the corresponding interrupt code - the interrupt acknowledgment code to interrupt code timer is started and reloaded with the value of this field. Reset value is set through IRQGENRELOAD[31:0] signal. This register is unused when the distributed interrupts are operating in the extended interrupt mode. See section 82.2.17 for details on interrupt code distribution. The width of this field is determined by the VHDL generic *irqtimerbits*.

Table 1502. 0x00000A38 - RTR.ISRCTIMER - Interrupt code distribution ISR change timer reload

31	5	4	0
RESERVED		RL	
0		0	
r		rw	

31: 5 RESERVED

4: 0 Interrupt code distribution ISR change timer reload (RL) - Interrupt code distribution ISR change timer reload value, counted in prescaler ticks. Each time an ISR bit change value, the corresponding ISR change timer is started and reloaded with the value of this field. See section 82.2.17 for details on interrupt code distribution.

Table 1503. 0x00000A40 - RTR.LRUNSTAT - Link running status

31	*N+1*	*N*	1	0
RESERVED		LR		R
0x0000		0x00000		0
r		r		r

31: *N+1* RESERVED

N: 1 Link running status (LR)- Each bit is set to 1 when the link interface for the SpaceWire port with the same number as the bit index is in run-state. If the link interface is not in run-state, the bit is set to 0. The width of this field is determined by the VHDL generic *spwports.*

0 RESERVED

Table 1504. 0x00000A44 - RTR.CAP - Capability

31	26	25	24	23	22	20	19	18	16	15	14	13	12	11	10	9	5	4	0
RESERVED		AF		R	PF		R	RM		R	AA	AX	DP	ID	SD	PC		CC	
0x000		*		0	*		0	*		0	*	*	*	*	*	*		*	
r		r		r	r		r	r		r	r	r	r	r	r	r		r	

31: 26 RESERVED

25: 24 AMBA FIFO size (AF) - The number of entries in the port FIFOs can be determined by the value of this field, according to the formula: Entries = $2^{(RTR.CAP.AF+2)}$

23 RESERVED

Table 1504. 0x00000A44 - RTR.CAP - Capability

22: 20	Port N-char FIFO size (PF) - The number of entries in the port FIFOs can be determined by the value of this field, according to the formula: Entries = $2^{(RTR.CAP.PF+4)}$.
19	RESERVED
18: 16	RMAP maximum data length (RM) - This field specifies the maximum data length in an RMAP read / write command that the configuration port can handle. The length can be determined according to the formula: Length = $2^{(RTR.CAP.RM+2)}$.
15	RESERVED
14	Asynchronous auxiliary time-code / distributed interrupt code support (AA) - Specifies that the router has support for the auxiliary time-code / distributed interrupt code interface inputs to be asynchronous to CLK. See section 82.2.16.
13	Auxiliary time-code / distributed interrupt code support (AX) - Specifies that the router has support for the auxiliary time-code / distributed interrupt code feature described in 82.2.16.
12	Dual port (DP) - Indicates if the SpaceWire ports are implemented with redundant interfaces.
11	Distributed interrupt code support (ID) - Specifies that the router has support for the interrupt distribution scheme, described in 82.2.17.
10	SpaceWire-D support (SD) - Specifies that the router has support for the SpaceWire-D, described in section 82.2.19.
9: 5	Port packet counter bits (PC) - Specifies the number of bits in the port's incoming / outgoing packet counters.
4: 0	Port character counter bits (CC) - Specifies the number of bits in the port's incoming / outgoing character counters.

Table 1505. 0x00000A50 - RTR.PNPVEND - SpaceWire Plug-and-Play - Device Vendor and Product ID

31 16	15 0
VI	PI
*	*
r	r

31: 16	SpaceWire Plug-and-Play Vendor ID (VI) - Double mapping of the VEND bits from the SpaceWire Plug-and-Play Device Vendor and Product ID field. See table 1528. Value is taken from VHDL generic *spwpnpvendid*.
25: 0	SpaceWire Plug-and-Play Product ID (PI) - Double mapping of the PROD bits from the SpaceWire Plug-and-Play Device Vendor and Product ID field. See table 1528. Value is taken from VHDL generic *spwpnpprodid*.

Table 1506. 0x00000A54 - RTR.PNPUVEND - SpaceWire Plug-and-Play - Unit Vendor and Product ID

31 16	15 0
VI	PI
0x0000	0x0000
rw	rw

31: 16	SpaceWire Plug-and-Play Unit vendor ID (VI) - Double mapping of the VEND bits from the SpaceWire Plug-and-Play Unit Vendor and Product ID field (see table 1537).
25: 0	SpaceWire Plug-and-Play Unit product ID (PI) - Double mapping of the PROD bits from the SpaceWire Plug-and-Play Unit Vendor and Product ID field (see table 1537).

Table 1507. 0x00000A58 - RTR.PNPUSN - SpaceWire Plug-and-Play - Unit Serial Number

31 8	7 0
SN	
0x000000	*
rw	

Table 1507. 0x00000A58 - RTR.PNPUSN - SpaceWire Plug-and-Play - Unit Serial Number

31: 0 SpaceWire Plug-and-Play Unit serial number (SN) - Double mapping of the SpaceWire Plug-and-Play Unit Serial Number field (see table 1538). Reset value for bits 3:0 is set through INSTANCEID[7:0] signal.

Table 1508. 0x00000A5C - RTR.PNPNETDISC - SpaceWire Plug-and-Play - Port network discovery enable

31	8	7	0
ND			
0x00000000			
rw			

31: 0 SpaceWire Plug-and-Play Unit network discovery (ND) - Enable network discovery. Only bits corresponding to existing ports, as defined by the sum of the VHDL generics *spwports*, *ambaports*, *fifoports* and *customport*, are implemented.

Table 1509. 0x00000C10,0x00000C20...0x00000DF0 - RTR.OCHARCNT - Outgoing character counter, ports > 0

31	30	0
OR	CC	
0	0x00000000	
wc	rw*	

31 Counter overrun (OR) - This bit is set to 1 when the character counter (RTR.OCHARCNT.CC) overflows. A write with a 1 to this field will clear the whole character counter (including this bit)

30: 0 Character counter (CC) - Number of data characters (EOP, EEP, time-codes, distributed interrupt codes are not included) that have been transmitted on the corresponding port. When the counter reaches its maximum value, it sets the RTR.OCHARCNT.OR bit to 1 and continue counting from zero. A write to this field where bit 30 is set to 1 will reset the RTR.OCHARCNT.OR bit. A write to this field where bit 30 is set to 0 has no effect. The width of the actual counter is configurable with the VHDL generic *charcntbits*.

Table 1510. 0x00000C14,0x00000C24...0x00000DF4 - RTR.ICHARCNT - Incoming character counter, ports > 0

31	30	0
OR	CC	
0	0x00000000	
wc	rw*	

31 Counter overrun (OR) - This bit is set to 1 when the character counter (RTR.ICHARCNT.CC) overflows. A write with a 1 to this field will the whole character counter (including this bit)

30: 0 Character counter (CC) - Number of data characters (EOP, EEP, time-codes, distributed interrupt codes are not included) that have been received on the corresponding port. When the counter reaches its maximum value, it sets the RTR.ICHARCNT.OR bit to 1 and continue counting from zero. A write to this field where bit 30 is set to 1 will reset the RTR.ICHARCNT.OR bit. A write to this field where bit 30 is set to 0 has no effect. The width of the actual counter is configurable with the VHDL generic *charcntbits*.

Table 1511. 0x00000C18,0x00000C28...0x00000DF8 - RTR.OPKTCNT - Outgoing packet counter, ports > 0

31	30	29	0
OR	PC		
0	0x00000000		
wc	rw*		

Table 1511. 0x00000C18,0x00000C28...0x00000DF8 - RTR.OPKTCNT - Outgoing packet counter, ports > 0

31 Counter overrun (OR) - This bit is set to 1 when the packet counter (RTR.OPKTCNT.PC) overflows. A write with a 1 to this field will reset the whole character counter (including this bit).

30: 0 Packet counter (PC) - Number of packets that have been transmitted on the corresponding port. When the counter reaches its maximum value, it sets the RTR.OPKTCNT.OR bit to 1 and continue counting from zero. A write to this field where bit 30 is set to 1 will reset the RTR.OPKTCNT.OR bit. A write to this field where bit 30 is set to 0 has no effect. The width of the actual counter is configurable with the VHDL generic *pktcntbits*.

Table 1512. 0x00000C1C,0x00000C2C...0x00000DFC - RTR.IPKTCNT - Incoming packet counter, ports > 0

31	30 29	0
OR	PC	
0	0x00000000	
wc	rw*	

31 Counter overrun (OR) - This bit is set to 1 when the packet counter (RTR.IPKTCNT.PC) overflows. A write with a 1 to this field will reset the whole character counter (including this bit).

30: 0 Packet counter (PC) - Number of packets that have been received on the corresponding port. When the counter reaches its maximum value, it sets the RTR.IPKTCNT.OR bit to 1 and continue counting from zero. A write to this field where bit 30 is set to 1 will reset the RTR.IPKTCNT.OR bit. A write to this field where bit 30 is set to 0 has no effect. The width of the actual counter is configurable with the VHDL generic *pktcntbits*.

Table 1513. 0x00000E00-0x00000E7C - RTR.MAXPLEN - Maximum packet length, ports > 0

31 25	24 0
RESERVED	ML
0x00	0x000000
r	rw

31: 25 RESERVED

24: 0 Maximum packet length (ML) - Maximum length of packets for which the corresponding port is the input port. This field is only used when the RTR.PCTRL.PL bit (RTR.PCTRLCFG.PL for port 0) is set to 1. See section 82.2.19 for details.

Table 1514. 0x00000E84-0x00000EFC - RTR.CREDCNT - Credit counter, SpaceWire ports

31 12	11 6	5 0
RESERVED	OC	IC
0x00000	0	0
r	r	r

31: 12 RESERVED

11: 6 Out credit counter (OC) - Number of outgoing credits. For each credit, the other end of the link is allowed to send one N-Char.

5: 0 In credit counter (IC) - Number of incoming credits. For each credit, the port is allowed to transmit one N-Char.

Table 1515. 0x00000F00 - RTR.GPO0 - General purpose out, bits 0-31

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
GPO
0x00000000
rw

Table 1515. 0x00000F00 - RTR.GPO0 - General purpose out, bits 0-31

31: 0 General purpose output bits 31:0. The total number of general purpose outputs are configured by the VHDL generics *gpolbits* + *gpopbits*. Bit numbers >= *gpolbits* are pulses and will be cleared one cycle after being set by a register write.

Table 1516. 0x00000F04 - RTR.GPO1 - General purpose out, bits 32-63

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPO																															
0x00000000																															
rw																															

31: 0 General purpose output bits 63:32. The total number of general purpose outputs are configured by the VHDL generics *gpolbits* + *gpopbits*. Bit numbers >= *gpolbits* are pulses and will be cleared one cycle after being set by a register write.

Table 1517. 0x00000F08 - RTR.GPO2 - General purpose out, bits 64-95

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPO																															
0x00000000																															
rw																															

31: 0 General purpose output bits 95:64. The total number of general purpose outputs are configured by the VHDL generics *gpolbits* + *gpopbits*. Bit numbers >= *gpolbits* are pulses and will be cleared one cycle after being set by a register write.

Table 1518. 0x00000F0C - RTR.GPO3 - General purpose out, bits 96-127

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPO																															
0x00000000																															
rw																															

31: 0 General purpose output bits 127:96. The total number of general purpose outputs are configured by the VHDL generics *gpolbits* + *gpopbits*. Bit numbers >= *gpolbits* are pulses and will be cleared one cycle after being set by a register write.

Table 1519. 0x00000F10 - RTR.GPI0 - General purpose in, bits 0-31

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPI																															
N/R																															
r																															

31: 0 General purpose inout bits 31:0. The total number of general purpose inputs are configured by the VHDL generic *gpibits*.

Table 1520. 0x00000F14 - RTR.GPI1 - General purpose in, bits 32-63

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPI																															
N/R																															
r																															

31: 0 General purpose inout bits 63:32. The total number of general purpose inputs are configured by the VHDL generic *gpibits*.

Table 1521. 0x00000F18 - RTR.GPI2 - General purpose in, bits 64-95

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPI																															
N/R																															
r																															

31: 0 General purpose inout bits 95:64. The total number of general purpose inputs are configured by the VHDL generic *gpibits*.

Table 1522. 0x00000F1C - RTR.GPI3 - General purpose in, bits 64-127

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPI																															
N/R																															
r																															

31: 0 General purpose inout bits 127:96. The total number of general purpose inputs are configured by the VHDL generic *gpibits*.

Table 1523. 0x00001004-0x000013FC - RTR.RTCOMB - Routing table, combined port mapping and address control, addresses 1-255

31	30	29	28	27	1	0
SR	EN	PR	HD	PE		PD
N/R	0	N/R	N/R	N/R		N/R
rw	rw	rw	rw	rw		rw

31 Spill-if-not-ready (SR) - This bit is a double mapping of the RTR.RTACTRL.SR bit. See table 1480.

30 Enable (EN) - This bit is a double mapping of the RTR.RTACTRL.EN bit. See table 1480.

29 Priority (PR) - This bit is a double mapping of the RTR.RTACTRL.PR bit. See table 1480.

28 Header deletion (HD) - This bit is a double mapping of the RTR.RTACTRL.HD bit. See table 1480.

27: 1 Port enable bits (PE) - This field is a double mapping of the RTR.RTPMAP.PE field. See table 1479. Only bits corresponding to existing ports, as defined by the sum of the VHDL generics *spwports*, *ambaports*, *fifoports* and *customport*, are implemented.

0 Packet distribution (PD) - This field is a double mapping of the RTR.RTPMAP.PD field. See table 1479.

NOTE: See note for RTR.RTPMAP (table 1479).

Table 1524. 0x00002000-0x00002FFC - RTR.APBAREA - APB address area

31	0
APB	
N/A	
rw	

31: 0 This register area provides access to memory mapped to an external AMBA interface.

82.7.1 SpaceWire Plug-and-Play interface

The configuration port supports parts of the SpaceWire Plug-and-Play protocol described in [SPW-PNP]. The supported fields are listed in table 1527, and explained in more detail in tables 1528 through 1542.

The SpaceWire Plug-and-Play protocol uses standard RMAP commands and replies with the same requirements as presented in section 82.6.1, but with the following differences:

- Protocol Identifier field of a command shall be set to 0x03.
- A command's address fields shall contain a word address. The SpaceWire Plug-and-Play addresses are encoded as shown in table 1525.
- The increment bit in the command's instruction field shall be set to 1, otherwise a reply with Status field set to 0x0A (authorization failure) is sent.
- RMAP Read-modify-write command is replaced by a compare-and-swap operation. The command's data fields shall contain the new data to be written, while the mask fields shall contain the value that the current data must match in order for the new data to be written. If there is a mismatch, a reply with Status field set to 0x0A (authorization failure) is sent.
- The reply packet's Status field can contain the additional status codes described in table 1526.

Table 1525. SpaceWire Plug-and-Play address encoding

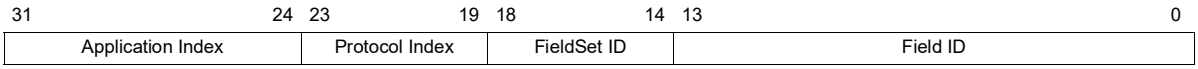


Table 1526. SpaceWire Plug-and-Play status codes

Value	Description
0xF0	Unauthorized access - A write, or compare-and-swap command arrived either when the router was not configured (Device ID field = 0), or the command did not match the owner information saved in the Link Information field and Owner Address fields.
0xF1	Reserved field set - A read, write, or compare-and-swap command's address field points to a non existing field set.
0xF2	Read-only field - A write, or compare-and-swap command's address points to a read-only field.
0xF3	Compare-and-swap-only-field - A write command's address points to a compare-and-swap-only field.

Note that it is not possible to access the SpaceWire Plug-and-Play fields through the AHB slave interface, except for the fields that are double mapped into the configuration port's address space (see section 82.7).

An access (read, write, or compare-and-swap) made either to a field outside the Device Information service, or to a field in an undefined field set within the Device Information service, will generate a reply with the Status field set to 0xF1. An access (read, write, or compare-and-swap) to an undefined or unsupported field in one of the defined field sets, within the Device Information service, is not treated as an error, and the Status field of the reply will be 0x00. Possible write-data for such an access is discarded, and possible read-data returned is always 0.

Please reference the [SPWPNP] for additional details to what is presented in this section.

*Table 1527.*SpaceWire Plug-and-Play support

SpW PnP Address	Register name	Acronym	Service - Field set - Field
0x00000000	SpaceWire Plug-and-Play - Device Vendor and Product ID	RTR.PNPVEND	Device Information - Device Identification - Device Vendor and Product ID
0x00000001	SpaceWire Plug-and-Play - Version	RTR.PNPVER	Device Information - Device Identification - Version
0x00000002	SpaceWire Plug-and-Play - Device Status	RTR.PNPDEVSTS	Device Information - Device Identification - Device Status
0x00000003	SpaceWire Plug-and-Play - Active Links	RTR.PNPACTLNK	Device Information - Device Identification - Active Links
0x00000004	SpaceWire Plug-and-Play - Link Information	RTR.PNPLNKINFO	Device Information - Device Identification - Link Information
0x00000005	SpaceWire Plug-and-Play - Owner Address 0	RTR.PNPOA0	Device Information - Device Identification - Owner Address 0
0x00000006	SpaceWire Plug-and-Play - Owner Address 1	RTR.PNPOA1	Device Information - Device Identification - Owner Address 1
0x00000007	SpaceWire Plug-and-Play - Owner Address 2	RTR.PNPOA2	Device Information - Device Identification - Owner Address 2
0x00000008	SpaceWire Plug-and-Play - Device ID	RTR.PNPDEVID	Device Information - Device Identification - Device ID
0x00000009	SpaceWire Plug-and-Play - Unit Vendor and Product ID	RTR.PNPUVEND	Device Information - Device Identification - Unit Vendor and Product ID
0x0000000A	SpaceWire Plug-and-Play - Unit Serial Number	RTR.PNPUSN	Device Information - Device Identification - Unit Serial Number
0x00004000	SpaceWire Plug-and-Play - Vendor String Length	RTR.PNPVSTRL	Device Information - Vendor / Product String - Vendor String Length
0x00006000	SpaceWire Plug-and-Play - Product String Length	RTR.PNPPSTRL	Device Information - Vendor / Product String - Product String Length
0x00008000	SpaceWire Plug-and-Play - Protocol Count	RTR.PNPPCNT	Device Information - Protocol Support - Protocol Count
0x0000C000	SpaceWire Plug-and-Play - Application Count	RTR.PNPACNT	Device Information - Application Support- Application Count

Table 1528. 0x00000000 - RTR.PNPVEND - SpaceWire Plug-and-Play - Device Vendor and Product ID

31 16	15 0
VEND	PROD
*	*
r	r

31: 16 Vendor ID (VEND) - SpaceWire vendor ID.

15: 0 Product ID (PROD) - Product ID.

Table 1529. 0x00000001 - RTR.PNPVER - SpaceWire Plug-and-Play - Version

31 24	23 16	15 8	7 0
MAJOR	MINOR	PATCH	RESERVED
0x01	0x03	0x00	0x00
r	r	r	r

31: 24 Major version number (MAJOR) - Constant value of 0x01.

23: 16 Minor version number (MINOR) - Constant value of 0x03.

15: 8 Patch / Build number (PATCH) - Constant value of 0x00.

7: 0 RESERVED

Table 1530. 0x00000002 - RTR.PNPDEVSTS - SpaceWire Plug-and-Play - Device Status

31 8	7 0
RESERVED	STATUS
0x000000	0x00
r	r

31: 8 RESERVED

7: 0 Device status (STATUS) - Constant value of 0x00.

Table 1531. 0x00000003 - RTR.PNPACTLNK - SpaceWire Plug-and-Play - Active Links

31 1	0
ACTIVE	R
0x00000000	0
r	r

31: 1 Link active (ACTIVE) - If set to 1, the port with the same number as the bit index is running. If set to 0, the port is not running. For the SpaceWire ports, the corresponding bit will be set to 1 if the link interface is in run-state and the port is not disabled through the Port Control register (RTR.PCTRL.DI = 0).

0 RESERVED

Table 1532. 0x00000004 - RTR.PNPLNKINFO -SpaceWire Plug-and-Play - Link Information

31 24	23 22	21	20 16	15 13	12 8	7	6	5	4 0
OLA	OAL	R	OL	RES	RL	T	U	R	LC
0x00	0x0	0	0x0	0x0	0x0	1	0	0	*
r	r	r	r	r	r	r	r	r	r

31: 24 Owner logical address (OLA) - Shows the value of the Initiator Logical Address field from the last successful compare-and-swap command that set the Device ID field.

23: 22 Owner address length (OAL) - Shows how many of the three Owner Address fields that contain valid data.

21 RESERVED

20: 16 Owner link (OL) - Shows the number of the port which was used for the last successful operation to set the value of the Device ID field.

15: 13 RESERVED

12: 8 Return link (RL) - Shows the number of the port through which the reply to the current read command will be transmitted.

7 Device type (T) - Constant value of 1, indicating that this device is a router.

Table 1532. 0x00000004 - RTR.PNPLNKINFO -SpaceWire Plug-and-Play - Link Information

6	Unit information (U) - Indicates if the unit identification information (Unit Vendor and Product ID field, and Unit Serial Number field) are valid. 0 = invalid, 1 = valid. This bit will be 0 after reset / power-up. Once the Unit Vendor and Product ID field has been written with a non-zero value, this bit will be set to 1.
5	RESERVED
4: 0	Link count (LC) - Shows the number of router ports.

Table 1533. 0x00000005 - RTR.PNPOA0 - SpaceWire Plug-and-Play - Owner Address 0

31	0
RA	
0x00000000	
r	

31: 0	Reply address (RA) - Shows byte 0-3 of the Reply Address from the last successful compare-and-swap command that set to the Device ID field. If there was no Reply Address, then this field is zero.

Table 1534. 0x00000006 - RTR.PNPOA1 - SpaceWire Plug-and-Play - Owner Address 1

31	0
RA	
0x00000000	
r	

31: 0	Reply address (RA) - Shows byte 4-7 of the Reply Address from the last successful compare-and-swap command that set to the Device ID field. If the Reply Address was four bytes or less, then this field is zero.

Table 1535. 0x00000007 - RTR.PNPOA2 - SpaceWire Plug-and-Play - Owner Address 2

31	0
RA	
0x00000000	
r	

31: 0 Reply address (RA) - Shows byte 8-11 of the Reply Address from the last successful compare-and-swap command that set to the Device ID field. If the Reply Address was eight bytes or less, then this field is zero.

Table 1536. 0x00000008 - RTR.PNPDEVID - SpaceWire Plug-and-Play - Device ID

31	0
DID	
0x00000000	
cas	

31: 0 Device ID (DID) - Shows the device identifier. After reset / power-up, or when this field is written to zero, the router is not considered to have an owner. The same applies to the case when the port indicated by the OL bits in the Link Information field is either disconnected, or disabled by setting the RTR.PCTRL.DI bit to 1. This field is only writable through a compare-and-swap operation.

Table 1537. 0x00000009 - RTR.PNPUVEND - SpaceWire Plug-and-Play - Unit Vendor and Product ID

31	16	15	0
VEND		PROD	
0x0000		0x0000	
r		r	

31: 16 Unit vendor ID (VEND) - Shows the unit vendor identifier. This field is read-only through the SpaceWire Plug-and-Play protocol, however it is writable through RMAP and AHB (see section82.7). When this field, ot the PROD field, is written with a non-zero value, the U bit in the Link Information field is set to 1.

15: 0 Unit product ID (VEND) - Shows the unit product identifier. This field is read-only through the SpaceWire Plug-and-Play protocol, however it is writable through RMAP and AHB (see section 82.7). When this field, or the VEND field, is written with a non-zero value, the U bit in the Link Information field is set to 1.

Table 1538. 0x0000000A - RTR.PNPUSN - SpaceWire Plug-and-Play - Unit Serial Number

31	0
USN	
0x00000000	
r	

31: 0 Unit serial number (USN) - Shows the unit serial number. This field is read-only through the SpaceWire Plug-and-Play protocol, however it is writable through RMAP and AHB (see section 82.7).

Table 1539. 0x00004000 - RTR.PNPVSTRL - SpaceWire Plug-and-Play - Vendor String Length

31 15	14 0
RESERVED	LEN
0x00000	0x0000
r	r

31: 15 RESERVED

14: 0 Vendor string length (LEN) - Constant value of 0, indicating that no vendor string is present.

Table 1540. 0x00006000 - RTR.PNPPSTRL - SpaceWire Plug-and-Play - Product String Length

31 15	14 0
RESERVED	LEN
0x00000	0x0000
r	r

31: 15 RESERVED

14: 0 Product string length (LEN) - Constant value of 0, indicating that no product string is present.

Table 1541. 0x00008000 - RTR.PNPPCNT - SpaceWire Plug-and-Play - Protocol Count

31 5	4 0
RESERVED	PC
0x0000000	0x00
r	r

31: 5 RESERVED

4: 0 Protocol count (PC) - Constant value of 0, indicating that no protocols can be managed by using SpaceWire Plug-and-Play.

Table 1542. 0x0000C000 - RTR.PNPACNT - SpaceWire Plug-and-Play - Application Count

31 8	7 0
RESERVED	AC
0x000000	0x00
r	r

31: 8 RESERVED

7: 0 Application count (AC) - Constant value of 0, indicating that no applications can be managed by using SpaceWire Plug-and-Play.

82.8 Vendor and device identifiers

The core has AMBA vendor ID 0x01 and device ID 0x8B (applies to the AHB slave interface). SpaceWire Plug and Play is not a standard yet so identifiers for this protocol have not been assigned.

82.9 Implementation

82.9.1 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

82.10 Configuration options

Table 1543 shows the configuration options of the core (VHDL generics).

*Table 1543.*Configuration options

Generic	Function	Allowed range	Default
input_type	Select receiver type. 0 = Self clocking (xor), 1 = Interface for Frontgrade SpaceWire transceiver, 2 = Single data rate sampling, 3 and 4 = Double data rate sampling, 5 = Self-clocking with external recovery, 6 = Self-clocking with external recovery and DDR register for data. This generic must be set to the same value as the GRSPW2_PHY generic with the same name.	0 - 6	0
output_type	Select transmitter type. 0 = single data rate, 1 = double data rate, 2 = interface for Frontgrade SpaceWire transceiver	0 - 2	0
rxtx_sameclk	Set to one if the same clock net is connected to both the receiver and transmitter (which means this feature is only applicable when the receiver uses sampling). This will remove some unnecessary synchronization registers.	0 - 1	0
fifosize	Sets the number of entries in the 9-bit receiver fifo (N-Char fifo).	16 - 2048	64
tech	Technology for on-chip memories.	-	-
scantest	Enable scan test support	0 - 1	0
techfifo	Enable tech support for on-chip memories.	0 - 1	0
ft	Enable fault-tolerance against SEU errors	0 - 2	0
spwen	Set to 1 to enable SpaceWire ports	0 - 1	1
ambaen	Set to 1 to enable AMBA ports	0 - 1	0
fifoen	Set to 1 to enable FIFO ports	0 - 1	0
spwports	Sets the number of SpaceWire link ports.	0 - 31	2
ambaports	Sets the number of AMBA ports.	0 - 16	0
fifoports	Sets the number of FIFO ports.	0 - 31	0
arbitration	Select the arbitration type. Currently unused.	0 - 1	0
rmap	Include hardware RMAP target in the AMBA ports. RMAP CRC logic will also be added in this case and the rmapcrc generic will have no additional effect. Bit index 0 of the binary representation of the integer corresponds to amba port 1, index 1 to amba port 2 etc.	0 - 16#FFFF#	0
rmapcrc	Enable RMAP CRC logic for the AMBA ports. Bit index 0 of the binary representation of the integer corresponds to amba port 1, index 1 to amba port 2 etc.	0 - 16#FFFF#	0
fifosize2	Sets the number of entries in the 32-bit AHB fifos for the AMBA ports.	4 - 32	32
almostsize	Sets the number of characters from the full or empty conditions that is used as the limit for the almost full and almost empty indications for the FIFO ports.	1 - 32	8

*Table 1543.*Configuration options

Generic	Function	Allowed range	Default
rxunaligned	Receiver unaligned write support. If set, the receiver can write any number of bytes to any start address without writing any excessive bytes. Bit index 0 of the binary representation of the integer corresponds to amba port 1, index 1 to amba port 2 etc.	0 - 16#FFFF#	0
rmapbufs	Sets the number of buffers to hold RMAP replies.	2 - 8	4
dmachan	Sets the number of DMA channels	1 - 4	1
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB interface index value for first AMBA port. Incremented by one for each subsequent port.	0 - NAPBLSV-1	0
paddr	Determines the starting APB address for theAMBA ports together with pmask. For the first port bits 19 downto 12 in the address will be paddr anded with pmask. See the grlib manual for more details. Subsequent ports will have a starting address which is an increment of the previous value. The increment is determined by pmask. 16#FFF# = increment 1, 16#FFE# = increment 2, 16#FFC# = increment 4 etc.	0 - 16#FFF#	0
pmask	Mask for the APB address for the AMBA ports. Anded with paddr to determine the starting address and the range of the area for each port.	0 - 16#FFF#	16#FFF#
pirq	Interrupt number for first AMBA port. Will be incremented by one for each subsequent port.	0 - NAHBIRQ-1	0
cfghindex	AHB slave index.	0 to NAHBSLV-1	0
cfghaddr	ADDR field of the AHB BAR0 defining the AHB slave address space.	0 to 16#FFF#	0
cfghmask	MASK field of the AHB BAR0 defining the AHB slave address space.	0 to 16#FFF#	16#FFF#
ahbslven	Enable AHB slave interface.	0 to 1	0
timerbits	Enables timers when set to a nonzero value. The value determines the number prescaler bits.	0 to 31	0
pnp	Enable Plug and play on the configuration port.	0 to 1	0
autoscrub	Enable automatic scrubbing of routing table and port setup memories.	0 to 1	0
sim	Enable simulation mode. It has no effect when synthesizing but for simulation it shortens the autoscrub period to 1024 clock cycles to make simulation times shorter.	0 to 1	0
dualport	Enable dual port mode (primary and redundant) for SpaceWire ports.	0 to 1	1
charcntbits	Enables characters counters if nonzero and sets the number of counter bits.	0-31	0
pktcntbits	Enables packet counters if nonzero and sets the number of counter bits.	0-31	0
prescalermin	Sets the minimum value of the prescaler reload register. This only affects writes to the register, not the reset value.	-	250
spacewired	Support for SpaceWire-D (packet truncation)	0 - 1	0
interruptdist	Enables for distributed interrupt support: 0: Disabled 1: Interrupt with acknowledgment mode 2: Extended Interrupt mode	0 - 2	0
apbctrl	Enabled the ADHOC AHB-slave interface to be connected to APBCTRL (APB bridge)	0 - 1	0

*Table 1543.*Configuration options

Generic	Function	Allowed range	Default
rmapmaxsize	Set maximum number of bytes supported in a RMAP/PNP read/write	4 - 512	4
gpolbits	Set number of bits for the general purpose level signals	0 - 128	0
gpopbits	Set number of bits for the general purpose pulse signals	0 - 128	0
gpibits	Set number of bits for the general purpose input signals	0 - 128	0
customport	Enabled custom port.	0 - 1	0
inputtest	Enables test-logic (only supported for SDR output type)	0 - 1	0
spwpnpvendid	Specifies the SpaceWire Plug-and-Play Device Vendor ID	0 - 16#FFFF#	0
spwpnpprodid	Specifies the SpaceWire Plug-and-Play Device Product ID	0 - 16#FFFF#	0
porttimerbits	Sets the number of bits for the port's data character timers.	1 - 32	10
irqtimerbits	Sets the number of bits for the Interrupt distribution time-out timers.	1 - 32	10
auxtimeen	Enables the Auxiliary time/interrupt interface.	0 - 1	1
num_txdesc	Specifies the number of entries in the transmit descriptor table	64, 128, 256, 512	64
num_rxdesc	Specifies the number of entries in the receive descriptor table	128, 256, 512, 1024	128
auxasync	0: Auxiliary time/interrupt interface is synchronous to core clock. 1: Auxiliary time/interrupt interface is asynchronous to core clock.	0 - 1	0
hirq	Interrupt line used by the AHB-slave interface.	0 - NAHBIRQ-1	0
codecclkgate	Enable the internal clockgate for the router clocks by setting this generic to 1. Note that this generic is only applicable to the wrapper grspwrouterm, as the clockgate is instantiated in there. If using the router without wrapper (grspwrouter), this generic is not available and the clockgate is disabled	0 - 1	0
internalrstgen	If this generic is set to 1, the reset generators for every clock domain are instantiated internally in the wrapper of the SpW router (grspwrouterm). Otherwise, the user is expected to implement the reset generation from the top level entity. This generic is only available when using the wrapper.	0 - 1	1
rstsrctmr	Enables the Triple Module Redundancy for the asynchronous reset nets of the core	0 - 1	0

82.11 Signal descriptions

Table 1544 shows the interface signals of the core (VHDL ports).

Table 1544. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Core clock	-
RST_CODEC [SPWPORTS-1:0]	N/A	Input	Resets to be used by the internal SpW codecs. There is one per SpW port. This signal is not used by the core if the internal reset generators are enabled in the wrapper (generic internalrst-gen to 1). In that case, it is recommended to set every element of the array to '0'.	Low
CLK_CODEC [SPWPORTS-1:0]	N/A	Input	Main clock to be used by the internal SpW codecs. There is one per SpW port. This signal is not used by the core if the internal reset generators are enabled in the wrapper (generic internalrstgen to 1). In that case, it is recommended to set every element of the array to '0'.	-
RXASYNCRST [SPWPORTS-1:0]	N/A	Input	Asynchronous resets for the SpW RX clocks to be used by the internal SpW codecs. There is one per SpW port. This signal is not used by the core if the internal reset generators are enabled in the wrapper (generic internalrstgen to 1). In that case, it is recommended to set every element of the array to '0'.	Low
RXSYNCRST0 [SPWPORTS-1:0]	N/A	Input	Synchronous resets for the SpW RX clocks to be used by the internal SpW codecs. Used by the primary SpW port (in a dualport configuration). This signal is not used by the core if the internal reset generators are enabled in the wrapper (generic internalrstgen to 1). In that case, it is recommended to set every element of the array to '0'. This generic is only available on the grsp-wrouter entity.	Low
RXCLK0 [SPWPORTS-1:0]	N/A	Input	Receiver clock vector. Used by the primary SpW port (in a dualport configuration). This generic is only available on the grspwrouter entity.	-
RXSYNCRST1 [SPWPORTS-1:0]	N/A	Input	Synchronous resets for the SpW RX clocks to be used by the internal SpW codecs. Used by the redundant SpW port (in a dualport configura-tion). This signal is not used by the core if the internal reset generators are enabled in the wrap-per (generic internalrstgen to 1). In that case, it is recommended to set every element of the array to '0'. This generic is only available on the grsp-wrouter entity.	Low
RXCLK1 [SPWPORTS-1:0]	N/A	Input	Receiver clock vector. Used by the redundant SpW port (in a dualport configuration). This generic is only available on the grspwrouter entity.	-
RXSYNCRST [SPWPORTS *(1+dualport)-1:0]	N/A	Input	Synchronous resets for the SpW RX clocks to be used by the internal SpW codecs. There are one or two per SpW port, depending on the value of the generic dualport. This signal is not used by the core if the internal reset generators are enabled in the wrapper (generic internalrstgen to 1). In that case, it is recommended to set every element of the array to '0'. This generic is only available when using the wrapper (grspwrout-erm).	Low

Table 1544. Signal descriptions

Signal name	Field	Type	Function	Active
RXCLK [SPWPORTS *(1+dualport)-1:0]	N/A	Input	Receiver clock vector. One or two clocks for each SpaceWire link, depending on the value of the generic dualport. This generic is only available when using the wrapper (grspwrouterm).	-
TXSYNCRST [SPWPORTS-1:0]	N/A	Input	Synchronous resets for the SpW TX clocks to be used by the internal SpW codecs. There is one per SpW port. This signal is not used by the core if the internal reset generators are enabled in the wrapper (generic internalrstgen to 1). In that case, it is recommended to set every element of the array to '0'.	Low
TXCLK [SPWPORTS-1:0]	N/A	Input	SpaceWire link transmitter clock. If not using the wrapper or the internal reset generator feature, a clock per link is required. However, if the generic internalrstgen is set to 1, only the first element (LSB) of the array is required, and the same clock will be used for every link. Clock division is done locally.	-
TXCLKN [SPWPORTS-1:0]	N/A	Input	Transmitter inverted default run-state clock. Only used in DDR transmitter mode for technologies not supporting local generation of inverted clock. As for TXCLK, only the first element of the array is required if the internal reset generation is enabled.	-
TESTEN	N/A	Input	Scan test enable.	High
TESTRST	N/A	Input	Scan test reset.	Low
TESTOEN	N/A	Input	Scan test output enable	-
SCANEN	N/A	Input	Scan test enable	-
DI[61:0]	N/A	Input	SpaceWire link data input vector. It should be connected to the number of GRSPW2_PHY entities corresponding to the number of SpaceWire links in the router. 1:0 go to link 0, 3:2 to link 1 etc.	-
DVI[61:0]	N/A	Input	SpaceWire link data valid input vector. It should be connected to the number of GRSPW2_PHY entities corresponding to the number of SpaceWire links in the router. 1:0 go to link 0, 3:2 to link 1 etc.	
DCONNECT [61:0]	N/A	Input	Disconnect vector. It should be connected to the number of GRSPW2_PHY entities corresponding to the number of SpaceWire links in the router. 1:0 go to link 0, 3:2 to link 1 etc.	
DCONNECT2 [61:0]			Copy of the signal DCONNECT as part of the Triple Modular Redundancy protection.	
DCONNECT3 [61:0]			Copy of the signal DCONNECT as part of the Triple Modular Redundancy protection.	
DO[61:0]	N/A	Output	Data output vector. In Frontgrade PHY mode two bits per link are used. 1:0 correspond to link 0, 3:2 to link 1 and so on. In the other two modes only the lower of the two bits in each pair are used.	

Table 1544. Signal descriptions

Signal name	Field	Type	Function	Active
SO[61:0]	N/A	Output	Strobe output vector. In Frontgrade PHY mode two bits per link are used. 1:0 correspond to link 0, 3:2 to link 1 and so on. In the other two modes only the lower of the two bits in each pair are used.	
AHBMI	*	Input	AHB master in signals.	-
AHBMO	*	Output	Vector of AHB master out signals. One set of signals for each AMBA port.	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBSI	*	Input	AHB slave in signals.	-
AHBSO	*	Output	AHB slave out signals.	-
RI	RMAPEN[30:0]	Input	RMAP enable signal for each AMBA port. AMBA port 0 is connected to RMAPEN[0] and port 1 to [1] etc.	-
	IDIVISOR[7:0]	Input	Initialization divisor value for the SpaceWire links.	-
	TXWRITE[30:0]	Input	Transmitter FIFO write signals for the FIFO interfaces.	-
	TXCHAR[30:0][8:0]	Input	Transmitter character signals for the FIFO interfaces.	-
	RXREAD[30:0]	Input	Receiver FIFO read signals for the FIFO interfaces.	-
	TICKIN[30:0]	Input	Tick input signals for the FIFO and AHB interfaces	High
	TIMEIN[30:0][7:0]	Input	Time input signals for the FIFO and AHB interfaces.	-
	RELOAD[31:0]	Input	Prescaler default reload value (set during reset).	-
	RELOADN[31:0]	Input	Individual timer default reload value (set during reset).	-
	TIMEREN	Input	Sets the reset value for the timer enable bit in the port control registers..	-
	TIMECODEEN	Input	Enable time-code functionality. If 0 no time-codes will be propagated by the router. When 1 time-code functionality is enabled.	High
	CFGLOCK	Input	Lock configuration port accesses from all ports except port 1. When set to 0 the configuration port can be accessed from all ports (if not individually disabled through register). If set to 1 the configuration area can only be accessed from port 1 (the AHB slave interface is not affected byt this). The configuration enable bit for port 1 does not have any effect in this case.	High
	SELFADDREN	Input	Reset value for selfaddren register bit. This bit enables ports to address themselves i.e a packet received on a port will be transmitted on the same port. If not enabled the packet will be discarded with an invalid address error.	High
	LINKSTARTREQ[31:1]	Input	Reset value for the linkstartreq register bit. When set ports will be automatically started (provided they are not disabled) if a packet needs to be transmitted on them.	

Table 1544. Signal descriptions

Signal name	Field	Type	Function	Active
RI	AUTODCON-NECT[31:1]	Input	Reset value for the autodconnect register bit. When set ports started with the linkstartreq feature will automatically disconnect after a timeout period of no activity.	
	INSTANCEID[7:0]	Input	Sets the reset value of the instance ID field of the version register.	
	ENBRIDGE[31:0]	Input	Reset value for bridge mode on FIFO ports	
	ENEXTTIME[30:0]	Input	Sets the reset value for the external time selection bit for FIFO interfaces. When 0 a tick-in on the corresponding TICKIN signal causes the internal time-counter to be incremented and the new value to be transmitted. When 1 the value from the TIMEIN signal will be used. In both causes the control flags are taken from TIMEIN[7:6].	High
	AUXTICKIN	Input	Time counter tick input. Increments internal time-counter (or set time-counter to auxtimein) and transmits the new value.	
	AUXTIMEINEN	Input	Select between: internal time-counter or auxtimein	
	AUXTIMEIN[7:0]	Input	The time-code or interrupt- / interrupt-acknowledge-code	
	IRQTIMEOUTRE-LOAD[31:0]	Input	Reset value for interrupt timeout value	
	AHBSO *	Input	ADHOC AHB-slave output	
	INTERRUPTCO-DEEN	Input	Reset value for interrupt enable	
	PEPEN	Input	Reset value for PNP enable	
	TIMECODEFILT	Input	Reset value for Time-code control flag mode (TF)	
	INTERRUPTFWD	Input	Reset value for Distributed interrupt code enable (IC) and Auxiliary distributed interrupt codes enable (AI)	
	SPILLIFN-RDY[31:1]	Input	Reset value for spill-if-not-read	
	TIMECODEREGEN	Input	Reset value for global time-code enable	
	GPI[127:0]	Input	General purpose register input	
	STATICROUTEEN	Input	Reset value for Static routing enable (SR)	
	SPWCLKLOCK	Input	Lock signal for codec clock generation. Only used when using the wrapper (grspwrouterm).	
	IRQGENRE-LOAD[31:0]	Input	Reset value for Interrupt acknowledgment code to interrupt code timer reload (RL)	
	INTERRUPTMODE	Input	Reset value for Enable extended distributed interrupts (EE) - If set to 0, all distributed interrupt codes with bit 5 set to 1 are handled as interrupt acknowledgment codes. If set to 1, all distributed interrupt codes with bit 5 set to 1 are handled as extended interrupt code.	
	TESTD	Input	Test input for SpaceWire data	
	TESTS	Input	Test input for SpaceWire Strobe	
	TESTINPUT	Input	Test input enable	

Table 1544. Signal descriptions

Signal name	Field	Type	Function	Active
RO	RXCHARAV[30:0]	Output	Receiver data available vector. One signal per FIFO port.	-
	RXAEMPTY[30:0]	Output	Receiver FIFO almost empty vector. One signal per FIFO port.	-
	TXFULL[30:0]	Output	Transmitter FIFO full vector. One signal per FIFO port.	-
	TXAFULL[30:0]	Output	Transmitter FIFO almost full vector. One signal per FIFO port.	-
	RXCHAR[30:0][8:0]	Output	Receiver FIFO character output vector. One character vector per FIFO port.	-
	AUXTICKOUT	Output	Tick-out signal that is asserted when a valid time-code or interrupt- / interrupt-acknowledge-code has been received.	High
	AUXTIME-OUT[7:0]	Output	Contains the received time-code or interrupt- / interrupt-acknowledge-code when AUXTICK-OUT is asserted.	-
	AUXTICKINDONE	Output	Asserted when a time-code or interupt- / interrupt-acknowledge-code sent via the AUX-TICKIN signal has been accepted for transmission.	High
	TICKOUT[30:0]	Output	Tick out vector. One signal per FIFO or AMBA port.	High
	TIME-OUT[30:0][7:0]	Output	Time-code output vector. One time-code signal per FIFO or AMBA port.	-
	GERROR	Output	Global error. Asserted high when one or both of LERROR and MERROR are asserted.	High
	LERROR	Output	Asserted when one or more link errors have been detected on the SpaceWire ports. Asserted until the corresponding status bits have been c leared.	High
	MERROR	Output	Asserted when one or more uncorrectable parity errors have been detected in the on-chip memories. Asserted until the corresponding status bits have been cleared.	High
	LINKRUN[30:0]	Output	Each bit is asserted (set to 1) when the corresponding link is in run-state. Bit 0 corresponds to port 1, bit 1 to port 2 etc. This is only valid if the port is configured as a SpaceWire link.	
	PORTERR[31:0]	Output	Bit[n] is asserted when port errors have been detected on the SpaceWire ports[n]. Asserted until the corresponding status bits have been c leared.	High
	SPWEN[30:0]	Output	Signals that SpaceWire port is enabled (power-down nand powerdownrx)	High
	AHBSI *	Output	Ad hoc AHB-slave input	-
	POWER-DOWN[30:0]	Output	TX reset and Link disable	
	POWER-DOWNRX[30:0]	Output	Link disabled	
	GPO[127:0]	Output	General purpose register output	-

Table 1544. Signal descriptions

Signal name	Field	Type	Function	Active
RO	RESET	Output	Signal which indicates that the SpW Router is globally reset. Its value is that of the bit 7 of the Router Configuration/Status register (offset: 0xA00). It shall be used to generate the resets of the core. This process is internally done by the wrapper grspwrouterm if the generic internalrst-gen is set to 1.	Low
	RXRST[30:0]	Output	Internal reset generated by each SpW link transmitter for synchronization purpose between transmitter and both its receiver channels. It shall be used to generate the asynchronous and synchronous receiver resets of each port. This process can be skipped by activating the internal reset generation in the wrapper (internalrstgen set to 1).	High
	CLOCKGATE[30:0]	Output	Signal to indicate that the clocks for a specific link can be clockgated. They are used internally by the wrapper if the generic internalrstgen is set to 1.	High
* see GRLIB IP Library User's Manual				

82.12 Signal definitions and reset values

The signals and their reset values are described in table 1545.

*Table 1545.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
spw_clk	Input	Transmitter default run-state clock	Rising edge	-
spw_rxd	Input, LVDS	Data input, positive	High	-
spw_rxdn	Input, LVDS	Data input, negative	Low	-
spw_rxs	Input, LVDS	Strobe input, positive	High	-
spw_rxsn	Input, LVDS	Strobe input, negative	Low	-
spw_txd	Output, LVDS	Data output, positive	High	Logical 0
spw_txdn	Output, LVDS	Data output, negative	Low	Logical 1
spw_txs	Output, LVDS	Strobe output, positive	High	Logical 0
spw_txsn	Output, LVDS	Strobe output, negative	Low	Logical 1

82.13 Timing

The timing waveforms are shown in figure 223 and 224. Timing parameters are defined in table 1546 and 1547.

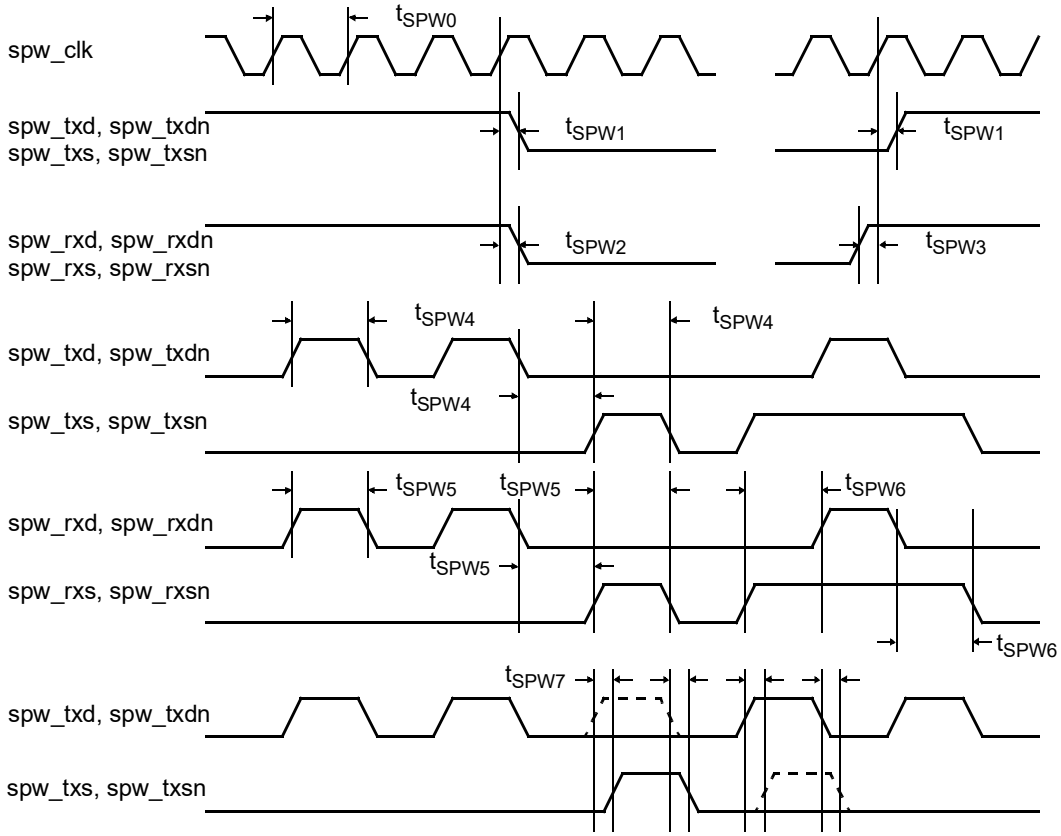


Figure 223. Timing waveforms

*Table 1546.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{SPW0}	transmit clock period	-	TBD	-	ns
t_{SPW1}	clock to output delay	rising *spw_clk* edge	TBD	TBD	ns
t_{SPW2}	input to clock hold	-	-	-	not applicable
t_{SPW3}	input to clock setup	-	-	-	not applicable
t_{SPW4}	output data bit period	-	-		*clk* periods
		-	t_{SPW0} - TBD	t_{SPW0} +TBD	ns
t_{SPW5}	input data bit period	-	TBD	-	ns
t_{SPW6}	data & strobe edge separation	-	TBD	-	ns
t_{SPW7}	data & strobe output skew	-	-	TBD	ns

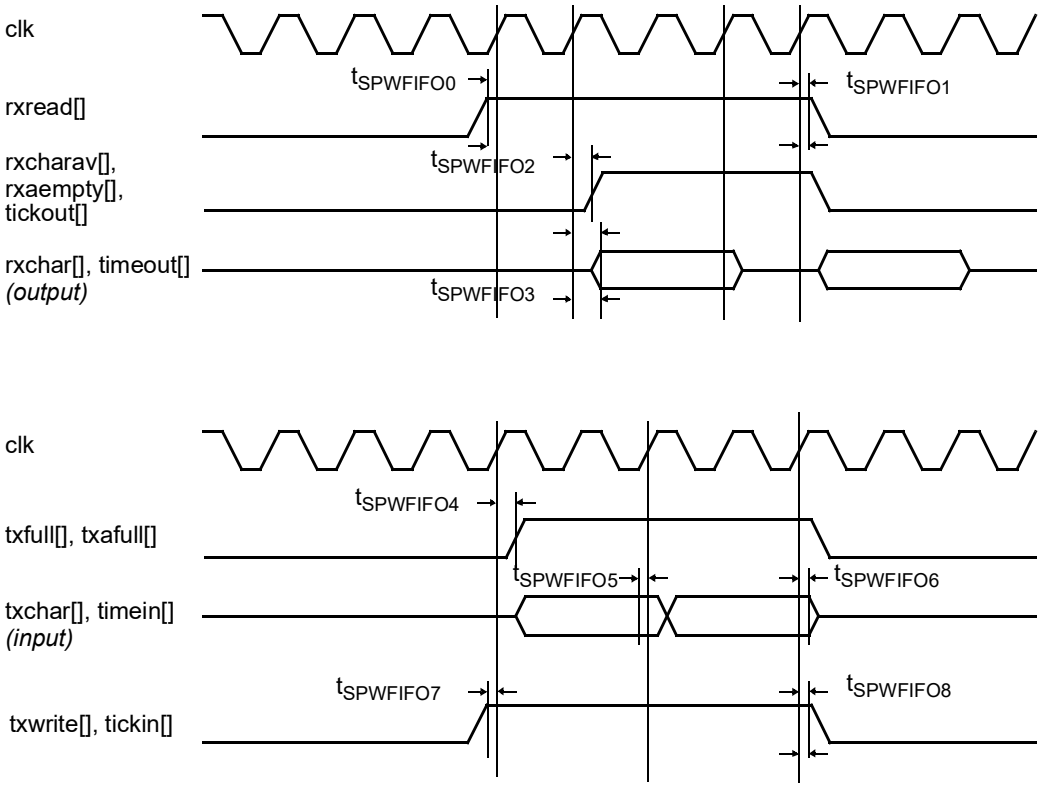


Figure 224. Timing waveforms - FIFO interface

Table 1547. Timing parameters - FIFO interface

Name	Parameter	Reference edge	Min	Max	Unit
$t_{SPWFIFO0}$	input to clock setup	rising clk edge	TBD	TBD	ns
$t_{SPWFIFO1}$	input from clock hold	rising clk edge	TBD	TBD	ns
$t_{SPWFIFO2}$	clock to output delay	rising clk edge	TBD	TBD	ns
$t_{SPWFIFO3}$	clock to rxchar, timeout output delay	rising clk edge	TBD	TBD	ns
$t_{SPWFIFO4}$	clock to output delay	rising clk edge	TBD	TBD	ns
$t_{SPWFIFO5}$	txchar, timein input to clock setup	rising clk edge	TBD	-	ns
$t_{SPWFIFO6}$	txchar, timein input from clock hold	rising clk edge	TBD	-	ns
$t_{SPWFIFO7}$	input to clock setup	rising clk edge	TBD	-	ns
$t_{SPWFIFO8}$	input from clock hold	rising clk edge	TBD	-	ns

82.14 Instantiation

This example of instantiation assumes that the router wrapper is used and the internal reset generators are automatically included. If the user prefers to generate the resets externally, it is advised to use the wrapper as a baseline to understand how some signals are taken into account for each clock domain.

```
-- Since the resets and clockgate are generated internally in the wrapper,
-- the only element from TXCLK and TXCLKN to be used is the bit '0' (spw_clk).
-- The others remain unconnected
txclk_array(0)                            <= spw_clkl;
txclkn_array(0)                           <= spw_clkln;
txclk_array(CFG_SPW_SPWPORTS-1 downto 1)  <= (others => '0');
txclkn_array(CFG_SPW_SPWPORTS-1 downto 1) <= (others => '0');

router0 : grspwrouterm
        generic map (
          input_type   => 0,
          output_type  => 0,
          rxtx_sameclk => 0,
          fifosize     => 64,
          tech         => CFG_SPW_TECH,
          scantest     => 0,
          techfifo     => CFG_SPW_TECHFIFO,
          ft           => 0,
          spwen        => 1,              -- Enable spacewire ports
          ambaen       => 1,              -- Enable AMBA interfaces
          fifoen       => 0,              -- Disable FIFO interfaces
          spwports     => CFG_SPW_SPWPORTS,
          ambaports    => 1,              -- Number of AMBA ports
          fifoports    => 0,              -- Number of FIFO ports
          arbitration  => 0,
          rmap         => CFG_SPW_RMAP,
          rmapcrc      => CFG_SPW_RMAPCRC,
          fifosize2    => 32,
          almostsize   => 1,              -- Only used for FIFO ports
          rxunaligned  => CFG_SPW_RXUNALIGNED,
          rmapbufs     => 4,
          dmachan      => 1,
          hindex       => 4,              -- Starting index
          pindex       => 10,             -- Starting index
          paddr        => 16#0d0#,        -- Starting base address
          pmask        => 16#ff0#,        -- Each reg base on 4 KiB boundary
          pirq         => 20,             -- Starting IRQ
          cfghindex    => 3,
          cfghaddr     => 16#800#,
          cfghmask     => 16#ff0#,
          ahbslven     => 1,
          timerbits    => 0,
          pnp          => 0,
          autoscrub    => 0,
          sim          => 0,              -- Simulation mode, not used
          dualport     => CFG_SPW_DUALPORT,
          charcntbits  => 0,              -- Character counters disabled
          pktcntbits   => 0,              -- Packet counters disabled
```

```
    prescalermin => 250,            -- Minimum value for writes to reload reg
    spacewired   => 1,
    interruptdist=> 2,
    apbctrl      => 0,
    rmapmaxsize  => 4,
    gpolbits     => 0,
    gpopbits     => 0,
    gpibits      => 0,
    customport   => 0,
    codecclkgate => 0,
    inputtest    => 0,
    porttimerbits=> 10,
    irqtimerbits => 10,
    auxtimeen    => 1,
    num_txdesc   => 64,
    num_rxdesc   => 128,
    auxasync     => 0)
  port map(
    rst          => rstn,
    clk          => hclk,
    rst_codec    => (others => '0'), -- Resets generated internally
    clk_codec    => (others => '0'), -- Clockgate generated internally
    rxasyncrst   => (others => '0'), -- Resets generated internally
    rxsyncrst    => (others => '0'), -- Resets generated internally
    rxclk        => rxclko,
    txsyncrst    => (others => '0'), -- Resets generated internally
    txclk        => txclk_array,     -- Only the element 0 will be used (spw_clkl)
    txclkn       => txclkn_array,    -- Only the element 0 will be used (spw_clkln)
    testen       => testen,
    testrst      => testrst,
    scanen       => scanen,
    testoen      => testoen,
    di           => di,
    dvi          => dvi,
    dconnect     => dconnect,  -- From GRSPW2_PHY
    dconnect2    => dconnect2, -- From GRSPW2_PHY
    dconnect3    => dconnect3, -- From GRSPW2_PHY
    do           => do,
    so           => so,
    ahbmi        => ahbmi,
    ahbmo        => ahbmo,
    apbi         => apbi,
    apbo         => apbo,
    ahbsi        => ahbsi,
    ahbso        => ahbso,
    ri           => ri,
    ro           => ro,
    mtesti       => mtesti,
    mtesto       => mtesto,
    mtestclk     => mtestclk
    );
```

83 SPWTDP - SpaceWire - Time Distribution Protocol

83.1 Overview

This interface implements the SpaceWire - Time Distribution Protocol (TDP). The protocol provides capability to transfer time values and synchronise them between onboard users of SpaceWire network. The time values are transferred as CCSDS Time Codes and synchronisation is performed through SpaceWire Time-Codes. The core also provides datation services. The core operates in an AMBA APB bus system. The AMBA APB bus is used for configuration, control and status handling. The interface is coupled with a SpaceWire node with AMBA AHB master and RMAP target implementation.

83.2 Protocol

The initiator and target maintain their own time locally. The Time Distribution Protocol provides the means for transferring time of initiator to targets and for providing a synchronization point in time. The time is transferred by means of an RMAP write command carrying a CCSDS Time Code (time message). The synchronization event is signaled by means of transferring a SpaceWire Time-Code. The transfer of the SpaceWire Time-Code is synchronized with time maintained by the initiator. To distinguish which SpaceWire Time-Code is to be used for synchronization, the value of SpaceWire Time-Code is transferred from initiator to target by means of an RMAP write command prior to actual transmission of SpaceWire Time-Code itself. When there is more than one target the CCSDS Time Code need to be transferred to each individual target separately [SPWCUC].

83.3 Functionality

The block diagram below explains the complete system.

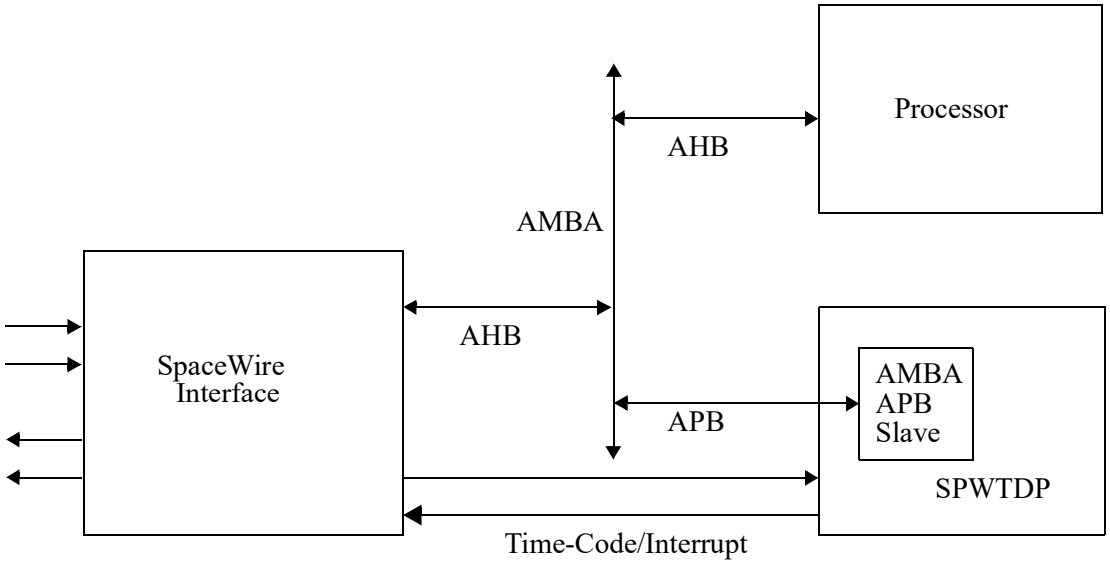


Figure 225. Block diagram

The foreseen usage of this core is to distribute and synchronise time between an initiator SPWTDP core and one or more target SPWTDP (slave) cores using the SpaceWire interface for communication between them.

The system can act as initiator (time master) and target being able to send and receive SpaceWire Time-Codes. The initiator requires SpaceWire link interface implements an RMAP initiator. The Target requires SpaceWire link interface implements an RMAP target. The SPWTDP component is a part of this system providing SpaceWire Time-Codes, CCSDS Time Codes, datation, time-stamping of

distributed interrupts, support for transmission of CCSDS Time Codes through RMAP and support for latency measurement and correction. In this implementation the CCSDS Time Codes carried between the SpaceWire network is based on CCSDS Unsegmented Code format (CUC) which is explained below [CCSDS]. The table below shows an example Preamble Field (P-Field) which corresponds to 40 bits of coarse time and 24 bits of fine time.

83.3.1 CCSDS Unsegmented Code: Preamble Field (P-Field)

*Table 1548.*CCSDS Unsegmented Code P-Field definition

Bit	Value		Interpretation
0	"1"		Extension flag, P-Field extended with 2nd octet
1-3	"010"	Agency-defined epoch (Level 2)	Time code identification
4 - 5	"11"	(number of octets of coarse time) + 1	Detail bits for information on the code
6 - 7	"11"	(number of octets of fine time)	
8	"0"		Extension flag, P-Field not extended with 3rd octet
9-10	"01"	Number of additional octets of the coarse time.	added to octet 1
11-13	"000"	Number of additional octets of the fine time.	added to octet 1
14-15			RESERVED

83.3.2 CCSDS Unsegmented Code: Time Field (T-Field)

For the unsegmented binary time codes described herein, the T-Field consists of a selected number of contiguous time elements, each element being one octet in length. An element represents the state of 8 consecutive bits of a binary counter, cascaded with adjacent counters, which rolls over at a modulo of 256.

*Table 1549.*Example CCSDS Unsegmented Code T-Field with 32 bit coarse and 24 bit fine time

CCSDS Unsegmented Code									
Preamble Field	Time Field								
	Coarse time						Fine time		
-	2^{31} 2^{24}	2^{23} 2^{16}	2^{15} 2^{8}	2^{7} 2^{0}	2^{-1} 2^{-8}	2^{-9} 2^{-15}	2^{-16} 2^{-24}		
0:15	0			31	32		55		

The basic time unit is the second. The T-Field coarse time (seconds) can be maximum 56 bits and minimum 8 bits. The T-Field fine time (sub seconds) can be maximum 80 bits and minimum of 0 bits.

The number of bits representing coarse and fine time implemented in this core can be obtained by reading the DPF bits of Datation Preamble Field register.

The coarse time code elements are a count of the number of seconds elapsed from the initial time value. This code is not UTC-based and leap second corrections do not apply according to CCSDS.

83.3.3 Time generation

The core consist of time generator which is the source for time in this system. The core may act as initiator or a target but both have their respective time generator. The Elapsed Time (ET) counter is implemented complying with the CUC T-Field. The number of bits representing coarse and fine time of a ET counter implemented in a design can be obtained by reading the DPF bits of Datation Preamble Field register.

The ET counter can be incremented either using an internal frequency synthesizer or by using an external enable signal. The External ET Increment Enable bit in Configuration 0 register must be enabled if external inputs are to be used.

Increment ET using internal frequency synthesizer:

The counter is incremented on the system clock only when enabled by the frequency synthesizer. The binary frequency required to determine the counter increment is derived from the system clock using a frequency synthesizer (FS). The frequency synthesizer is incremented with a pre-calculated increment value, which matches the available system clock frequency. The frequency synthesizer generates a tick every time it wraps around, which makes the ET time counter to step forward with the precalculated increment value. The output of frequency synthesizer is used for enabling the increment of ET counter. The increment rate of the ET counter and frequency synthesizer counter should be set according to the system clock frequency. The ET counter increment rate is set by providing values to ETINC bits in Configuration 2 register and frequency synthesizer counter is set by providing values to FSINC bits in Configuration 1 register. The following table specifies some example ETINC and FSINC values for some frequencies. The below values are also obtained for Coarse time width 32, Fine time width 24 and Frequency synthesizer width of 30. To calculate for other frequencies and configuration refer the spreadsheet provided along with this document.

Table 1550. Example values of ETINC and FSINC for corresponding frequencies

Frequency	ETINC	FSINC
50 MHz	0	360287970
250 MHz	0	72057594
33333333	2	135107990

Increment ET using external input:

The EP register in Configuration 0 specify whether to increment the ET counter based on rising or falling edge of the external enable signal. Also the ETINC bits in Configuration 2 register specify from which bit the ET counter must increment.

The following section describes the cores capabilities if it configured as initiator or target.

83.3.4 Initiator

An initiator is a SpaceWire node distributing CCSDS Time Codes and SpaceWire Time-Codes. It is also an RMAP initiator, capable of transmitting RMAP commands and receiving RMAP replies. There is only one active initiator in a SpaceWire network during a mission phase.

The initiator performs the following tasks

- Transmission of SpaceWire Time-Codes

The SpaceWire Time-Codes are provided by this component and transmission of those codes to targets should be performed by a SpaceWire interface.

- Transmission of CCSDS Time Codes through RMAP
- Datation, time-stamping and latency measurement

83.3.5 Target

A target is a SpaceWire node receiving CCSDS Time Codes and SpaceWire Time-Codes. A target is also an RMAP target, capable of receiving RMAP commands and transmitting RMAP replies. There can be one or more targets in a SpaceWire network.

The target performs the following tasks

- Reception of SpaceWire Time-Codes

The SpaceWire Time-Codes sent from initiator are received by SpaceWire interface and provided to this component in target.

- Reception of CCSDS Time Codes through RMAP
- Qualification of received time messages (CCSDS Time Codes) using SpaceWire Time-Codes
- Initialization and Synchronisation of received CCSDS Time Codes with Elapsed Time counter available in this component
- Datation, time-stamping and latency correction
- Jitter and drift mitigation (the MA bit in Status 0 register specify the availability of this service)

83.3.6 Configuring initiator and target

The core is interfaced via an AMBA Advanced Peripheral Bus (APB) slave interface, providing a register view that is compatible with the Time Distribution Protocol (TDP). The core must be configured according to the requirement either as initiator or target.

- Initializing initiator

The initiator transmits the SpaceWire Time-Codes out of the core only when the Transmit Enable TE bit in Configuration 0 register is enabled. The ET counter in initiator can be initialized (to provide any initial value). Initialization is done by writing a time value into the Command Elapsed Time registers available in the command field, the NC bit in the Control register of command field should be enabled to initialize the time value stored in the Command Elapsed Time registers to be the local time (Transmit Enable TE bit in Configuration 0 register must be enabled). The NC bit in the Control register will disable itself when the time is initialized. The INSYNC bit in Status 0 register will enable when initialization is performed. The MAPPING bits in Configuration 0 register determines the interval between SpaceWire Time-Code transmissions which is explained in detail in the section below.

The target time must be configured with time values from the initiator. The targets register space must be configured and controlled through RMAP by an initiator to achieve time synchronisation. The target time synchronisation is explained in detail under the section initialization and synchronisation of target through RMAP.

83.3.7 SpaceWire Time-Code

SpaceWire Time-Codes are continuously transmitted from an initiator node (time master) to all slave nodes. The transmission of the SpaceWire Time-Code is synchronized with the ET counter in the initiator node. The six bits of the Time-Code time information correspond to six bits of the local ET counter (MAPPING bits in Configuration 0 register determines its exact mapping and interval between SpaceWire Time-Code transmissions). Value of 0b00000 for MAPPING bits in Configuration 0 register will send SpaceWire Time-Code at every Second. When the value is 0b00001 SpaceWire Time-Codes are sent at every 0.5 Seconds interval and so on (maximum value of MAPPING can be 0b11111 but this value cannot be more than the number of bits implemented as fine time). The ET bits with lower weights than the size bits mapped to Time Codes time information bits are all zero at time of SpaceWire Time-Codes transmission. The Table below shows an example Local ET counter and Mapping. If the Coarse time is 32 bits and Fine time is 24 bits and mapping value is 6 then 0 to 31 is coarse(32 bits), 32 to 55 is fine time and mapped SpaceWire Time-Code is 32 to 37.

Table 1551. Example Local ET counter with Mapping values

0														25	26	27	28	29	30	31	32	33	34	35	36	37	38				55
Mapping Values																				0	1	2	3	4	5	6	7				24
If the Mapping value is 6 then the mapped SpaceWire Time-Codes is 32 to 37																					32	33	34	35	36	37					

Table 1551. Example Local ET counter with Mapping values

If the Mapping value is 0 then the mapped SpaceWire Time-Codes is 26 to 31	26	27	28	29	30	31											
If the Mapping value is 5 then the mapped SpaceWire Time-Codes is 31 to 36						31	32	33	34	35	36						
If the Mapping value is 7 then the mapped SpaceWire Time-Codes is 33 to 38								33	34	35	36	37	38				

83.3.8 Initialization and synchronisation of target through RMAP

An initiator must provide the time values and set the target in order to get the time synchronized. The below text explains how an initiator can synchronise the target.

The SPWTC in Control register of initiator core component should be configured initially with a SpaceWire Time-Code value at which the time message needed to be transferred. When the SpaceWire Time-Code generated internally using the ET counter matches the SPWTC in Control register a Time Message TM interrupt will be generated (TME bit Time Message Enable should be enabled in the Interrupt Enable register). Based on this interrupt the local time (ET counter) in initiator should be accessed from the Datation registers and used to calculate the time message needed to be transmitted.

- Time message generation

The Time message transmitted using RMAP should be an exact mapping of the Command field (explained under Registers section). The Time message transmitted should write the Command field available in target. Control register available in Command field specify weather the target should be initialized or synchronized, at which SpaceWire Time-Codes it should happen (synchronization event) and details of coarse and fine time available in the time message. The New code NC bit available in Control register should be enabled and if the target should be initialized then Init Sync IS bit in Control register must be enabled otherwise target will be synchronized.

The Command Elapsed Time in time message are calculated from the local time (ET counter) available in the initiator. The local time can be obtained by reading the Datation Field of initiator component. While reading the Datation registers always the total implemented coarse time and fine time must be read in order (from 0 till the implemented Datation Elapsed Time registers). The DPF of Datation Preamble Field register gives the coarse and fine time implemented which gives the total local ET counter (coarse + fine width).

For example if the implementation has 32 bit coarse and 24 bit fine time then it is enough to access the first two Datation Elapsed Time registers (0 and 1). The 32 bits of Datation Elapsed Time 0 and only the most significant 24 bits (31 to 8) of Datation Elapsed Time 1 registers (32 + 24 =56 bits) represents the local time. These 56 bits only be used for Command Elapsed time (time message) calculation.

The SpaceWire Time-Codes at which the Time Message interrupt generated is embedded in the local ET counter. The Command Elapsed time which is transmitted as time message should be an incremented time value of this SpaceWire Time-Code and Command Elapsed time bits with lower weights than the size bits mapped to SpaceWire Time-Code time information bits are all must be zero.

The incremented time value is to make the initialization or synchronisation of time message in target will happen after the reception of qualifying SpaceWire Time-Codes. The qualifying SpaceWire Time-Code is embedded in the Command Elapsed time (part of time message) sent from initiator. This qualifying SpaceWire Time-Code value should also be written in the SPWTC in Control section of the time message.

- Time qualification in target

In target, the Command field will contain the time message when it is written by the initiator through RMAP. When the SPWTC of Control register in Command field matches with a received SpaceWire Time-Code then initialization or synchronization will occur (according to NC bit and IS bit in the Control register) to the local ET counter of the target SPWTDP component. The Time message qualified TCQ bit in the status register will enable itself, this bit will disable itself when the conditions for

time message qualification is achieved (SPWTC of Control register matches with a received SpaceWire Time-Code) but no new time message is received (NC bit is zero). When the local ET counter is initialized or synchronized the NC bit in the control register will disable itself. The INSYNC bit in Status 0 register will enable when initialization is performed specifying the target is initialized. Initialization completely writes time message values into the implemented local Elapsed time counter and synchronisation verifies whether the time message Command Elapsed Time and local Elapsed Time counter matches till the mapped SpaceWire Time-Code level (with a tolerance of previous value) and only modifies the local Elapsed Time if their is a mismatch. If the target is not implemented with jitter and mitigation unit then the synchronisation forces the target time (ET counter) with the time message received.

For example, the initiator can create time message exactly at 0x00000001 coarse time and 0x040000 fine time (32 bit coarse time and 24 bit fine time, mapping value of 6 i.e. 64 SpaceWire Time-Codes per second, time message is generated at 0b000001 SpaceWire Time-Code), the value in the time message to be sent to the target can be coarse time 0x00000002 and 0x040000 fine time, (32 bit coarse time and 24 bit fine time, mapping value of 6, time message is qualified at the next reception of 0b000001 SpaceWire Time-Code, i.e. after a second). Both SPWTC in Control registers available in the initiator and target can be 0b000001 for this example. The time is synchronized after a second in this example. Depending on the frequency of SpaceWire Time-Codes and data link rate several different combination of ways to achieve time synchronisation is possible.

83.3.9 Latency measurement using Time-Stamps

The incoming and outgoing SpaceWire Distributed Interrupts are time stamped in initiator and target. The initiator calculates latency based on these time stamp values. The time stamped values in target are accessed from initiator through RMAP. The Latency Enable LE bit in Configuration 0 register must be enabled between the two nodes in the SpaceWire network for which the latency is to be calculated. The core supports 32 distributed interrupts and acknowledgment (Interrupt and acknowledgment numbers 0 to 31) or 64 distributed interrupts. The distributed interrupt transmission from initiator (which is the origin for latency calculation) is controlled by a mask register STM available in Configuration 3 register and SpaceWire time code register TSTC available in Time-Stamp SpaceWire Time-Code and Preamble Field Tx register, these registers specifies how often and at which time code distributed interrupt is transmitted and time stamping is performed.

The time stamping can be performed in two methods (only Interrupts or Interrupts and Acknowledgment), the DI bit in Configuration 3 register of SPWTDP component in target should be configured to specify which type of method is used. If only distributed interrupts (no acknowledgment) are used then DI bit should be 0. The transmitted and received distributed interrupts INTX and INRX in the Configuration 0 registers of both initiator and target must be configured with the interrupt number which will be used for the latency measurement. For example if the INTX in initiator Configuration 0 is configured with 0b00100 then the target INRX should be configured with the same value. Similarly if the INTX in target Configuration 0 is configured to be 0b00101 then the initiator INRX should be configured with the same value. Initially initiator sends a distributed interrupt when the conditions are matched (STM and TSTC registers match) and when the target received this distributed interrupt it will send another interrupt which will be received by the initiator. At each end transmission and reception is time stamped (current local time is stored in Time Stamp registers) and interrupt transmitted is INTX and received interrupt is checked whether it received INRX.

If both distributed interrupts and acknowledgment method is to be used then DI bit should be 1. The transmitted and received distributed interrupts INTX and INRX in the Configuration 0 registers of both initiator and target can have the same interrupt number (the acknowledgment number for a particular interrupt will be same as interrupt number). Similar to the previous method at each end transmission and reception is time stamped which will be used for latency calculations.

The Latency calculation can be started in initiator based on DIR (distributed interrupt received) interrupt available in Interrupt Status register (the interrupt should be enabled in the Interrupt Enable register). The latency is calculated form the time stamp registers based on the equation explained below

Latency = ((initiator time stamp Rx - initiator time stamp Tx) - (target time stamp Tx - target time stamp Rx)) /2

By calculating the Latency value repeatedly (at least for about 128 times, more number of times provides increased accuracy) and taking an average of it will provide the final latency value. The initiator should transfer the latency correction information to the Latency Field registers in the target by means of RMAP transfer. When the latency values are written it will be adjusted to local time in the target and the LC bit in Status 0 register is enabled (set to '1'), this status register can be disabled by writing '1' into the corresponding field.

83.3.10 Mitigation of jitter and drift

The Jitter and drift mitigation is performed in target when Jitter Enable JE and Mitigation Enable ME bit in Configuration 0 register is enabled. The process will only start when the target is initialized (the target local ET counter should have initialized through a time message from initiator and INSYNC bit in Status 0 register is enabled). The SPWTDP must be implemented with a mitigation unit for jitter and drift mitigation (the MA bit in Status 0 register specify the availability of this service).

The mitigation unit for jitter and drift is enabled with the mitigation VHDL generic.

83.3.11 External Datation

The external signals latch and save are used to provide external datation services. The Elapsed Time is continuously latched when the latch input signal goes high, the corresponding External datation mask register must be enabled for that particular signal. When save input goes high the latched value will remain same (at when the previous latch condition met) and all the mask bit previously enabled will be cleared. The EDS bit in Status Register 0 will go high when the latch and save condition matches and cleared when the latched elapsed time is read. The purpose of this status register is to ensure that all the implemented coarse and fine time are read. Reading the lowest implemented fine time makes the status register to go low. An output pulse is also produced when conditions for external datation is met. The pulse is driven for one system clock period on the occurrence of external save condition.

If a simpler version of latching the time is needed based on a signal going high at any instance then the latch and save signals can be provided with the same input.

There are four External datation services implemented and each of them has its own mask EDMx, status EDS and time EDxETx registers. All the four External datation services are based on the input latch and save signal vectors. The external datation pulse vector consist of four outputs corresponding to each of the external datation services.

83.3.12 Pulses

The core provides eight external outputs used for clock pulse distribution. The timing of each pulse output is individually derived from the Elapsed Time counter. It is possible to program for each pulse output individually the following parameters:

- periodicity pulse
- width of pulse
- polarity of pulse
- enable/disable pulse generation (reset status is disabled)

The pulse has two parts, the active and the inactive part. The active part always starts the pulse, followed by the inactive part. The polarity or logical level of the active part is programmable. The inactive part takes the logical inversion of the active pulse, and is the default output from the generator when the pulse is not issued or the overall generation is disabled.

The periodicity of the pulse corresponds to one of the ET bits that can be selected in the range 27 to 2-8 seconds, providing a range from 128 seconds to 3,91 ms, i.e. 0,0078 to 256 Hz frequency. See register definition for details.

The width of the active part of the pulse corresponds to one of the ET bits that can be selected in the range 26 to 2-9 seconds, providing a range from 64 seconds to 1,95 ms. See register definition for details.

It is possible to generate a pulse that has a duty cycle of 50%. It is also possible to generate a pulse for which the active part is as short as 2^{-9} seconds, and its period is as high as 27 seconds. The effective duty cycle can be as low as $2^{-9}/27$ for the longest period, up to 50% for the shortest period of 2-8 seconds = 256 Hz. The duty cycle choice becomes more restricted as the frequency increases. Note that it is only possible to reduce the duty cycle in one direction: 50%/50%, 25%/75%... 1%/99%. The active part of the pulse can thus never be more than 50% of the cycle. It should be noted that the active pulse width must be at most 50% of the pulse period.

The pulse outputs are guaranteed to be spike free. If a pulse output is disabled by means of writing to the corresponding register (PDRx) (i.e. writing a zero to the Pulse Enable bit (PE)), the pulse output will be immediately driven to the inversion of the Pulse Level bit (PL), which corresponds to the level of the inactive part of the pulse. It is thus possible to modify immediately the pulse output by disabling it using the PE bit and then changing the PL bit, since the output will always drive the inversion of the PL bit while disabled.

An ongoing pulse output will be immediately disabled (the pulse output will be immediately driven to the inversion of the Pulse Level bit) if any external modification of the ET counter is triggered (for both master and slave) due to initialization/synchronisation or set using APB registers.

83.3.13 Set Elapsed Time using external input

The ET counter can be set using an external enable signal (configurable rising or falling edge, see register SP in Configuration 0 register). To set the ET counter the SE bit in configuration register must be enabled, the value to be loaded into the ET counter must be written into the Command Elapsed Time registers. The ARM bit field in the Status 0 register will set itself to ‘1’ when the first Command Elapsed Time register is written. After the occurrence of the external enable signal the value will be loaded into the ET counter and the ARM bit field in the Status 0 register will set itself to ‘0’. An interrupt can also be generated when the ET counter is loaded, the corresponding interrupt (Set ET External Interrupt Enable) must be enabled.

83.3.14 Multiple Port

It is possible to transmit or receive SpaceWire Time-Codes to multiple SpaceWire links. The Multiple port enable bit in the Configuration 0 register must be enabled, using the Inport and Outport fields in the Configuration 3 register the corresponding inputs and outputs can be enabled or disabled.

At every system clock (if the system is configured as target) all the enabled ports are monitored for a SpaceWire Time-Code input. If received the SpaceWire Time-Code is compared with the previously received Time-code. If the newly received Time-code is same as the previously received then no modification is performed other than updating the Received port RP register in Status 0. If an incremented Time-code is received then the corresponding Time-code is used for further processing (time qualification etc.). In all other case (Time-code - X or Time-code + X) the received Time code is stored (in previously received Time-code internal register, used for processing the next time-code) and an interrupt is generated if corresponding interrupt (Non consecutive SpaceWire Time-code Interrupt) bit is enabled. For all the Time-Code received the corresponding port in which it is received is updated at Received port RP register in Status 0.

The Inport and Outport fields also control the reception and transmission of Distributed interrupts. While performing latency calculation only one port must be enabled, since distributed interrupts cannot be filtered out based on increments like in SpaceWire Time-Codes.

Multiple port transmission can also be disabled and only one port can be used for transmission or reception using SEL register in Configuration 0.

83.3.15 Synchronisation of target using SpaceWire Time-Codes

It is possible to synchronise the target only using SpaceWire Time-Codes. A master sending SpaceWire Time-Codes (using its Elapsed time counter) at regular interval can synchronise the Elapsed time in the target. The frequency of Time-code transmission in the master and the frequency of (when to expect a) Time-code in the target must match, this can be achieved by setting the Mapping fields in the Configuration 0 register. The incoming SpaceWire Time-Codes and the Time-code position mapped in the target Elapsed Time is compared, if they match the bits available after the compared bits are made zero, if the local time map is less than one (External Time-code arrived early) then the bits available after the compared bits are made zero and the other part (including the mapped part) is incremented by one. If the above two cases occurred then the target time is in sync with the master time and the Insync bit in Status 0 register is enabled. If the incoming Time-code is in out of order (Non consecutive) then the synchronisation is stopped, the Insync bit in Status 0 register is disabled (but the local time keeps running) and an interrupt is generated if corresponding interrupt (Non consecutive SpaceWire Time-code Interrupt) bit is enabled.

83.4 Data formats

All Elapsed Time (ET) information is compliant with the CCSDS Unsegmented Code defined in [CCSDS] and repeated hereafter.

83.4.1 Numbering and naming conventions

Convention according to the CCSDS recommendations, applying to time structures:

- The most significant bit of an array is located to the left, carrying index number zero.
- An octet comprises eight bits.

*Table 1552.*CCSDS n-bit field definition

CCSDS n-bit field		
most significant		least significant
0	1 to n-2	n-1

Convention according to AMBA specification:

- The least significant bit of an array is located to the right, carrying index number zero.
- Big-endian support.

*Table 1553.*AMBA n-bit field definition

AMBA n-bit field		
most significant		least significant
n-1	n-2 down to 1	0

83.5 Reference documents

[CCSDS] Time Code Formats, CCSDS 301.0-B-4, www.CCSDS.org

[SPW] Space engineering: SpaceWire - Links, nodes, routers and networks, ECSS-E-ST-50-12C

[RMAP] Space engineering: SpaceWire - Remote memory access protocol, ECSS-E-ST-50-52C

[SPWCUC] High Accuracy Time Synchronization over SpaceWire Networks

83.6 Registers

The core is programmed through registers mapped into AMBA APB address space.

*Table 1554.*Registers

APB address offset	Register
0x000-0x00F	Configuration Field
0x000	Configuration 0
0x004	Configuration 1
0x008	Configuration 2
0x00C	Configuration 3
0x010 - 0x01F	Status Field
0x010	Status 0
0x014	Status 1
0x018	RESERVED
0x01C	RESERVED
0x020 - 0x03F	Command Field
0x020	Control
0x024	Command Elapsed Time 0
0x028	Command Elapsed Time 1
0x02C	Command Elapsed Time 2
0x030	Command Elapsed Time 3
0x034	Command Elapsed Time 4
0x038	RESERVED
0x03C	RESERVED
0x040 - 0x05F	Datation Field
0x040	Datation Preamble Field
0x044	Datation Elapsed Time 0
0x048	Datation Elapsed Time 1
0x04C	Datation Elapsed Time 2
0x050	Datation Elapsed Time 3
0x054	Datation Elapsed Time 4
0x058	RESERVED
0x05C	RESERVED
0x060 - 0x09F	Time-Stamp Field
0x060	Time-Stamp Preamble Field Rx
0x064	Time-Stamp Elapsed Time 0 Rx
0x068	Time-Stamp Elapsed Time 1 Rx
0x06C	Time-Stamp Elapsed Time 2 Rx
0x070	Time-Stamp Elapsed Time 3 Rx
0x074	Time-Stamp Elapsed Time 4 Rx
0x078	RESERVED
0x07C	RESERVED
0x080	Time-Stamp SpaceWire Time-Code and Preamble Field Tx
0x084	Time-Stamp Elapsed Time 0 Tx

APB address offset	Register
0x088	Time-Stamp Elapsed Time 1 Tx
0x08C	Time-Stamp Elapsed Time 2 Tx
0x090	Time-Stamp Elapsed Time 3 Tx
0X094	Time-Stamp Elapsed Time 4 Tx
0x098	RESERVED
0x09C	RESERVED
0x0A0-0x0BF	Latency Field
0x0A0	Latency Preamble Field
0x0A4	Latency Elapsed Time 0
0x0A8	Latency Elapsed Time 1
0x0AC	Latency Elapsed Time 2
0x0B0	Latency Elapsed Time 3
0x0B4	Latency Elapsed Time 4
0x0B8	RESERVED
0x0BC	RESERVED
0x0C0	Interrupt Enable
0x0C4	Interrupt Status
0x0C8	Delay Count
0x0CC	Disable Sync
0x0D0-0x0FF	RESERVED
0x100-0x18F	External Datation Field
0x100	External Datation 0 Mask
0x104	External Datation 1 Mask
0x108	External Datation 2 Mask
0x10C	External Datation 3 Mask
0x110-0x12F	External Datation 0 Time
0x110	External Datation 0 Preamble Field
0x114	External Datation 0 Elapsed Time 0
0x118	External Datation 0 Elapsed Time 1
0x11C	External Datation 0 Elapsed Time 2
0x120	External Datation 0 Elapsed Time 3
0x124	External Datation 0 Elapsed Time 4
0x128	RESERVED
0x12C	RESERVED
0x130-0x14F	External Datation 1 Time
0x150-0x16F	External Datation 2 Time
0x170-0x18F	External Datation 3 Time
0x190-0x19F	RESERVED
0x1A0-1BC	Pulse Definition Register 0 to 7
0x1C0-0x1FF	RESERVED

83.6.1 Configuration 0

*Table 1555.*0x000 - CONF0 - Configuration 0

31 25	24	23 21	20	19	18	17	16	15	14 13	12 8	7	6	5 4	3	2	1	0
RESERVED	JE	RES	ST	EP	ET	SP	SE	LE	AE	RES	MAPPING	TD	MU	SEL	ME	RE	TE	RS
0	0	0	0	1	0	1	0	0	0	0	*	0	0	0	0	0	0	0
r	rw	r	rw	rw	rw	rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw	rw	rw

31: 25	RESERVED	
24:	JE	Jitter Correction Enable (only for target) The jitter correction process in target will start when this bit is enabled. (Mitigation Enable bit should also be enabled). Reset value: ‘0’. (valid only when Mitigation unit available)
23: 20	RESERVED	
21:	ST	Synchronisation using SpaceWire Time-Code Enable (only for target) Reset value: ‘0’.
20:	EP	External ET Increment Polarity. To select the rising or falling edge of the external enable signal to increment the Elapsed time. Value ‘1’ Rising edge. Value ‘0’ Falling edge. Reset value: ‘1’.
19:	ET	External ET Increment Enable. Enable to increment the Elapsed Time based on external signal. When disabled the internal frequency synthesizer is used to increment the Elapsed Time counter. Reset value: ‘0’.
18:	SP	Set ET External Polarity. To select the rising or falling edge of the external enable signal to load the Elapsed Time with the contents of the command field register. Reset value: ‘1’.
17:	SE	Set ET External Enable. Based on the external enable signal load the Elapsed Time with the contents of the command field register. Reset value: ‘0’.
16:	LE	Latency Enable. To calculate latency between an initiator and target this bit must be enabled in both of them. Reset value: ‘0’.
15:	AE	AMBA Interrupt Enable The interrupts (explained in interrupt registers) in this core will generate an AMBA interrupt only when this bit is enabled. Reset value: ‘0’
14 13	RESERVED	
12: 8	MAPPING	Defines mapping of SpaceWire Time-Codes versus CCSDS Time-code. Value 0b00000 will send SpaceWire Time-Codes every Second, Value 0b00001 will send SpaceWire Time-Codes every 0.5 Second, Value 0b00010 will send SpaceWire Time-Codes every 0.25 Second, Value 0b00011 will send SpaceWire Time-Codes every 0.125 Second Maximum value it can take is 0b11111 but this value cannot be more than the number of bits implemented as fine time. Reset value: Implementation dependent
7:	TD	Enable TDP when set. Reset value: ‘0’.
6:	MU	Multiple Port Enable. Reset value ‘0’.
5: 4	SEL	Select for SpaceWire Time-Codes and Distributed Interrupt transmission and reception, one of 0 through 3. Can be used only when Multiple Port MU is disabled. Reset value: 0b00
3:	ME	Mitigation Enable (only for target) The drift correction process in target will start when this bit is enabled. Reset value: ‘0’.(valid only when Mitigation unit available)
2:	RE	Receiver Enable (only for target) Reset value: ‘0’.
1	TE	Transmit Enable (only for initiator) Reset value: ‘0’.
0	RS	Reset core. Makes complete reset when enabled. Reset value: ‘0’.

83.6.2 Configuration 1

*Table 1556.*0x004 - CONF1 - Configuration 1

31 30 29	0
R	FSINC
0	0
r	rw

31: 30 RESERVED

29: 0 FSINC

Increment value of the Frequency Synthesizer which is added to the counter every system clock cycle. It defines the frequency of the synthesized reference time.

Refer the spreadsheet provided along with this document to obtain this value.

Reset value: Implementation dependent

All implemented registers are writable and readable.

83.6.3 Configuration 2

*Table 1557.*0x008 - CONF2 - Configuration 2

31	8	7	0
CV		ETINC	
*		*	
rw		rw	

31: 8 CV Compensation Value

Value added to FSINC for variations of drift of the target clock.(only for target)

Refer the spreadsheet provided along with this document to obtain this value.

This value also depends on the MAPPING value in configuration 0 register. Specify the needed MAPPING value in the spreadsheet while calculating this value.

Reset value: Implementation Dependent

(valid only when Mitigation unit available)

7: 0 ETINC Value of the Elapsed Time counter is to be incremented each time when the Frequency Synthesizer wraps around.

Refer the spreadsheet provided along with this document to obtain this value.

Reset value: Implementation dependent

83.6.4 Configuration 3

*Table 1558.*0x00C - CONF3 - Configuration 3

31 28	27 24	23 22	21 16	15 14	13	12	11	10	9 5	4 0
OUTPORT	INPORT	RESERVED	STM	RESERVED	DI64R	DI64T	DI64	DI	INRX	INTX
0	0	0	0	0	0	0	0	0	0	0
rw	rw	r	rw	r	rw	rw	rw	rw	rw	rw

31: 28	OUTPORT	Enable the corresponding output ports. The multiple port enable bit MU in Configuration 0 register must be enabled. Reset value: ‘0’
27: 24	INPORT	Enable the corresponding input ports. The multiple port enable bit MU in Configuration 0 register must be enabled. Reset value: ‘0’
23: 22	RESERVED	
21: 16	STM	SpaceWire Time-Code Mask Mask For TSTC register available at Time-Stamp SpaceWire Time-Code and Preamble Field Tx register. Value all bits zero will send Distributed interrupts at all SpaceWire Time-Codes irrespective of any values in TSTC register. Value all ones will send Distributed interrupts at complete match of SpaceWire Time-Code with TSTC register. (only for initiator)
15: 14	RESERVED	
13:	DI64R	The MSb for received Distributed Interrupt when interrupt numbers 32 to 63 is used. Possible only for DI = ‘0’ (only interrupt mode) and DI64 is enabled. Reset value: ‘0’
12:	DI64T	The MSb for transmitted Distributed Interrupt when interrupt numbers 32 to 63 is used. Possible only for DI = ‘0’ (only interrupt mode) and DI64 is enabled. Reset value: ‘0’
11:	DI64	Enable Distributed Interrupts 64, when set all 64 Distributed interrupt numbers can be used for latency calculation. Possible only for DI = ‘0’ (only interrupt mode). Reset value: ‘0’
10:	DI	Distributed Interrupt method, when set interrupt and acknowledge mode else only interrupt mode. (only for target) Reset value: ‘0’
9: 5	INRX	Interrupt Received.(Distributed) The distributed interrupt number received by initiator or target. Reset value: 0b000000
4: 0	INTX	Interrupt Transmitted.(Distributed) The distributed interrupt number transmitted by initiator or target. Reset value: 0b000000

83.6.5 Status Register 0

*Table 1559.*0x010 - STAT0 - Status Register 0

31	30 29	28	27 24	23	22 16	15 14	13 8	7 4	3	2	1	0
MA	RP	R	EDS	R	FW	RES	CW	RES	ARM	LC	TCQ	INSYNC
0	0	0	0	0	0	0	0	0	0	0	0	0
r	r	r	r	r	r	r	r	r	r	wc	r	r

31:	MA	Mitigation unit available 1 Drift and Jitter mitigation unit available in target 0 Not available (only for target)
30: 29	Received Port	When multiple ports receive SpaceWire time-codes, this register specify on which port the SpaceWire Time-Code is received recently.
28:	RESERVED	
27: 24	EDS	External Datation Status 24: External Datation 0 Status bit 25: External Datation 1 Status bit 26: External Datation 2 Status bit 27: External Datation 3 Status bit When conditions matched for external datation this bit will go high. This bit will go low when all the implemented time values are read.
23	RESERVED	
22: 16	FW	Fine width of command CCSDS Time Code received. Calculated from Preamble field of Command Register.
15: 14	RESERVED	
13: 8	CW	Coarse width of command CCSDS Time Code received, calculated from Preamble field of Command Register.
7: 4	RESERVED	
3:	ARM	This field is enabled when the command field register is written with the value to be loaded into the Elapsed time. The Set ET External Enable SE bit in the Configuration 1 must be enabled. When an external enable signal occurred and the command field register contents are loaded into the Elapsed time then this bit will get disabled.
2	LC	Latency Corrected (only for target) This register can be cleared by writing value '1' to this field.
1	TCQ	Time message is qualified by SpaceWire Time-Codes.
0	INSYNC	In Sync at Time code level, enabled when time values are Initialized or Synchronized

83.6.6 Status Register 1

*Table 1560.*0x014 - STAT1 - Status Register 1

31 30 29	0
R	IV
0	*
r	r

31: 30	RESERVED	
29: 0	IV	Increment Variation. The variation in FSINC while achieving the time synchronisation (only for target) Reset value: Implementation dependent (valid only when Mitigation unit available)

83.6.7 Control

*Table 1561.*0x20 - CTRL - Control

31	30	29 24	23 16	15 0
NC	IS	R	SPWTC	CPF
0	0	0	0	0
rw	rw	r	rw	rw

31:	NC	New Command
30:	IS	Init or Sync 1 Initialization of received time message 0 Synchronisation of received time message (only for target)
29: 24	RESERVED	
23: 16	SPWTC	SpaceWire Time-Code value used for initialization and synchronisation In initiator the SpaceWire Time-Codes generated internally using the local ET counter matches this register a Time Message TM interrupt will be generated which is used to send Time message over the SpaceWire network. In target this register should match the received SpaceWire Time-Code for time qualification.
15: 0	CPF	Command Preamble Field. The number of coarse and fine time available in Command Elapsed Time registers should be mentioned in this field. Based on this preamble field the target will initialize or synchronise the local ET counter.(only for target)

83.6.8 Command Elapsed Time 0

*Table 1562.*0x024 - CET0 - Command Elapsed Time 0

31 0
CET0
0
rw

31: 0	CET0	Command Elapsed Time 0 Initialize or Synchronise local ET counter value (0 to 31).

83.6.9 Command Elapsed Time 1

*Table 1563.*0x028 - CET1 - Command Elapsed Time 1

31 0
CET1
0
rw

31: 0	CET1	Command Elapsed Time 1 Initialize or Synchronise local ET counter value (32 to 63)

83.6.10 Command Elapsed Time 2

Table 1564. 0x02C - CET2 - Command Elapsed Time 2

31	0
CET2	
0	
rw	

31: 0	CET2	Command Elapsed Time 2 Initialize or Synchronise local ET counter value (64 to 95).

83.6.11 Command Elapsed Time 3

Table 1565. 0x030 - CET3 - Command Elapsed Time 3

31	0
CET3	
0	
rw	

31: 0	CET3	Command Elapsed Time 3 Initialize or Synchronise local ET counter value (96 to 127).

83.6.12 Command Elapsed Time 4

Table 1566. 0x034 - CET4 - Command Elapsed Time 4

31 24	23 0
CET4	RESERVED
0	0
rw	r

31: 24	CET4	Command Elapsed Time 4 Initialize or Synchronise local ET counter value (128 to 135).
23: 0	RESERVED	

83.6.13 Datation Preamble Field

Table 1567. 0x040 - DPF - Datation Preamble Field

31 16	15 0
RESERVED	DPF
0	0x2F00
r	r

31: 16	RESERVED	
15: 0	DPF	Datation Preamble Field The number of coarse and fine time implemented can be obtained from this Preamble Field.

83.6.14 Datation Elapsed Time 0

Table 1568. 0x044 - DET0 - Datation Elapsed Time 0

31	0
DET0	
0	
r	

31: 0	DET0	Datation Elapsed Time 0 CCSDS Time Code value (0 to 31) of local ET counter value.

83.6.15 Datation Elapsed Time 1

Table 1569. 0x048 - DET1 - Datation Elapsed Time 1

31	0
DET1	
0	
r	

31: 0	DET1	Datation Elapsed Time 1 CCSDS Time Code value (32 to 63) of local ET counter value.

83.6.16 Datation Elapsed Time 2

Table 1570. 0x04C - DET2 - Datation Elapsed Time 2

31	0
DET2	
0	
r	

31: 0	DET2	Datation Elapsed Time 2 CCSDS Time Code value (64 to 95) of local ET counter value.

83.6.17 Datation Elapsed Time 3

Table 1571. 0x050 - DET3 - Datation Elapsed Time 3

31	0
DET3	
0	
r	

31: 0	DET3	Datation Elapsed Time 3 CCSDS Time Code value (96 to 127) of local ET counter value.

83.6.18 Datation Elapsed Time 4

Table 1572. 0x054 - DET4 - Datation Elapsed Time 4

31 24	23 0
DET4	RESERVED
0	0
r	r

31: 24 DET4 Datation Elapsed Time 4

CCSDS Time Code value (128 to 135) of local ET counter value.

23: 0 RESERVED

83.6.19 Time-Stamp Preamble Field Rx

Table 1573. 0x060 - TRPFRx - Time-Stamp Preamble Field Rx

31 16	15 0
RESERVED	TRPF
0	0x2F00
r	r

31: 16 RESERVED

15: 0 TRPF Time stamp Preamble Field

The number of coarse and fine time implemented can be obtained from this Preamble Field.

83.6.20 Time Stamp Elapsed Time 0 Rx

Table 1574. 0x064 - TR0 - Time Stamp Elapsed Time 0 Rx

31 0
TR0
0
r

31: 0 TR0 Time stamped local ET value (0 To 31) when distributed interrupt received.

83.6.21 Time Stamp Elapsed Time 1 Rx

Table 1575. 0x068 - TR1 - Time Stamp Elapsed Time 1 Rx

31 0
TR1
0
r

31: 0 TR1 Time stamped local ET value (32 to 63) when distributed interrupt received.

83.6.22 Time Stamp Elapsed Time 2 Rx

Table 1576. 0x06C - TR2 - Time Stamp Elapsed Time 2 Rx

31 0
TR2
0
r

31: 0 TR2 Time stamped local ET value (64 to 95) when distributed interrupt received.

83.6.23 Time Stamp Elapsed Time 3 Rx

Table 1577. 0x070 - TR3 - Time Stamp Elapsed Time 3 Rx

31	0
TR3	
0	
r	

31: 0 TR3 Time stamped local ET value (96 to 127) when distributed interrupt received.

83.6.24 Time Stamp Elapsed Time 4 Rx

Table 1578. 0x074 - TR4 - Time Stamp Elapsed Time 4 Rx

31 24	23 0
TR4	RESERVED
0	0
r	r

31: 24 TR4 Time stamped local ET value (128 to 135) when distributed interrupt received.

23: 0 RESERVED

83.6.25 Time-Stamp SpaceWire Time-Code and Preamble Field Tx

Table 1579. 0x080 - TTPFTx - Time-Stamp SpaceWire Time-Code and Preamble Field Tx

31 24	23 16	15 0
TSTC	RESERVED	TTPF
0	0	0x2800
rw	r	r

31: 24 TSTC Time stamp time code

Time stamp on this time-code value, used for time stamping when this register matched with SpaceWire Time-Codes. The mask for this matching is available in configuration register 3. (only for initiator)

23: 16 RESERVED

15: 0 TTPF Time stamp Preamble Field

The number of coarse and fine time implemented can be obtained from this Preamble Field.

83.6.26 Time Stamp Elapsed Time 0 Tx

Table 1580. 0x084 - TT0 - Time Stamp Elapsed Time 0 Tx

31	0
TT0	
0	
r	

31: 0 TT0 Time stamped local ET value (0 to 31) when distributed interrupt transmitted.

83.6.27 Time Stamp Elapsed Time 1 Tx

Table 1581. 0x088 - TT1 - Time Stamp Elapsed Time 1 Tx

31	0
TT1	
0	
r	

31: 0 TT1 Time stamped local ET value (32 to 63) when distributed interrupt transmitted.

83.6.28 Time Stamp Elapsed Time 2 Tx

Table 1582. 0x08C - TT2 - Time Stamp Elapsed Time 2 Tx

31	0
TT2	
0	
r	

31: 0 TT2 Time stamped local ET value (64 to 95) when distributed interrupt transmitted.

83.6.29 Time Stamp Elapsed Time 3 Tx

Table 1583. 0x090 - TT3 - Time Stamp Elapsed Time 3 Tx

31	0
TT3	
0	
r	

31: 0 TT3 Time stamped local ET value (96 to 127) when distributed interrupt transmitted.

83.6.30 Time Stamp Elapsed Time 4 Tx

Table 1584. 0x094 - TT4 - Time Stamp Elapsed Time 4 Tx

31 24	23 0
TTT0	RESERVED
0	0
r	r

31: 24 TT4 Time stamped local ET value (128 to 135) when distributed interrupt transmitted.

23: 0 RESERVED

83.6.31 Latency Preamble Field

Table 1585. 0x0A0 - LPF - Latency Preamble Field

31 16	15 0
RESERVED	LPF
0	0x2F00
r	r

31: 16 RESERVED

15: 0 LPF Latency Preamble Field

The number of coarse and fine time implemented can be obtained from this Preamble Field. (only for target)

83.6.32 Latency Elapsed Time 0

Table 1586. 0x0A4 - LE0 - Latency Elapsed Time 0

31	0
LE0	
0	
rw	

31: 0 LE0 Latency Value (0 to 31) written by initiator. (only for target)

83.6.33 Latency Elapsed Time 1

Table 1587. 0x0A8 - LE1 - Latency Elapsed Time 1

31	0
LE1	
0	
rw	

31: 0 LE1 Latency Value (32 to 63) written by initiator. (only for target)

83.6.34 Latency Elapsed Time 2

Table 1588. 0x0AC - LE2 - Latency Elapsed Time 2

31	0
LE2	
0	
rw	

31: 0 LE2 Latency Value (64 to 95) written by initiator. (only for target)

83.6.35 Latency Elapsed Time 3

Table 1589. 0x0B0 - LE3 - Latency Elapsed Time 3

31	0
LE3	
0	
rw	

31: 0 LE3 Latency Value (96 to 127) written by initiator. (only for target)

83.6.36 Latency Elapsed Time 4

Table 1590. 0x0B4 - LE4 - Latency Elapsed Time 4

31 24	23 0
LE4	RESERVED
0	0
rw	r

31: 24 LE4 Latency Value (128 to 135) written by initiator. (only for target)

23: 0 RESERVED

83.6.37 Interrupt Enable

*Table 1591.*0x0C0 - IE - Interrupt Enable

31 20	19	18 11	10	9	8	7	6	5	4	3	2	1	0
RESERVED	NCTCE	PE(7-0)	SETE	EDIE3	EDIE2	EDIE1	EDIE0	DITE	DIRE	TTE	TME	TRE	SE
0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31: 20	RESERVED	
19:	NCTCE	Non consecutive SpaceWire Time-Code received Interrupt Enable
18: 11	PE(7-0)	Pulse Interrupt Enable
10	SETE	Set ET External Interrupt Enable
9	EDIE3	External Datation 3 Interrupt Enable
8	EDIE2	External Datation 2 Interrupt Enable
7	EDIE1	External Datation 1 Interrupt Enable
6	EDIE0	External Datation 0 Interrupt Enable
5	DITE	Distributed Interrupt Transmitted Interrupt Enable
4	DIRE	Distributed Interrupt Received Interrupt Enable
3	TTE	SpaceWire Time-Code Transmitted Interrupt Enable (only for initiator)
2	TME	Time Message transmit Interrupt Enable (only for initiator)
1	TRE	SpaceWire Time-Code Received Interrupt Enable (only for target)
0	SE	Sync Interrupt Enable (only for target)

83.6.38 Interrupt Status

*Table 1592.*0x0C4 - IS - Interrupt Status

31 20	19	18 11	10	9	8	7	6	5	4	3	2	1	0
RESERVED	NCTC	P(7-0)	SET	EDI3	EDI2	EDI1	EDI0	DIT	DIR	TT	TM	TR	S
0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	wc	wc	wc	wc	wc	wc	wc	wc	wc	wc	wc	wc	wc

31: 20	RESERVED	
19:	NCTC	Generated when Non consecutive SpaceWire Time-Code is received
18: 11	P(7-0)	Generated when an active Pulse is transmitted, at the start of active pulse an interrupt is generated
10	SET	Generated when Elapsed Time is loaded with contents of the Command Field register based on external enable signal.
9	EDI3	Generated when conditions for External Datation 3 is matched
8	EDI2	Generated when conditions for External Datation 2 is matched
7	EDI1	Generated when conditions for External Datation 1is matched
6	EDI0	Generated when conditions for External Datation 0 is matched
5	DIT	Generated when distributed interrupt is transmitted (Latency calculation should be enabled)
4	DIR	Generated when distributed interrupt is Received (Latency calculation should be enabled)
3	TT	Generated when SpaceWire Time-Codes is transmitted (only for initiator)
2	TM	Generated when the conditions for transmitting time message occurred, based on this time message should be transmitted from initiator (only for initiator)
1	TR	Generated when SpaceWire Time-Code is received (only for target)
0	S	Generated when the target is initialized or synchronized with initiator (only for target)

The interrupts are cleared by writing value 1 on respective bits.

83.6.39 Delay Count

Table 1593. 0x0C8 - DC - Delay Count

31 15	14 0
RESERVED	DC
0	0x7FFF
r	r

31: 15 RESERVED

14: 0 DC — Delay Count

Delay induced between SpaceWire Time-Codes and Distributed Interrupt transmission in system clock units. The delay introduced is the value in this register multiplied by the system clock.

(only for initiator)

83.6.40 Disable Sync

Table 1594. 0x0CC - DS - Disable Sync

31	30 24	23 0
EN	RESERVED	CD
0	0	0xFFFFFF
rw	r	rw

31: EN — Enable

30: 24 RESERVED

23: 0 CD — Configurable delay to capture missing SpaceWire Time-Code (only for target)

The INSYNC bit in the Status 0 register will disable itself when an expected SpaceWire Time-Code is not arrived after the delay mentioned in this register. The delay corresponds to the fine time of Elapsed Time counter and should not overlap with the MAPPING register. Any Overlapping register must also be set to Zero.

83.6.41 External Datation 0 Mask

Table 1595. 0x100 - EDM0 - External Datation 0 Mask

31 0
EDM0
0
rw

31: 0 EDM0 — External datation can be enabled by writing ‘1’ into the bit for that corresponding external input. When conditions are matched the Elapsed Time will be latched.

The latched values are available at External Datation 0 Time Register.

All the mask bits will go low after any one of the conditions with respect to the enabled mask bits.are matched.

83.6.42 External Datation 0 Preamble Field

Table 1596. 0x110 - EDPF0 - External Datation 0 Preamble Field

31 16	15 0
RESERVED	EDPF0
0	0x2F00
r	r

Table 1596. 0x110 - EDPF0 - External Datation 0 Preamble Field

31: 16	RESERVED	
15: 0	EDPF0	External Datation Preamble Field The number of coarse and fine time implemented can be obtained from this Preamble Field.

83.6.43 External Datation 0 Elapsed Time 0

*Table 1597.*0x114 - ED0ET0 - External Datation 0 Elapsed Time 0

31	0
ED0ET0	
0	
r	

31: 0	ED0ET0	External Datation Elapsed Time 0 Latched CCSDS Time Code value (0 to 31) of local ET counter.

83.6.44 External Datation 0 Elapsed Time 1

*Table 1598.*0x118 - ED0ET1 - External Datation 0 Elapsed Time 1

31	0
ED0ET1	
0	
r	

31: 0	ED0ET1	External Datation Elapsed Time 1 Latched CCSDS Time Code value (32 to 63) of local ET counter.

83.6.45 External Datation 0 Elapsed Time 2

*Table 1599.*0x11C - ED0ET2 - External Datation 0 Elapsed Time 2

31	0
ED0ET2	
0	
r	

31: 0	ED0ET2	External Datation 0 Elapsed Time 2 Latched CCSDS Time Code value (64 to 95) of local ET counter.

83.6.46 External Datation 0 Elapsed Time 3

*Table 1600.*0x120 - ED0ER3 - External Datation 0 Elapsed Time 3

31	0
ED0ET3	
0	
r	

31: 0	ED0ET3	External Datation 0 Elapsed Time 3 Latched CCSDS Time Code value (96 to 127) of local ET counter.

83.6.47 External Datation 0 Elapsed Time 4

*Table 1601.*0x124 - ED0ET4 - External Datation 0 Elapsed Time 4

31 24	23 0
ED0ET4	RESERVED
0	0
r	r

*Table 1601.*0x124 - ED0ET4 - External Datation 0 Elapsed Time 4

31: 24	ED0ET4	External Datation 0 Elapsed Time 4 Latched CCSDS Time Code value (128 to 135) of local ET counter.
23: 0	RESERVED	

83.6.48 Pulse Definition Register 0 to 7

*Table 1602.*0x1A0-0x1BC - PDR0 to PDR7 - Pulse Definition Register 0 to 7

31 24	23 20	19 16	15 11	10	9 2	1	0
RESERVED	PP	PW	RESERVED	PL	RESERVED	PE	R
0	0	0	0	1	0	0	0
r	rw	rw	r	rw	r	rw	r

31: 24	RESERVED	
23: 20	PP	Pulse Period Value ‘0000’ = 2^{7} seconds Value ‘0001’ = 2^{6} seconds ... Value ‘1110’ = 2^{-7} seconds Value ‘1111’ = 2^{-8} seconds Period = $2^{(7-PP)}$ Frequency = $2^{-(7-PP)}$
19: 16	PW	Pulse Width Value ‘0000’ = 2^{6} seconds Value ‘0001’ = 2^{5} seconds ... Value ‘1110’ = 2^{-8} seconds Value ‘1111’ = 2^{-9} seconds Width = $2^{(6-PW)}$
15: 11	RESERVED	
10:	PL	Pulse Level Defines logical level of active part of pulse output. ‘0’ = Low, ‘1’ = High
9: 2	RESERVED	
1:	PE	Pulse Enable ‘0’ = disabled, ‘1’ = enabled
0:	RESERVED	

Note: The registers which are not mentioned either as only for initiator or target are used in both initiator and target.

The Definition of External Datation 1 Mask, External Datation 2 Mask and External Datation 3 Mask registers are exactly same as External Datation 0 Mask Register.

The Definition of External Datation 1 Time, External Datation 2 Time and External Datation 3 Time registers are exactly same as External Datation 0 Time Registers (i.e. External Datation 0 Preamble Field and External Datation 0 Elapsed Time 0,1,2,3,4).

83.7 Vendor and device identifiers

The module has vendor identifier 0x01 and device identifier 0x097. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

83.8 Implementation

83.8.1 Reset

The core does not change reset behavior depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

83.9 Configuration options

Table below shows the configuration options of the core (VHDL generics).

*Table 1603.*Configuration options

Generic	Description	Allowed range	Default
mitigation	Jitter and drift correction unit enable	0-1	0
tech	Select technology	0-NTECH	inferred
gCoarse	Number of CUC coarse bits	8-56	32
gFine	Number of CUC fine bits	0-80	24
gMaster	Initiator implementation enable	0-1	1
gSlave	Target implementation enable	0-1	1
gFrequency	Width of Frequency Synthesizer	2-30	30
gFSIncrement	Increment of FS counter	0-0x3FFFFFFF	360287970
gETIncrement	Increment of ET counter	0-255	0
gComp	Compensation value for jitter and drift variations	0-0xFFFFFF	461
gPField	P-Field	0-0xAF7C	0x2F00
gMapping	Initial mapping value	0-31	6
gExtDatation	Implementation of External Datation	0-1	1
gNoExtDat	Number of External Datation services	1-4	4
gPulses	Pulse support	0-1	1
gNoPulses	Number of Pulse generation services	1-8	8
gMulEN	Multiple ports Enable	0-1	1
gNoPorts	Number of ports	1-4	4
gSetET	Set ET support	0-1	1
gExtETInc	Increment Elapsed Time based on external input	0-1	1
gSpWSync	Support for Synchronisation using only SpaceWire Time-Codes	0-1	1
delay	Number of Delay Count bits need to be implemented.	2-15	9
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the core.	0 - NAHBIRQ-1	1

83.10 Signal descriptions

Table below shows the interface signals of the core (VHDL ports).

Table 1604. Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
GRTDPI	tickindone	Input	SpaceWire Time-Code input processed	High
	tickoutraw	Input	SpaceWire Time-Code/Distributed interrupt output event	High
	timeout	Input	SpaceWire Time-Code/Distributed interrupt output	-
	ext_dat_latch	Input	Vector to input external signals on which the Elapsed Time is latched. Used for External datation service.	-
	ext_dat_save	Input	Vector to input external signals on which the Elapsed Time is saved. Used for External datation service.	-
	setet	Input	Input to set ET from Register	-
	ext_et_inc	Input	Input to increment ET	-
GRTDPO	tickinraw	Output	SpaceWire Time-Code/Distributed interrupt input request	High
	timein	Output	SpaceWire Time-Code/Distributed interrupt input	-
	elapsedtime	Output	Elapsed time	-
	enable	Output	TDP enable, this signal reflects the content of the TD bit in the Configuration 0 register.	High
	ext_dat_pulse	Output	Produces an output pulse when conditions for external datation is met. The pulse is driven for one system clock period on the occurrence of external save condition. This vector consist of four outputs corresponding to each external datation services.	-
	pulses	Output	Pulse output	-
DIAG_CTICK	N/A	Output	This tick is generated when SpaceWire Time-Code is transmitted in initiator. This tick is generated when a diagnostic SpaceWire Time-Code is generated from targets fully corrected time.	High
DIAG_JTICK	N/A	Output	The incoming SpaceWire Time-Code tick, to visualize the jitter (only for target)	High

* see GRLIB IP Library User's Manual

The inputs setet and ext_et_inc are re-synchronized internally. All the other inputs does not have internal support to remove meta-stability resulting from clock domain crossings. If the inputs are driven from clock domains other than the clock provided for this core then the clock synchronisation conditions must be dealt externally for these inputs.

83.11 Signal definitions and reset values

The core has no external signals.

83.12 Timing

The core has no external timing.

83.13 Library dependencies

Table 1605 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1605.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	GRSPACE-WIRETDP	Signals, component	Component declarations, signals.
TMTC	SPACEWIRETDP	Signals, component	Component declarations, signals.

83.14 Instantiation

This example shows how the core can be instantiated.

```
library  ieee;
use      ieee.std_logic_1164.all;

library  grlib;
use      grlib.amba.all;
use      grlib.devices.all;
use      grlib.stdlib.all;

library  tmtc;
use      tmtc.grspacewiretdp.all;
use      tmtc.spacewiretdp.all;

entity grspwtdp_ex is

end entity grspwtdp_ex;

architecture rtl of grspwtdp_ex is

signal apbi            : apb_slv_in_type;
signal apbo            : apb_slv_out_vector := (others => apb_none);

signal spwi            : grspw_in_type;
signal spwo            : grspw_out_type;

signal tdpi            : grtdp_In_Type;
signal tdpo            : grtdp_Out_Type;
signal tdp_enable      : Std_ULogic;
signal tdp_elapsedtime : Std_Logic_Vector(0 to 135);
signal ext_dat_latch   : Std_Logic_Vector(31 downto 0);
signal ext_dat_save    : Std_Logic_Vector(31 downto 0);
signal tdp_elapsedtime : Std_Logic_Vector(0 to 135);
signal ext_dat_pulse   : Std_Logic_Vector(3 downto 0);
signal pulses          : Std_Logic_Vector(7 downto 0);
signal setet           : Std_ULogic;
signal ext_et_inc      : Std_ULogic;

signal diag_ctick      : Std_ULogic;
signal diag_jtick      : Std_ULogic;


begin

spw_time_0: grspwtdp
  generic map(
     mitigation         => 1,
     tech               => 0,
```

```
        gCoarse             => 32,
        gFine               => 24,
        gMaster             => 1,
        gSlave              => 1,
        gFrequency          => 30,
        gETIncrement        => 0,
        gFSIncrement        => 360287970, -- Default for 50 Mhz
        gComp               => 461,
        gPFField            => 16#2F00#,
        gMapping            => 6,
        delay               => 9,
        pindex              => 4,
        paddr               => 4,
        pmask               => 16#ffe#,
        pirq                => 1)
    port map(
        rstn                => rstn,
        clk                 => clkm,
        apbi                => apbi,
        apbo                => apbo(4),
        grtdpi              => tdpi,
        grtdpo              => tdpo,
        diag_ctick          => diag_ctick,
        diag_jtick          => diag_jtick);

  spwi.timein               <= tdpo.timein;
  spwi.tickinraw            <= tdpo.tickinraw(0);
  tdp_elapsedtime           <= tdpo.elapsedtime;
  tdp_enable                <= tdpo.enable;
  ext_dat_pulse             <= tdpo.ext_dat_pulse;
  pulses                    <= tdpo.pulses;

  tdpi.tickindone(0)        <= spwo.tickindone;
  tdpi.tickoutraw(0)        <= spwo.tickoutraw;
  tdpi.timeout(0)           <= spwo.timeout;
  tdpi.ext_dat_latch        <= ext_dat_latch;
  tdpi.ext_dat_save         <= ext_dat_save;
  tdpi.setet                <= setet;
  tdpi.ext_et_inc           <= ext_et_inc;
  end;
```

84 GRSRIO - Serial RapidIO endpoint with AHB or AXI4 bus master interface

84.1 Overview

The GRSRIO core implements the RapidIO Interconnect Specification Part 1 (I/O Logical Specification) and Part 2 (Message Passing Logical Specification) for data messages, doorbell messages, outbound maintenance messages and I/O operations, as defined in Rev. 2.1. This is done by means of a flexible and configurable DMA engine with a configurable bus master interface.

Additionally the core implements the MECS Time Synchronization Protocol according to Rev. 4.0.

The targeted SRIO endpoint is the SRIOIP-GEN2IP core by Integrated Device Technology. The choice of the SRIOIP-GEN2 end point has several implications on the SRIO:

- CAR and CSR registers defined in I/O and Message Passing Logical Specification are assumed to be already implemented in the end point
- Inbound maintenance request are assumed to be already handled by the end point

Two wrappers are available for the core, one with an AHB AMBA 2.0 Bus Master (grsrio_gen2ahb) to be readily compatible with Frontgrade Gaisler's GRLIB and one with an AXI4 (grsrio_gen2axi) to also support latest System-on-Chip (SoC) architectures.

The DMA engine serves several queues of descriptors placed in memory with two possible modes, selectable at compile time:

- Legacy mode: When *g_hyb_que* is set to “false”, descriptors for data messages and I/O operations are 8 words long and the ones for doorbell messages 4. In this configuration, each kind of operation (IO memory access, data message and doorbell message) have separated transmission queues, from 1 up to 32 each. In this configuration, descriptors for data messages and I/O operations are 8 words long, while ones for doorbell messages 4. In this configuration, the core has an interface and a behavior compatible with the GRSRIO Logical Layer IP core Version 1.
- Hybrid-queues mode: When *g_hyb_que* is set to “true”, there are just *g_no_hyb_tx* transmission queues, each capable of transmitting all kinds of logical operations. This allows a larger CPU offload and a lower number of queues required for each software entity. In this configuration, all TX descriptors are 8 words long. RX descriptors for data messages and I/O operations are 8 words long and the ones for doorbell messages 4.

A timestamp functionality is provided for each descriptor. When *g_mecs* is set to “true” the time value is taken by the internal MECS Timestamp Generator, while when *g_mecs* is set to false or the MECS functionality is disabled, the time value is taken from an external source.

The core provides a capability register to allow the software to know what functionalities the core instantiation supports and the number of queues instantiated.

84.2 Operation

84.2.1 Receiving Data messages

Inbound data messages are stored in one or more message reception queues. The desired number of reception queues can be set by VHDL generic *g_no_msg_rx* at compile time.

The inbound data messages are stored to a specific queue depending on their destination ID and mailbox number. The accepted destination ID of a queue can be defined in register RXMSG_DST_ID and the accepted mailbox number in the field RXMSG_CTRL.MBOX. Furthermore, ranges of destination IDs and mailbox numbers can be accepted by also setting the masking register RXMSG_DST_MSK and in the field RXMSG_CTRL.MBOXMSK. For instance, the user could set up five queues, the first four queues accepting messages addressed to mailbox 0 to 3 and the fifth queue accepting messages addressed to all the other ones (4-63). If several message reception queues are configured to accept

the same destination ID / mailbox number pair, the message queue with the lowest index number, that is currently not busy, will receive the message.

If a message is received, which is addressed to a mailbox number /destination ID combination that is not accepted by any message reception queue, the processing core automatically generates and transmits ERROR responses to the source node.

A single queue can receive only a multi-segment data message per time, but more queues can be set with the same destination ID or destination ID/Mailbox combination, effectively allowing up to 16 transfer between a couple of destination/source IDs.

Each reception queue is a circular buffer that is located in memory. Before activating the queue, the memory start address of the circular buffer must be set in register RXMSG_MADDR(_LSW) and RXMSG_MADDR_MSW (if CFG_AW is set to 64) and its depth in register RXMSG_CONF.CBD. Each queue can be configured to store between 2 and 65,536 message descriptors. Each descriptor has a size of 32 bytes and it is up to the software to reserve the necessary memory space for the whole circular buffer.

Each reception queue can be configured to trigger a set of interrupts: (i) full interrupts that indicate that no more inbound data message can be accepted because the reception queue is full (RXMSG_STAT.IF), (ii) error interrupts that indicate bus errors and timeout errors in case of multi-packet messages (RXMSG_STAT.IE), and (iii) reception interrupts that indicate that a full message was successfully received (RXMSG_STAT.IR).

The GRSRIO logical layer does not distinguish between single (up to 256 bytes) and multi-packet messages (up to 4K) and only one DMA descriptor is used for both types of messages. However, the user must set the maximum allowed message size for the queue in register RXMSG_CONF.MAX_SIZE.

All following conditions must be true for a message to be accepted by the queue:

- The queue is enabled.
- The queue is not full.
- The destination ID is accepted.
- The mailbox number is accepted.
- The message does not exceed the maximum allowed size.

If no queue is available that can accept the message, the GRSRIO logical layer replies to the destination node with an ERROR response. If there are queues that could accept the packet but they are all full or busy with ongoing multi-segment data messages, the GRSRIO logical layer replies to the destination node with a RETRY response.

The timeout can be configured globally in register TO_CONF and expresses the maximum allowed time span between a DONE response for one message segment and the reception of the next segment. If the timeout elapses, the descriptor will be closed with the Serial RapidIO Error (SE) flag set and an error interrupt is generated if enabled.

The circular buffer management between hardware and software is done via two descriptor pointers. The tail pointer in RXMSG_TLPTR is managed by hardware whereas the head pointer in RXMSG_HDPTR is managed by software.

The software can add one or more descriptors to the circular buffer and then activate them by increasing the head pointer value in RXMSG_HDPTR. Every time the GRSRIO logical layer receives a new message successfully, it will increment the tail pointer to the next descriptor. The software can always check how many free descriptors are left in the buffer by subtracting the tail pointer value from the head pointer value.

In the following, the relevant descriptor words of the message reception queues are described:

Table 1606. GRSRIO message reception descriptor word 0 (address offset 0x0)

31	0
ADDR_MSW	

31: 0 Most Significant Word of the Memory Start Addres (ADDR_MSW): MSW of the address of the start of the memory block reserved for the reception of data messages. This field is reserved when *CFG_AW* is set to “true”.

Table 1607. GRSRIO message reception descriptor word 1 (address offset 0x4)

31	0
ADDR(_LSW)	

31: 0 Memory Start Address (ADDR(_LSW)): when *CFG_AW* is set to 32, this is the address of the start of the memory block reserved for the reception of this message. When *CFG_AW* is set to 64, this is the LSW of that address.

Table 1608. GRSRIO message reception descriptor word 2 (address offset 0x8)

31	0
SOURCE_ID	

31: 0 Source Identifier (SOURCE_ID). RapidIO source device identifier. If the GRSIO is configured for 8-bit identifiers, only the lower 8 bits are valid.

Table 1609. GRSRIO message reception descriptor word 3 (address offset 0xC)

31	0
DEST_ID	

31: 0 Destination Identifier (DEST_ID). Rapid IO destination device identifier. If the GRSIO is configured for 8-bit identifiers, only the lower 8 bits are valid.

Table 1610. GRSRIO message reception descriptor word 4 (address offset 0x10)

31	30	29 28	27 26	25 20	19 18	17 16	15 4	3	2	1	0
VC	CR	PRIOR	LET	MBOX	TT	RES	MSIZE	R	SE	DO	RI

31 Virtual Channel (VC). Set if also the VC bit of the message packet was set.

30 Critical Request Flow (CR). Set when the message arrived in a packet in a critical or preferred flow with respect to other flows of the same priority.

29: 28 Priority (PRIOR). Priority of the packet in which this message arrived.

27: 26 Letter (LET). Although concurrent reception of different letters is not supported, messages can still arrive in different letters. This field contains the letter number (0-3) of the message.

25: 20 Mailbox (MBOX). This field contains the number of the mailbox to which this message was addressed. Messages for mailbox numbers greater than 3 have a maximum size of 256 bytes whereas messages for mailbox numbers 0 to 3 can have a size of up to 4K.

19: 18 Transport Type Field (TT). Set to 0b00 if device ID is 8-bit wide and set to 0b01 if device ID is 16-bit wide.

17: 16 RESERVED

Table 1610. GRSRIO message reception descriptor word 4 (address offset 0x10)

15: 4	Message Size (MSIZE). Size of the received message in bytes: Size = (MSIZE+1) bytes.
3	RESERVED
2	Serial RapidIO Error (SE). Set when a multi-packet message was not received correctly. The GRSRIO implements a response-to-request timeout mechanism according to RapidIO specification part 8. If this timeout elapses during the reception of a message, the descriptor is closed and this error flag is set.
1	Done Flag (DO). Set when the message was correctly received, that is, when the processing core generated and transmitted a DONE response to the destination node.
0	Enable Reception Interrupt (RI). If set, the GRSRIO triggers an interrupt when the message was correctly received, that is, when also the done flag DO is set. The interrupt is triggered after the DMA engine has finished all memory accesses related to this descriptor.

Table 1611. GRSRIO message reception descriptor word 5 (address offset 0x14)

31	0
TIMESTAMP_MSW	

31: 0	LSW of the Timestamp (TIMESTAMP_LSW). MSW of the local timer when the message was completely received or when message reception was aborted due to timeout.

Table 1612. GRSRIO message reception descriptor word 6 (address offset 0x18)

31	0
TIMESTAMP_LSW	

31: 0	LSW of the Timestamp (TIMESTAMP_LSW). LSW of the local timer when the message was completely received or when message reception was aborted due to timeout.

Table 1613. GRSRIO message reception descriptor word 7 (address offset 0x1C)

31	0
RESERVED	

31: 0	RESERVED

84.2.2 Transmitting Data Messages

Outbound data messages are stored in message or hybrid (depending on *g_hyb_que*) transmission queues. The desired number of transmission queues can be set by VHDL generic *g_no_msg_tx* or *g_no_hyb_tx* at compile time. In the latter case the queues are shared with the other transaction types.

If several transmission queues have messages ready to be sent, the one with the highest priority gets to transmit first. If more than one active queues have the highest priority, the arbitration between them is done with a round robin algorithm. However, the following conditions must be true:

- The queue is enabled.
- The message has a priority level that can currently be accepted by the SRIO end point.

Each transmission queue is a circular buffer that is located in memory. Before activating the queue, the memory start address of the circular buffer must be set in register TXMSG_MADDR(_LSW) and TXMSG_ADDR_MSW (or TX_MADDR(_LSW) and TX_MADDR_MSW) and its depth in register TXMSG_CTRL.CBD or TX_CTRL.CBD. Each queue can be configured to store between 2 and 65,536 message descriptors. Each descriptor has a size of 32 bytes and it is up to the software to reserve the necessary memory space for the whole circular buffer.

Each transmission queue can be configured to trigger a set of interrupts: (i) error interrupts that indicate bus errors, retry errors, and timeout errors (TXMSG_CTRL.IE or TX_CTRL.IE), and (ii) transmission interrupts that indicate that a full message was successfully transmitted (TXMSG_STAT.IT or TX_STAT.IT). The generation of transmission interrupts can be further configured by setting the transmission interrupt mode bit (TXMSG_CTRL.IM or TX_CTRL.IM). If set, a transmission interrupt is always triggered after the transmission queue became empty. If not set, the generation of transmission interrupts depends on the Enable Transmission Interrupt (TI) field of each message descriptor.

Messages or segments of messages are automatically retried until either the packet is accepted by the destination node or until the threshold level defined in the descriptor field RCOUNT is reached. For multi-packet messages, RCOUNT is the number of possible retries for all message segments together. If RCOUNT in the descriptor is set to zero the transmission will be retried until the transmission is successful.

Each transmission queue has a fixed source ID (TXMSG_SRC_ID or TX_SRC_ID), priority (TXMSG_CTRL.PRIO), critical request flow setting (TXMSG_CTRL.CR or TX_CTRL.CR), virtual channel setting (TXMSG_CTRL.VC) and transport type (TXMSG_CTRL.TT or TX_CTRL.TT). For each message individually, the destination ID (DEST_ID), mailbox number (MBOX, XMBOX) and letter number (LET) can be set up in the corresponding descriptor.

Each transmission queue implements a request-to-response timeout mechanism that counts the time between the transmission of a message (or message segment) and a response from the destination node. This timeout can be configured globally in the TO_CONF register. If the timeout elapses, the descriptor will be closed with the Timeout Error (TE) flag set and an error interrupt is generated if enabled.

A transmission queue serves one descriptor at a time. Two packets are sent from the same queue when the descriptor defines more than one packet, and then replies for them are awaited before moving to the next couple. This is done in order to enhance the throughput of transfers up to 4 KB. Receiving a RETRY request when two packets are outstanding will cause both packets to be sent again and when an ERROR response is received the descriptor is written back with the number of segment causing the error.

The circular buffer management between hardware and software is done via two descriptor pointers. The tail pointer in TXMSG_TLPTR or TX_TLPTR is managed by hardware whereas the head pointer in TXMSG_HDPTR or TX_HDPTR is managed by software.

Sending a data message does not increase the internal srcTID counter, as data messages are not identified using this field.

The software can add one or more descriptors to the circular buffer and then activate them by increasing the head pointer value in TXMSG_HDPTR or TX_HDPTR. Every time the GRSRIO logical layer transmits a new message successfully, it will increment the tail pointer to the next descriptor. The software can always check how many descriptors awaiting transmission are left in the buffer by subtracting the tail pointer value from the head pointer value.

In the following, the relevant descriptor words of the message transmission queues are described:

Table 1614. GRSRIO message transmission descriptor word 0 (address offset 0x0)

31	0
ADDR_MSW	

31: 0	Most Significant Word of the Memory Start Address: MSW of the address of the start of the memory block storing the payloads of the data messages to be transmitted. This field is reserved when CFG_AW = 32.

Table 1615. GRSRIO message transmission descriptor word 1 (address offset 0x4)

31	0
ADDR(_LSW)	

31: 0 (Less Significant Word of the) Memory Start Address: When CFG_AW = 32, this is the address pointing to the start of the memory block storing the payload of the message to be transmitted, otherwise it is the LSW of such address.

Table 1616. GRSRIO message transmission descriptor word 2 (address offset 0x8)

31	0
DEST_ID	

31: 0 Destination Identifier (DEST_ID). Rapid IO destination device identifier. If the GRSRIO is configured for 8-bit identifiers, only the lower 8 bits are valid.

Table 1617. GRSRIO message transmission descriptor word 3 (address offset 0xC)

31 24	23 22	21 20	19 16	15 1	0
RESERVED	LET	MBOX	XMBOX	RESERVED	TI

31: 24 RESERVED

23: 22 Letter (LET). This field contains the letter number (0-3) of the message.

21: 20 Mailbox (MBOX). This field contains the number of the mailbox to which this message is addressed. Messages for mailbox numbers greater than 3 (using XMBOX as extension) are only allowed to have a maximum size of 256 bytes whereas messages for mailbox numbers 0 to 3 can have a size of up to 4K.

19: 16 Extended Mailbox (XMBOX). This field specifies the upper four bits of the mailbox to which this message is addressed. If the message size is larger than 256 bytes, this field must be zero.

15: 1 RESERVED

0 Enable Transmission Interrupt (TI). If set, the GRSRIO triggers an interrupt when the message was correctly transmitted, that is, when also the done flag DO is set (only if TXMSG_CONF.IM or TX_CONF.IM is set to 0). The interrupt is triggered after the DMA engine has finished all memory accesses related to this descriptor.

Table 1618. GRSRIO message transmission descriptor word 4 (address offset 0x10)

31	30 22	21 6	5 4	3	2	1	0
R	MSIZE	RCOUNT	RES	RE	TE	SE	DO

31 RESERVED

30: 22 Message Size (MSIZE). Size of the message in multiples of 8 bytes: Size = (MSIZE+1)*8 bytes. After transmission, this field is updated by GRSRIO with the remaining data that could not be transmitted, that is, in case of a correctly transmitted message this field should always read 0.

21: 16 Retry Count (RCOUNT). Outbound messages are automatically retried until the destination node either accepts the message or until the user-defined retry threshold level in this field is reached. For multi-packet messages, this retry threshold level applies to all segments on the whole. After transmission, this field is updated by GRSRIO with the remaining number of retries, that is, if no retry was required the original field content does not change. When set to zero the transmission will be retried forever.

15: 4 RESERVED

Table 1618. GRSRIO message transmission descriptor word 4 (address offset 0x10)

3	Retry Error (RE). Set when this message was retried too many times and the retry threshold level was reached.
2	Timeout Error (TE). Set when the request-to-response timeout has expired, that is, when the destination node did not acknowledge the reception of the message (or part of the message in case of a multi-packet message) in time.
1	Serial RapidIO Error (SE). Set when an ERROR or unknown response was received from the destination node.
0	Done Flag (DO). Set when the message was correctly transmitted, that is, when the processing core received a DONE response from the destination node for each packet of the message.

Table 1619. GRSRIO message transmission descriptor word 5 (address offset 0x14)

31	0
TIMESTAMP_MSW	

31: 0	MSW of the Message Time-Stamp (TIMESTAMP_MSW): This is the MSW of the local timer when message transmission was completed (with or without an error), that is when the response or the last response for multi-segment messages is received.

Table 1620. GRSRIO message transmission descriptor word 6 (address offset 0x18)

31	0
TIMESTAMP_LSW	

31: 0	LSW of the Message Time-Stamp (TIMESTAMP_LSW): This is the LSW of the local timer when message transmission was completed (with or without an error), that is when the response or the last response for multi-segment messages is received.

Table 1621. GRSRIO message transmission descriptor word 7 (address offset 0x1C)

31	2	1	0
RESERVED			DSC_TYP

31: 2	RESERVED
1: 0	DSC_TYP: when *g_hyb_que* is set to true, this field must be set to 0b01 to mark the operation defined by the descriptor as a Data Message operation. This field is reserved and will be ignored when *g_hyb_que* is set to false.

84.2.3 Receiving Doorbell Messages

Inbound doorbell messages are stored in one or more doorbell reception buffers. The desired number of reception buffers can be set by VHDL generic *g_no_dbell_rx* at compile time. Doorbell messages are small enough to be stored directly in the circular buffer and therefore do not need DMA descriptors. Furthermore, descriptors are 4 words long instead of 8 to decrease latency of operations.

The inbound doorbell messages are stored to a specific buffer depending on their destination ID. The accepted destination ID of a buffer can be defined in register RXDBL_DST_ID. Furthermore, ranges of destination IDs can be accepted by also setting the masking field RXDBL_DST_MSK. If several doorbell reception buffers are configured to accept the same destination ID, the doorbell buffer with the lowest index number, that is currently not busy, will receive the doorbell message.

Each reception buffer is a circular buffer that is located in memory. Before activating the buffer, the memory start address of the circular buffer must be set in register RXDBL_MADDR_LSW (and RXDBL_MADDR_MSW if *CFG_AW = 64*) and its depth in register RXDBL_CTRL.CBD. Each

queue can be configured to store between 2 and 65,536 doorbell messages. Each doorbell message buffer entry has a size of 16 bytes and it is up to the software to reserve the necessary memory space for the whole circular buffer.

Each reception buffer can be configured to trigger a set of interrupts: (i) full interrupts that indicate that no more inbound doorbell messages can be accepted because the reception buffer is full (RXDBL_CTRL.IF), (ii) error interrupts that indicate bus errors (RXDBL_CTRL.IE), and (iii) reception interrupts that indicate that a doorbell message was successfully received (RXDBL_CTRL.IR).

All following conditions must be true for a doorbell message to be accepted by the queue:

- The queue is enabled.
- The queue is not full.
- The destination ID is accepted.

If no buffer is available that can accept the message, the GRSRIO logical layer replies to the destination node with an ERROR response. If a buffer is available but full, the GRSRIO logical layer replies to the destination node with a RETRY response.

The circular buffer management between hardware and software is done via two descriptor pointers. The tail pointer in RXDBL_TLPTR is managed by hardware whereas the head pointer in RXDBL_HDPTR is managed by software.

The software can add one or more buffer entries to the circular buffer and then activate them by increasing the head pointer value in RXDBL_HDPTR. Every time the GRSRIO logical layer receives a new doorbell message successfully, it will increment the tail pointer to the next buffer entry. The software can always check how many free buffer entries are left in the buffer by subtracting the tail pointer value from the head pointer value.

The GRSRIO logical layer also allows the forwarding of received doorbell messages to an external port. To enable this feature, the EXTDBL_CONF.DO signal must be set. In addition, the logical layer can be configured to only forward specific doorbell messages. This feature can be enabled by setting the EXTDBL_CONF.DC bit and by setting up a compare value in EXTDBL_FLTR.DBCOMP and a mask value in EXTDBL_FLTR.DBMASK.

In the following, the relevant buffer entry words of the doorbell reception buffers are described:

Table 1622. GRSRIO doorbell message reception buffer word 0 (address offset 0x0)

31	0
SOURCE_ID	

31: 0	Source Identifier (SOURCE_ID). RapidIO source device identifier. If the GRSIO is configured for 8-bit identifiers, only the lower 8 bits are valid.

Table 1623. GRSRIO doorbell message reception buffer word 1 (address offset 0x4)

31	0
DEST_ID	

31: 0	Destination Identifier (DEST_ID). Rapid IO destination device identifier. If the GRSIO is configured for 8-bit identifiers, only the lower 8 bits are valid.

Table 1624. GRSRIO doorbell message reception buffer word 2 (address offset 0x8)

31	16	15	11	10	9	8	7	6	5	4	3	1	0
INFO		RESERVED		TT		VC	CR	PRIOR		DO	RES		RI

*Table 1624.*GRSRIO doorbell message reception buffer word 2 (address offset 0x8)

31: 16	Doorbell Information (INFO). This field contains the payload of the received doorbell message.
15: 11	RESERVED
10: 9	Transport Type Field (TT). Set to 0b00 if device ID is 8-bit wide and set to 0b01 if device ID is 16-bit wide.
8	Virtual Channel (VC). Set if also the VC bit of the doorbell message packet was set.
7	Critical Request Flow (CR). Set when the doorbell message arrived in a packet in a critical or preferred flow with respect to other flows of the same priority.
6: 5	Priority (PRIOR). Priority of the packet in which this doorbell message arrived.
4	Done Flag (DO). Set when the doorbell message was correctly received, that is, when the processing core generated and transmitted a DONE response to the destination node.
3: 1	RESERVED
0	Enable Reception Interrupt (RI). If set, the GRSRIO triggers an interrupt when the doorbell message was correctly received, that is, when also the done flag DO is set. The interrupt is triggered after the DMA engine has finished all memory accesses to this buffer entry.

*Table 1625.*GRSRIO doorbell message reception buffer word 3 (address offset 0xC)

31	0
TIMESTAMP	

31: 0	Message Time-Stamp (TIMESTAMP). LSW of the local timer when the message was completely received or when message reception was aborted due to timeout.Timestamp of 64 bits are not supported for reception of doorbells.

84.2.4 Transmitting doorbell messages

Outbound doorbell messages are stored in doorbell or hybrid transmission queues (depending on *g_hyb_que*). The desired number of transmission buffers can be set by VHDL generic *g_no_dbell_tx* or *g_no_hyb_tx* at compile time. The descriptor in this case directly contains the SRIO payload, as it is just 2 B long. Furthermore, when *g_hyb_que* is set to false, descriptors are 4 words long instead of 8 to decrease latency of operations. When *g_hyb_que* is set to true TX descriptors of doorbells messages are 8 words long like the ones for the other transaction types.

If several transmission queues have messages ready to be sent, the one with the highest priority gets to transmit first. If more than one active queues have the highest priority, the arbitration between them is done with a round robin algorithm. However, the following conditions must be true:

- The queue is enabled.
- The doorbell message has a priority level that can currently be accepted by the SRIO end point.

Each transmission queue is a circular buffer that is located in memory. Before activating the buffer, the memory start address of the circular buffer must be set in register TXDBL_MADDR_LSW (and TXDBL_MADDR(_MSW)) or TX_MADDR_LSW (and TX_MADDR(_MSW)) and its depth in the register field TXDBL_CTRL.CBD or TX_CTRL.CBD. Each queue can be configured to store between 2 and 65,536 doorbell messages. Each buffer entry has a size of 16 bytes when *g_hyb_que* is set to “false” and 32 bytes when *g_hyb_que* is set to “true”. It is up to the software to reserve the necessary memory space for the whole circular buffer.

Each transmission buffer can be configured to trigger a set of interrupts: (i) error interrupts that indicate bus errors, retry errors, and timeout errors (TXDBL_CTRL.IE or TX_CTRL.IE), and (ii) transmission interrupts that indicate that a doorbell message was successfully transmitted (TXDBL_CTRL.IT or TX_CTRL.IT). The generation of transmission interrupts can be further con-

figured by setting the transmission interrupt mode bit (TXDBL_CTRL.IM or TX_CTRL.IM). If set, a transmission interrupt is always but only triggered after the transmission buffer became empty. If not set, the generation of transmission interrupts depends on the Enable Transmission Interrupt (TI) setting of the doorbell buffer entry.

Doorbell messages are automatically retried until either the packet is accepted by the destination node or until the threshold level defined in the buffer entry field RCOUNT is reached. If RCOUNT is set to zero they are retried forever.

Each transmission buffer has a fixed source ID (TXDBL_SRC_ID), priority (TXDBL_CTLR.PRIO), critical request flow setting (TXDBL_CTRL.CR), virtual channel setting (TXDBL_CTRL.VC) and transport type (TXDBL_CTRL.TT). For each doorbell message individually, the destination ID (DEST_ID) can be set up in the corresponding buffer entry.

Each transmission buffer implements a request-to-response timeout mechanism that counts the time between the transmission of a doorbell message and a response from the destination node. This timeout can be configured globally in register TO_CONF. If the timeout elapses, the buffer entry will be closed with the Timeout Error (TE) flag set and an error interrupt is generated if enabled. A transmission queue is blocked while it is awaiting a response from the destination node. However, other transmission queues can be serviced in the meanwhile.

Every time a new packet is sent the srcTID field of the packet is increased, also when a transaction is retried.

The circular buffer management between hardware and software is done via two descriptor pointers. The tail pointer in TXDBELLMSG3.TLPTR is managed by hardware whereas the head pointer in TXDBELLMSG3.HDPTR is managed by software.

The software can add one or more buffer entries to the circular buffer and then activate them by increasing the head pointer value in TXDBELLMSG3.HDPTR. Every time the GRSRIO logical layer transmits a new doorbell message successfully, it will increment the tail pointer to the next buffer entry. The software can always check how many buffer entries awaiting transmission are left in the buffer by subtracting the tail pointer value from the head pointer value.

The GRSRIO logical layer also allows the generation of doorbell messages through an external port. To enable this feature, the CORECONF1.DI bit must be set. Pending external doorbell transmission requests have always priority over internal transmission requests.

In the following, the relevant descriptor words of the message transmission queues are described:

*Table 1626.*GRSRIO doorbell message transmission buffer word 0 (address offset 0x0)

31	0
DEST_ID	

31: 0	Destination Identifier (DEST_ID). Rapid IO destination device identifier. If the GRSIO is configured for 8-bit identifiers, only the lower 8 bits are valid.

*Table 1627.*GRSRIO doorbell message transmission buffer word 1 (address offset 0x4)

31	16	15	1	0
INFO		RESERVED		TI

31: 16	Doorbell Information (INFO). This field contains the payload of the doorbell message.
15: 1	RESERVED
0	Enable Transmission Interrupt (TI). If set, the GRSRIO triggers an interrupt when the doorbell message was correctly transmitted, that is, when also the done flag DO is set. The interrupt is triggered after the DMA engine has finished all memory accesses to this buffer entry.

*Table 1628.*GRSRIO doorbell message transmission buffer word 2 (address offset 0x8)

31 22	21 6	5	4	3	2	1	0
RESERVED	RCOUNT	RES		RE	TE	SE	DO

31: 22 RESERVED

21: 16 Retry Count (RCOUNT). Outbound doorbell messages are automatically retried until the destination node either accepts the message or until the user-defined retry threshold level in this field is reached. After transmission, this field is updated by GRSRIO with the remaining number of retries, that is, if no retry was required the original field content does not change.When this field is set to zero the transmission will be retried forever.

15: 4 RESERVED

3 Retry Error (RE). Set if the doorbell message was retried as many times as defined by RCOUNT but the transmission was still not successful.

2 Timeout Error (TE). Set when the request-to-response timeout has elapsed, that is, when the destination node did not acknowledge the reception of the doorbell message in time.

1 Serial RapidIO Error (SE). Set when an ERROR response was received from the destination node.

0 Done Flag (DO). Set when the doorbell message was correctly transmitted, that is, when the processing core received a DONE response from the destination node.

*Table 1629.*GRSRIO doorbell message transmission buffer word 3 (address offset 0xC)

31 0
TIMESTAMP(_MSW_/_LSW)

31: 0 Message Time-Stamp (TIMESTAMP). When *g_hyb_que* is set to “false” 64 bit timestamps are not supported and this is the LSW of the local timer when message transmission was completed, that is when the response is received (with or without an error). When *g_hyb_que* is set to “false” it is the MSW instead.

The fields below exist only when *g_hyb_que* is set to “true” (hybrid-queues mode).

*Table 1630.*GRSRIO doorbell message transmission buffer word 4 (address offset 0x10)

31 0
TIMESTAMP(_LSW)

31: 0 Least significant Word of the Time-Stamp: LSW of the local timer when message transmission was completed, that is when the response is received (with or without an error), captured just before the “Transmission interrupt” is triggered.

*Table 1631.*GRSRIO doorbell message transmission buffer word 5 (address offset 0x14)

31 0
RESERVED

31: 0 Reserved

*Table 1632.*GRSRIO doorbell message transmission buffer word 6 (address offset 0x18)

31 0
RESERVED

31: 0 Reserved

*Table 1633.*GRSRIO doorbell message transmission buffer word 7 (address offset 0x1C)

31	2	1	0
RESERVED		DES_TYP	

31: 2	Reserved
1: 0	Descriptor Type (DES_TYP): indicates which kind of descriptor this is when in hybrid queues mode. Must be set to 0b00 or to 0b11 to indicate that this is a doorbell descriptor.

84.2.5 Inbound I/O operations

Inbound memory I/O operations can gain direct access to the local memory. The GRSRIO logical layer supports all memory I/O operations except of MAINTENANCE Read, Write and Port-Write Requests since these operations are already handled by the SRIO End-Point. The core provides both memory protection on physical addresses and address translation by means of two independent sets of partitions and windows.

When *CFG_AW* = 32 (local addresses of 32 bits), the physical address is decided according to SRIO Address(33:0), effectively allowing to handle a virtual memory space bigger than the physical one. The values of Address(33:30) are used to decide which TRNSWIN_CONF.WIN_ADDR to use in the final result of the translation. The physical address is then calculated by the following formula:

Physical Address = TRNSLWIN.WIN_ADDR (window *i*) &
SRIO Address((27- RXIO_CONF.ADDR_SHIFT):0)

Where *i* is the non negative integer found in Address(33:30), “&” is the concatenation operator and i is the index of the hit translation window.

When *CFG_AW = 64* (local addresses of 64 bits), the physical address is decided according to the SRIO Address(65:0), effectively allowing the handling of a virtual memory space bigger than the physical one. The values of Address(65:62) are used to decide which TRNSWIN_-CONF.WIN_ADDR in the final result of the translation. The physical address is then defined by the following formula:

Physical Address = TRNSLWIN.WIN_ADDR (window *i*) &
SRIO Address((59-RXIO_CONF.ADDR_SHIFT):0)

Where *i* is the non negative integer found in Address(65:62) and “&” is the concatenation operator.

The translation windows are all disabled by default and only the ones enabled by setting the respective TRNSWIN.EN to “1” are kept into account for translation. If no enabled translation windows are hit, the physical address is obtained directly assigning the value of SRIO Address(31:0) to the physical address in case of *CFG_AW = 32* and SRIO Address(63:0) to the physical address in case of in case of *CFG_AW = 64*.

Remote memory access is by default deactivated. The software can activate up to four (if *g_8_in_win* is set to “false”) or eight (if *g_8_in_win* is set to “true”) memory partitions using the switches MEMPRT.PWE for writing and MEMPRT.PRE for reading each partition. Operations trying to read from a partition where read is not enable or writing in a partition where writing is not enabled are ignored and the GRSRIO logical layer replies an ERROR response to the destination node in such a case.

The start address of each partition is defined in the respective MEMPRT.PART_ADDR. These fields contain the upper 12 bits of such address. The size of each partition can be adjusted by setting a corresponding mask value in the respective MEMPRT.PART_MASK.

A physical address of an inbound memory I/O operation falls into the memory protection partition *i* if it satisfies the following condition:

(Physical Address (CFG_AW-1:CFG_AW-11) **xor** MEMPRT.PART_ADDR (partition *i*))
and MEMPRT.PART_MASK (partition *i*)= 0x000

In this case, a mask value of 0xFFF defines the smallest possible partition with a size of 1MB and a mask value of 0x000 creates a partition that spans the whole 32-bit wide memory space. In this case, a

mask value of 0xFFF defines the smallest possible partition with a size of 4GB and a mask value of 0x000 creates a partition that spans the whole 64-bit wide memory space. In the next release (GEN3) it is planned to have a mask value of 0xFFF to define the smallest possible partition with a size of 1MB and a mask value of 0x000 to define partitions of 4GB.

The software can set up two types of interrupts: Memory access interrupts (RXIO_CTRL.IM) are triggered after each successful memory access, error interrupts (RXIO_CTRL.IE) are triggered in case of bus errors or forbidden memory access error, that is, when the destination node tries to access a memory space, which is either not covered by a partition or which does not allow write or read access.

For debug purposes, the last accessed local memory address is logged in the read-only field RXIO_LACCESS_MSW (if *CFG_AW* is set to 64) and RXIO_LACCES(_LSW).

84.2.6 Transmitting Memory I/O operations

Outbound memory I/O operations are stored in one or more memory I/O or hybrid transmission queues. The desired number of transmission queues can be set by VHDL generic *g_no_io_tx* or *g_no_hyb_tx* at compile time.

If several transmission queues have messages ready to be sent, the one with the highest priority gets to transmit first. If more than one active queues have the highest priority, the arbitration between them is done with a round robin algorithm. However, the following conditions must be true:

- The queue is enabled.
- The memory I/O operation message has a priority level that can currently be accepted by the SRIO end point.

Each transmission queue is a circular buffer that is located in memory. Before activating the buffer, the memory start address of the circular buffer must be set in register TXIO_MADDR_MSW and TXIO_MADDR(_LSW) or TX_MADDR_MSW and TX_MADDR(_LSW) and its depth in the field register TXIO_CTRL.CBD or TX_CTRL.CBD. The queue can be configured to store between 2 and 65,536 memory I/O operation descriptors. Each descriptor has a size of 32 bytes and it is up to the software to reserve the necessary memory space for the whole circular buffer.

Each I/O descriptor for NREAD, NWRITE, NWRITE_R and SWRITE can define a number of operations with contiguous payload from 1 to 4096. This is done in order to be able to send contiguous data up to 1MB with a single descriptor (avoid dead time due to the opening and the closing of descriptor at the end of each operation). In this case the RapidIO address will be increased according to the formula below each time the packet will be sent, although the one contained in the descriptor will not be updated:

$$\text{RapidIO address} = (\text{EXT} \,\&\, \text{SRIO_EXT_ADDR} \,\&\, \text{SRIO_ADDR} \,\&\, b000) + P_B * I_p$$

Where "&" is the concatenation operator, P_B is the length in bytes of the payload in each operation of the descriptor (defined by the field SIZE of the descriptor), I_p is the number of packets already sent from this descriptor, and EXT, SRIO_EXT_ADDR, SRIO_ADDR are the respective fields in the TX descriptor.

Each queue can be configured to trigger a set of interrupts: (i) error interrupts that indicate bus errors, SRIO errors and timeout errors (TXIO_CTRL.IE or TX_CTRL.IE), and (ii) transmission interrupts that indicate that a memory I/O operation was successfully transmitted (TXIO_CTRL.IT or TX_CTRL.IT). The generation of transmission interrupts can be further configured by setting the transmission interrupt mode bit (TXIO_CTRL.IM or TX_CTRL.IM). If set, a transmission interrupt is only triggered after the transmission queue became empty. If not set, the generation of transmission interrupts depends on the Enable Transmission Interrupt (TI) setting of the descriptor.

Each transmission queue has a fixed source ID (TXIO_SRC_ID or TX_SRC_ID), priority (TXIO_CTRL.PRIO or TX_CTRL.PRIO), critical request flow setting (TXIO_CTRL.CR or TX_CTRL.CR), virtual channel setting (TXIO_CTRL.VC) and transport type (TXIO_CTRL.TT or TX_CTRL.TT).

For each memory I/O operation individually, the destination ID (DEST_ID) can be set up in the corresponding descriptor.

Each transmission queue implements a request-to-response timeout mechanism that counts the time between the transmission of a memory I/O operation and a response from the destination node (in case of operations that require a response such as NWRITE_R or atomic transactions). This timeout can be configured globally in register TO_CONF. If the timeout elapses, the descriptor will be closed with the Timeout Error (TE) flag set and an error interrupt is generated if enabled.

A transmission queue serves one descriptor at a time. Anyway, more outstanding packets per queue (two, planned to be extended to 64 in future releases) is blocked while it is awaiting a response from the destination node. However, other transmission queues can be serviced in the meanwhile.

The circular buffer management between hardware and software is done via two descriptor pointers. The tail pointer in TXIO_TLPTR or TX_TLPTR is managed by hardware whereas the head pointer in TXIO_HDPTR or TX_HDPTR is managed by software.

The software can add one or more descriptors to the circular buffer and then activate them by increasing the head pointer value in TXIO_HDPTR or TX_HDPTR. Every time the GRSRIO logical layer transmits a new memory I/O operation successfully, it will increment the tail pointer to the next descriptor. The software can always check how many operations awaiting transmission are left in the buffer by subtracting the tail pointer value from the head pointer value.

Every time a new packet is sent the srcTID field of the packet is increased, except when NWRITE and SWRITE transactions are sent (they have no srcTID field).

The type of the operation is defined in field TYPE, and the read- or write size in field SIZE. The size field is the concatenation of wdptr and rdsize/wrsize (see also RapidIO Interconnect Specification Part 1), the possible values are listed in Table 1634.

- For NREAD operations, the data read from the destination node will be stored at the memory address defined in ADDR.
- For NWRITE, NWRITE_R and SWRITE operations, the data which should be written to the remote node must be stored at the address defined in ADDR.
- For ATOMIC Set, Clear, Increment and Decrement operations, the data returned from the destination node will be stored at the memory address defined in ADDR. It is always fixed to 8 bytes. However, only wdptr and rdsize combinations are allowed that modify 1, 2, or 4 bytes.
- For ATOMIC Test-and-Swap and Swap operations, both the data transmitted to the destination node (the “swap” data) and the data returned from the destination node is stored at the address defined in ADDR. It is always fixed to 8 bytes. But again, only wdptr and wrsize combinations are allowed that modify 1, 2, or 4 bytes.
- For ATOMIC Compare-and-Swap operations, both the data transmitted to the destination node (the “compare” data and the “swap” data) and the data returned from the destination node is stored at the address defined in ADDR. The outgoing data is fixed to 16 bytes, where the first 8 bytes store the “compare” data and the second 8 bytes the “swap” data. The data returned from the destination node has always 8 bytes. Like for the other ATOMIC transactions, only wdptr and wrsize combinations that modify 1, 2, or 4 bytes are allowed.
- For MAINTENANCE Read operations, the data read from the destination node will be stored at the memory address defined in ADDR. It can be 4 or 8 bytes long, therefore the SIZE field must be set to b0100, b1100 or b01011.

- For MAINTENANCE Write and Port-Write Request operations, the data which should be written to the remote node must be stored at the address defined in ADDR. It can be 4 or 8 bytes long, therefore the SIZE field must be set to b0100, b1100 or b01011.

Table 1634. Allowed combinations of wdptr and rdsize/wrsize for SIZE field

wdptr	rdsize/wrsize	No. of Bytes	Bytes Lanes	Notes
0	0000	1	10000000	
0	0001	1	01000000	
0	0010	1	00100000	
0	0011	1	00010000	
1	0000	1	00001000	
1	0001	1	00000100	
1	0010	1	00000010	
1	0011	1	00000001	
0	0100	2	11000000	
0	0101	3	11100000	
0	0110	2	00110000	
0	0111	5	11111000	
1	0100	2	00001100	
1	0101	3	00000111	
1	0110	2	00000011	
1	0111	5	00011111	
0	1000	4	11110000	
1	1000	4	00001111	
0	1001	6	11111100	
1	1001	6	00111111	
0	1010	7	11111110	
1	1010	7	01111111	
0	1011	8	11111111	
1	1011	16		
0	1100	32		
1	1100	64		
0	1101	96		only read
1	1101	128		
0	1110	160		only read
1	1110	192		only read
0	1111	224		only read
1	1111	256		

In the following, the relevant descriptor words of the memory I/O operations transmission queues are described:

Table 1635. GRSRIO I/O operation transmission descriptor word 0 (address offset 0x0)

31	0
ADDR_MSW	

31: 0 This field is effective only when the CFG_AW is set to 64.

*Table 1636.*GRSRIO I/O operation transmission descriptor word 1 (address offset 0x4)

31	0
ADDR(_LSW)	

31: 0 LSW Local Memory Start Address (LSW_ADDR). Lowest 32 bits of the address pointing to the start of the memory block storing the payload of the I/O write operation (NWRITE, NWRITE_R, SWRITE, MAINTENANCE Write and Port-Write Request) to be transmitted. For NREAD operations, this address is the start address where the received data will be stored. When CFG_AW = 32 this is the full 32-bit local memory address.

*Table 1637.*GRSRIO I/O operation transmission descriptor word 2 (address offset 0x8)

31	0
SRIO_EXT_ADDR	

31: 0 Serial RapidIO Extended Address (SRIO_EXT_ADDR). The extended 32 bits of a 66-bit physical address (bits 32:63 in big-endian notation).

*Table 1638.*GRSRIO I/O operation transmission descriptor word 3 (address offset 0xC)

31	30 29	28 0
R	EXT	SRIO_ADDR/HOP_CONF

31 RESERVED

30: 29 Serial RapidIO XAMSBS Field (EXT). Most significant bits of a 66-bit physical address (bits 64:65 in big-endian notation).

28: 0 Serial RapidIO Address / Hop Count & Configuration Offset (SRIO_ADDR/HOP_CONF). Lower part of the RapidIO memory address. The least significant three bits of the address are not specified as they are always set to logic 0. Therefore, this field contains bits 0:28 (in big-endian notation) of the RapidIO address. In case of maintenance operations, this field contains the hop count (28:21) concatenated with the configuration offset value (20:0).

*Table 1639.*GRSRIO I/O operation transmission descriptor word 4 (address offset 0x10)

31	0
DEST_ID	

31: 0 Destination Identifier (DEST_ID). Rapid IO destination device identifier. If the GRSIO is configured for 8-bit identifiers, only the lower 8 bits are valid.

*Table 1640.*GRSRIO I/O operation transmission descriptor word 5 (address offset 0x14)

31 26	25 21	20 17	16	15 ¨4	3	2	1	0
RESERVED	SIZE	TYPE	TI	NIOOP	R	TE	SE	DO

31: 26 RESERVED

25: 21 I/O Operation Size (SIZE). Read or write size of I/O operation. Concatenation of wdptr and rdsize/wrsize.

Table 1640. GRSRIO I/O operation transmission descriptor word 5 (address offset 0x14)

20: 17	I/O Operation Type (TYPE). Type of the I/O operation: 0: NREAD, 1: NWRITE, 2: NWRITE_R, 3: SWRITE, 4: ATOMIC Set, 5: ATOMIC Clear, 6: ATOMIC Increment, 7: ATOMIC Decrement, 8: ATOMIC Test-and-Swap, 9: ATOMIC Swap, 10: ATOMIC Compare-and-Swap, 11: MAINTENANCE Read, 12: MAINTENANCE Write, 13: MAINTENANCE Port-Write Request, all others: reserved.
16	Enable Transmission Interrupt (TI). If set, the GRSRIO triggers an interrupt when the I/O operation was correctly transmitted, that is, when also the done flag DO is set. The interrupt is triggered after the DMA engine has finished all memory accesses related to this descriptor.
15: 4	Number of I/O operations (NIOOP): indicates how many I/O operations are defined by the descriptor (the number of I/O transactions with contiguous payload will be NIOOP+1). This functionality can be used (NIOOP greater than 0) only with NREAD, NWRITE,NWRITE_R and SWRITE descriptors. The maximum number of I/O operations definable by a single descriptor is 4096, which allows transfers up to 1 MB with a single descriptor (when SIZE is set to 0b11111).
3	Reserved
2	Timeout Error (TE). Set when the request-to-response timeout has elapsed, that is, when the destination node did not acknowledge the reception of the message (only for operations such as NWRITE_R that generate DONE responses).
1	Serial RapidIO Error (SE). Set when an ERROR response was received from the destination node.
0	Done Flag (DO). Set when the I/O operation was correctly transmitted, that is, when the processing core received a DONE response from the destination node (only for operations such as NWRITE_R that generate DONE responses).

Table 1641. GRSRIO I/O operation transmission descriptor word 6 (address offset 0x18)

31	0
TIMESTAMP	

31: 0	I/O Operation Time-Stamp (TIMESTAMP). LSW of the local timer when transmission of all the transactions defined by the descriptor is completed, that is when the last response is received. For requests without responses, the timestamp is set when the last request is generated. 64 bit timestamp are not supported Please confirm:

Table 1642. GRSRIO I/O operation transmission descriptor word 7 (address offset 0x1C)

31	2	1	0
RESERVED		DES_TYP	

31: 2	RESERVED
1: 0	Descriptor Type (DES_TYP): indicates which kind of descriptor this is when in hybrid queues mode. Must be set to 0b10 to indicate that this is a I/O descriptor.

84.3 MECS Time Synchronization Protocol

The core, combined with the SRIOGEN2-IP end point, implements the “MECS Time Synchronization Protocol” according to RapidIO specifications Rev 4.0 if *g_mecs* is set to “true”. SMECS are not implemented. “Enhanced” MECS events with cmd = i are signaled from and triggered in the end point as changes in the signal level of respectively MECS_CAPTURED_IN[i] and MECS_TRIGGER_OUT[i].

84.4 Bus Master interface

84.4.1 Overview

Atomicity of operations as a target is supported by the AHB Master. Atomicity of operations as a target is currently not supported by the AXI4 Master. For this reason, when the grsrio_gen2axi wrapper is used inbound atomic operations are discarded and flagged as unsupported transaction.

The generic_bm_x is designed to use a standard bus master interface in which the translation of bus master commands to AHB or AXI bus protocol is handled by the component. This way if an IP core is designed to handle memory accesses with the standard bus master interface, which is defined in this document, it can be made compatible to AHB or AXI buses without additional effort. In addition, memory access control is simplified since standard bus interface (AHB,AXI) features like boundary crossing, alignment requirements is handled by the generic_bm_x IP core and no special action needed on the standard bus mater interface. Furthermore, additional bus protocols apart from AHB or AXI might be added in the future. The generic_bm_x core has a variety of configuration options to simplify the memory access logic required for bus master interface and optimize the bus accesses on AHB or AXI bus.

Fig. 226 illustrates the overall block diagram of generic_bm_x. Front-End handles all the handshaking on the bus_master interface. Middle-End handles all the burst related operations and generate standard commands to the back-end interface. Back-end interface generates bus specific burst commands depending on the selected bus protocol type. For each specific bus protocol (AHB,AXI) the back-end component is changed.

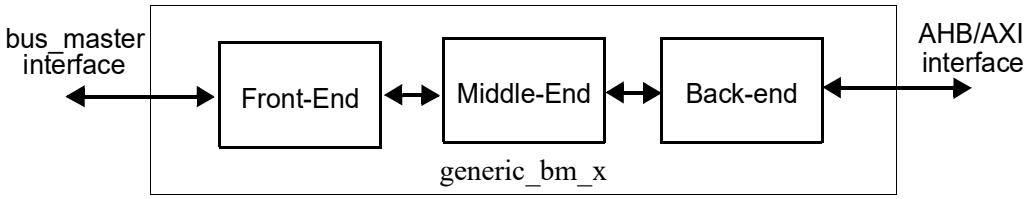


Figure 226. Overall block diagram of generic_bm_x

84.4.2 Configuration Options

MAX_SIZE: this generic determines the total maximum possible size of the burst on the bus master interface in terms of **bytes**. It has to be power of two. This generic also determines the width of the bmrd_size and bmwr_size input ports. This generic is fixed to 256 for GRSRIO core.

BM_DW: Data-width of the bus master interface in terms of **bits** and has to be a multiple of 32. This generic is fixed to 128 for GRSRIO core.

BE_DW: Data-width of the back-end bus in terms of **bits**. It can be equal or narrower compared to BM_DW and has to be a multiple of 32.

ADDR_WIDTH: Address width of the front-end and back-end bus.

MAX_BURST_LENGTH: Determines the maximum length of a burst transaction in terms of **number of beats** that is generated on the back-end bus. This generic can be useful if the maximum burst length needs to be limited to have fair arbitration in the system. It should be noted that the total maximum size of the burst in terms of bytes that is generated on the back-end bus depends on the be_dw

and can be calculated with the formula “number_of_beats*(be_dw/8)”. Limiting the maximum burst length can reduce the throughput hence it is recommended not be limited unless there is a need, for example due to arbitration fairness on the back-end bus side. In order to calculate the maximum possible burst length the following formula can be used “max_size/(be_dw/8)” and max_burst_length generic can not exceed this value. Also if max_size is bigger than the allowed burst_chop_mask value for the specific bus protocol then max_size can be replaced with maximum burst_chop_mask value since the total allowed burst size in terms of bytes can not exceed maximum burst_chop_mask value.

BURST_CHOP_MASK: Determines when a burst should end a new burst should start on the back-end bus in terms of **byte alignment.** Maximum and default value of this generic depends on the back-end protocol. Maximum value is 1024 bytes for AHB and 4096 bytes for AXI which is determined by the protocol specifications. Reducing the byte alignment might be useful when combined with max_burst_length in cases where the memory that is accessed by the back-end bus has an internal buffer which gives better throughput on specific alignments. For example if max_burst_length is limited to 8 beats on a 32-bit bus and a slave memory gives the best throughput when bursts are aligned to 32-byte boundary than burst_chop_mask can be set to 32. This way, if a burst on the bus_master front and starts with an address which is not aligned to 32-bytes, the first burst will be limited to finish on 32-byte alignment so that the remaining bursts will be always aligned to 32-bytes. It should be noted that, value of this generic is not recommended to be reduced from the default value unless a specific optimization is needed as explained in this chapter , otherwise the throughput will be reduced unnecessarily.

BE_RD_PIPE: When it is set to zero a pipeline stage exists between the read data path of the back-end bus to the read data-path of the bus_master interface. When set to one, this pipeline stage is removed. Setting this value to zero reduces read latency by one for aligned accesses but if the timing requirements do not met it can be set to a non-zero value.

UNALIGN_LOAD_OPT: When set to one, unaligned load optimization is enabled. When the optimization is enabled and a burst access do not finish exactly at the end of the data word of the back-end bus, then instead of generating a separate narrow access for the last word, the entire word is read and the corresponding part is used. This reduces the latency for unaligned burst accesses. But since this optimization reads more data than requested it should not be activated if the data is going to be read from a FIFO-like structure otherwise it can cause data loss.

EXCL_ENABLED: This generic only exists in AHB back-end bus and not compatible with the AXI back-end bus. When set to true it generates locked accesses on the AHB bus. This generic is fixed to true with the AHB back-end. GRSRIO core supports atomic transactions only when using AHB back-end.

84.4.3 Access Alignment

The generic_bm_x IP core does not have any alignment restriction on the bus_master interface. All accesses on the bus_master interface will be translated to access types that are allowed by the back-end bus. If the start address of an access is unaligned relative to the back-end bus, the first accesses on the back-end bus will be narrow accesses until the address reaches to an aligned value. After that a burst access will be generated as much as the size allows. If the ending is not aligned the last accesses will also be narrow accesses. For highest throughput the accesses should be aligned relative to the back-end bus width.

The generic_bm_x IP core will handle the boundary crossing for the back-end bus hence there is no restriction on the start address or the size of the burst on the bus_master interface. But since the burst has to restart on boundary crossing, in order to achieve high throughput with long bursts, the start address should be aligned such a way that boundary crossings are eliminated or minimized.

84.4.4 Endianness

The endianness on the back-end side is always big-endian. On the bus-master interface the most significant byte of the access is always placed to the uppermost bits (relative to the vhdl std_logic_vec-

tor). It should be noted that, the bus_master interface is not byte-invariant. Regardless of the access size or start address, the data word that is outputted do not have any gaps and the most significant byte of the access is placed to the uppermost bits and the remaining data follows to the low order bits.

In order to achieve big-endianness on the AXI back-end bus the bytes are swapped since the AXI protocol is byte-invariant. Following examples shows different access patterns with different back-end bus protocols and data-widths. For all accesses the bit position is relative to the std_logic_vector used in VHDL.

Fig. 227 illustrates a 16Byte access on both AHB and AXI backends, when both BM_DW and BE_DW generics are set to 128. Fig. 228 illustrates a 16 Byte access on both AHB and AXI backends, when BM_DW is set to 128 and BE_DW is set to 64. Fig. 229 illustrates a 2 Byte access on both AHB and AXI backends, when BM_DW is set to 128 and BE_DW is set to 64.

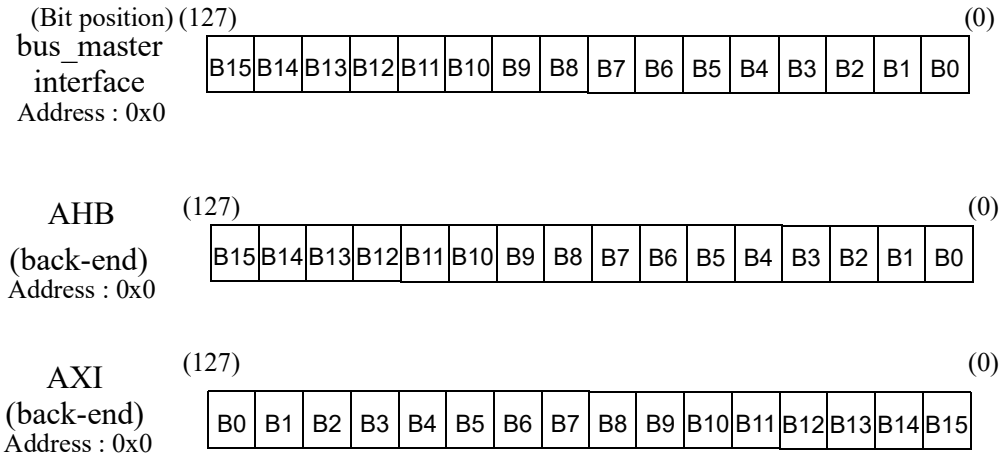


Figure 227. 16 Byte Access on AHB and AXI back-ends (BM_DW=128-bit, BE_DW=128-bit)

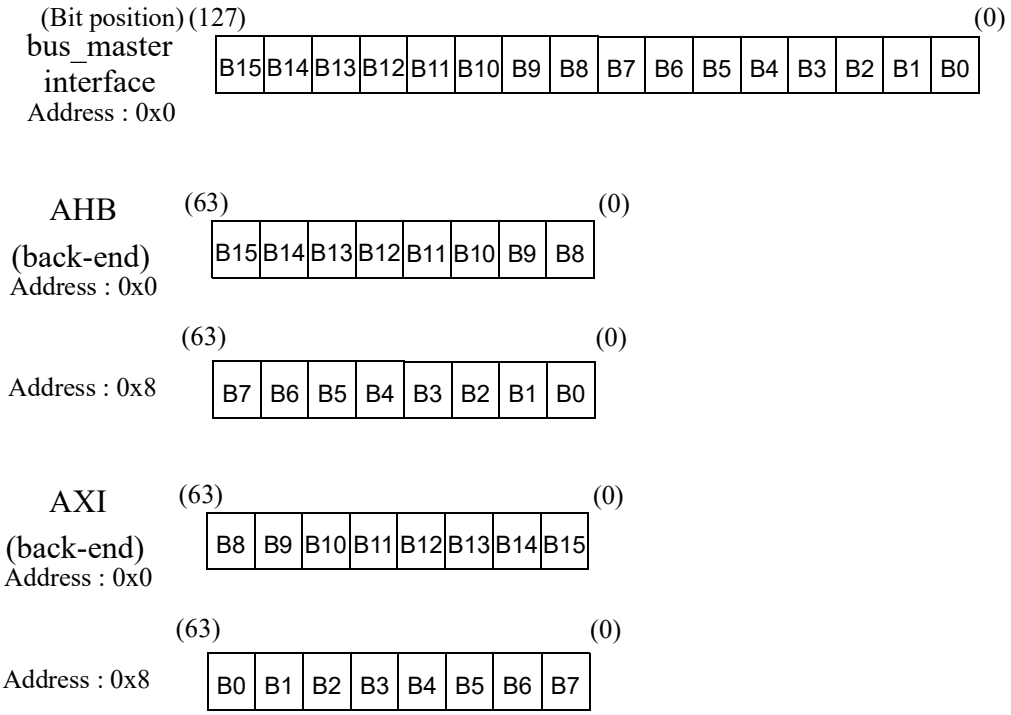


Figure 228. 16 Byte Access on AHB and AXI back-ends (BM_DW=128-bit, BE_DW=64-bit)

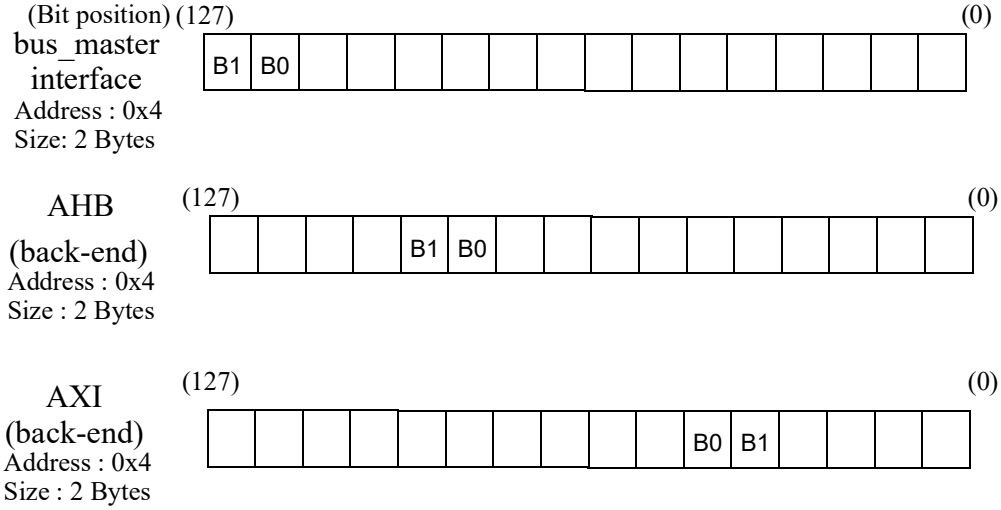


Figure 229. 2 Byte Access on AHB and AXI back-ends (BM_DW=128-bit, BE_DW=128-bit)

84.4.5 Software Considerations

The bus master interface of the GRSRIO core does not have any alignment restriction. But alignment can be important to achieve low latency and high throughput. In order to improve latency of doorbell messages (where the payload is contained in the descriptor), the descriptor should be aligned to 16 Byte boundary. When transmitting packets with 256 Byte payload, the best throughput can be achieved if data-packet is aligned to 256 Byte boundary since it will avoid boundary crossing on both AHB and AXI.

84.5 Registers

The core is programmed through registers mapped into APB address space. The registers are listed in table and described in detail in the subsequent tables. Addresses not listed in table are reserved. A read access to a reserved register, or reserved field with a register, will always return zero, and a write access has no effect. The register layout used is exemplified in table 1644, and the values used in the reset value row and field type row are explained in tables 1645 and 1646.

To easily support allocation of hardware resources to software entities, the address space for each hardware resource (queues, translation windows, memory protection partitions) are placed on 4KB boundaries.

Internally, the GRSRIO core decodes the address as listed in table 1643:

Table 1643. Decoding of ABP address

Bits	Description
19:17	Type of register block.
16:12	Index of queues/windows/partitions
11:2	Number of register.

The memory map has a size of 1MB and must also be aligned to 1MB boundaries.

Table 1644.<APB address offset> - <Register acronym> - <Register name>

31 24	23 16	15 8	7 0
EF3	EF2	EF1	EF0
<Reset value for EF3>	<Reset value for EF2>	<Reset value for EF1>	<Reset value for EF0>
<Bit-field type for EF3>	<Bit-field type for EF2>	<Bit-field type for EF1>	<Bit-field type for EF0>

31: 24 Example bit-field 3 (EF3) - <Bit-field description>
23: 16 Example bit-field 2 (EF2) - <Bit-field description>
15: 8 Example bit-field 1 (EF1) - <Bit-field description>
7: 0 Example bit-field 0 (EF0) - <Bit-field description>

Table 1645. Reset value definitions

Value	Description
0	Reset value 0. Used for single-bit fields.
1	Reset value 1. Used for single-bit fields.
0xNN	Hexadecimal representation of reset value. Used for multi-bit fields.
n/r	Field not reseted
*	Special reset condition, described in textual description of the bit-field. Used for example when reset value is taken from an input signal.

Table 1646. Bit-field type definitions

Value	Description
r	Read-only. Writes have no effect.
rw	Readable and writable.
rw*	Readable and writable. Special condition for write, described in textual description of the bit-field.
wc	Write-clear. Readable, and cleared when written with a 1. Writing 0 has no effect.

*Table 1647.*GRSRIO registers when *g_hyb_que* is set to false.

APB address offset	**Register**
0x00000	CORE_CAP
0x00004	CORE_RST
0x00008	TO_CONF
0x0000C	CORE_STAT
0x00010	INT_TXMSG_STAT
0x00014	INT_RXMSG_STAT
0x00018	INT_TXDBL_STAT
0x0001C	INT_RXDBL_STAT
0x00020	INT_TXIO_STAT
0x00024	INT_RXIO_STAT
0x00028	EXTDBL_CTRL
0x0002C	EXTDBL_FLTR
0x00030	EXTDBL_SRC_ID
0x00034	EXTDBL_DEST_ID
0x00038	ADDRTRNSL_SHIFT
0x10000+ i*0x1000	TRNSLWIN_CONF (window *i*)
0x20000+ i*0x1000	MEMPRT_CONF (partition *i*)
0x40000 + i*0x1000	TXMSG_CTRL (queue i)
0x40004 + i*0x1000	TXMSG_STAT (queue i)
0x40008 + i*0x1000	TXMSG_SRC_ID (queue i)
0x40010 + i*0x1000	TXMSG_TLPTR (queue i)
0x40014 + i*0x1000	TXMSG_HDPTR (queue i)
0x40018 + i*0x1000	TXMSG_MADDR_MSW[1] (queue i)
0x4001C + i*0x1000	TXMSG_MADDR(_LSW) (queue i)
0x60000 + i*0x1000	RXMSG_CTRL (queue i)
0x60004 + i*0x1000	RXMSG_STAT (queue i)
0x60008 + i*0x1000	RXMSG_DST_ID (queue i)
0x6000C + i*0x1000	RXMSG_DST_MSK (queue i)
0x60010 + i*0x1000	RXMSG_TLPTR (queue i)
0x60014 + i*0x1000	RXMSG_HDPTR (queue i)
0x60018 + i*0x1000	RXMSG_MADDR_MSW[1] (queue i)

*Table 1647.*GRSRIO registers when *g_hyb_que* is set to false.

APB address offset	Register
0x6001C + i*0x1000	RXMSG_MADDR(_LSW) (queue i)
0x80000 + i*0x1000	TXDBL_CTRL (queue i)
0x80004 + i*0x1000	TXDBL_STAT (queue i)
0x80008 + i*0x1000	TXDBL_SRC_ID (queue i)
0x80010 + i*0x1000	TXDBL_TLPTR (queue i)
0x80014 + i*0x1000	TXDBL_HLPTR (queue i)
0x80018 + i*0x1000	TXDBL_MADDR_MSW[1] (queue i)
0x8001C + i*0x1000	TXDBL_MADDR(_LSW) (queue i)
0xA0000+ i*0x1000	RXDBL_CTRL (queue i)
0xA0004 + i*0x1000	RXDBL_STAT (queue *i*)
0xA0008 + i*0x1000	RXDBL_ID (queue *i*)
0xA000C+ i*0x1000	RXDBL_MSK (queue *i*)
0xA0010+ i*0x1000	RXDBL_TLPTR (queue *i*)
0xA0014+ i*0x1000	RXDBL_HLPTR (queue *i*)
0xA0018+ i*0x1000	RXDBL_MADDR_MSW[1] (queue *i*)
0xA001C+ i*0x1000	RXDBL_MADDR(_LSW) (queue *i*)
0xC0000+ i*0x1000	TXIO_CTRL (queue *i*)
0xC0004+ i*0x1000	TXIO_STAT (queue *i*)
0xC0008+ i*0x1000	TXIO_SRC_ID (queue *i*)
0xC0010+ i*0x1000	TXIO_TLPTR (queue *i*)
0xC0014+ i*0x1000	TXIO_HDPTR (queue *i*)
0xC0018+ i*0x1000	TXIO_MADDR_MSW[1] (queue *i*)
0xC001C+ i*0x1000	TXIO_MADDR(_LSW) (queue *i*)
0xE0000	RXIO_CTRL
0xE0004	RXIO_STAT
0xE0008	RXIO_LACCESS_MSW[1]
0xE000C	RXIO_LACCESS(_LSW)
0xE0010	RXIO_LADDR_MEM_ERR_MSW[1]
0xE0014	RXIO_LADDR_MEM_ERR(_LSW)
0xE0018	RXIO_CNTRS

Where *i* is (depending on the register):

- the index of the queue for each type, configurable with generics *g_no_io_tx*, *g_no_dbell_rx*, *g_no_dbell_tx*, *g_no_msg_rx* and *g_no_msg_tx* at compile time (from 0 up to 31)
- the index of the translation window (from 0 to 15)
- the index of the memory protection partition (from 0 up to 3 or 7, depending if *g_8_in_win* is set to false or true)

[1] This field is reserved when *CFG_AW = 32*

*Table 1648.*GRSRIO registers when *g_hyb_que* is set to true.

APB address offset	Register
0x00000	CORE_CAP
0x00004	CORE_RST
0x00008	TO_CONF
0x0000C	CORE_STAT
0x00010	TX_INT_STAT
0x00014	RXMSG_INT_STAT
0x0001C	RXDBL_INT_STAT
0x00024	RXIO_INT_STAT
0x00028	EXTDBL_CTRL
0x0002C	EXTDBL_FLTR
0x00030	EXTDBL_SRC_ID
0x00034	EXTDBL_DST_ID
0x00038	ADDRTRNSL_SHIFT
0x10000+ i*0x1000	TRNSLWIN_CONF (window *i*)
0x20000+ i*0x1000	MEMPRT_CONF (partition *i*)

*Table 1648.*GRSRIO registers when *g_hyb_que* is set to true.

APB address offset	Register
0x40000 + i*0x1000	TX_CTRL (queue i)
0x40004 + i*0x1000	TX_STAT (queue i)
0x40008 + i*0x1000	TX_SRC_ID (queue i)
0x40010 + i*0x1000	TX_TLPTR (queue i)
0x40014 + i*0x1000	TX_HDPTR (queue i)
0x40018 + i*0x1000	TX_MADDR_MSW[1] (queue i)
0x4001C + i*0x1000	TX_MADDR(_LSW) (queue i)
0x60000 + i*0x1000	RXMSG_CTRL (queue i)
0x60004 + i*0x1000	RXMSG_STAT (queue i)
0x60008 + i*0x1000	RXMSG_DST_ID (queue i)
0x6000C + i*0x1000	RXMSG_DST_MSK (queue i)
0x60010 + i*0x1000	RXMSG_TLPTR (queue i)
0x60014 + i*0x1000	RXMSG_HDPTR (queue i)
0x60018 + i*0x1000	RXMSG_MADDR_MSW[1] (queue i)
0x6001C + i*0x1000	RXMSG_MADDR(_LSW) (queue i)
0xA0000+ i*0x1000	RXDBL_CTRL (queue i)
0xA0004 + i*0x1000	RXDBL_STAT (queue *i*)
0xA0008 + i*0x1000	RXDBL_ID (queue *i*)
0xA000C+ i*0x1000	RXDBL_MSK (queue *i*)
0xA0010+ i*0x1000	RXDBL_TLPTR (queue *i*)
0xA0014+ i*0x1000	RXDBL_HLPTR (queue *i*)
0xA0018+ i*0x1000	RXDBL_MADDR_MSW[1] (queue *i*)
0xA001C+ i*0x1000	RXDBL_MADDR(_LSW) (queue *i*)
0xE0000	RXIO_CTRL
0xE0004	RXIO_STATUS
0xE0008	RXIO_LACCESS_MSW[1]
0xE000C	RXIO_LACCESS_LSW
0xE0010	RXIO_LADDR_MEM_ERR_MSW[1]
0xE0014	RXIO_LADDR_MEM_ERR(_LSW)
0xE0018	RXIO_CNTRS

The following registers are available only when *g_mecs* is set to true, otherwise they are reserved.

Table 1649. Additional GRSRIO registers available when g_mecs is set to true.

APB address offset	Register
0xF0000	RESERVED
0xF0004	Timestamp CAR
0xF0008	Timestamp Generator Status CSR
0xF000C	MECS Tick Interval CSR
0xF0010	RESERVED
0xF0014	MECS Next Timestamp MSW
0xF0018	MECS Next Timestamp LSW
0xF001C	RESERVED
0xF0020	MECS Implementation Specific Settings
from 0xF0024 to 0xF0030	RESERVED
0xF0034	Timestamp Generator MSW CSR
0xF0038	Timestamp Generator LSW CSR

84.5.1 General registers

Table 1650. CORE_CAP - Core Capability Register

31 27	26 22	21 17	16 12	11 7	6 4	3	2	1	0
NO_TX_HYB_MSG	NO_RX_MSG	NO_TX_DBL	NO_RX_DBL	NO_TX_IO	RESERVED	AXI	NP	HM	MS
*	*	*	*	*	0	*	*	*	*
r	r	r	r	r	r	r	r	r	r

31: 27 Number of hybrid transmission queues or transmission queues for Data Messages (NO_TX_HYB_MSG): Number of transmission queues in hybrid mode (when g_hyb_que is set to “true”) or for Data Messages (when g_hyb_que is set to false) available in the core.

26: 22 Number of reception queues for Data Messages (NO_RX_MSG): Number of reception queues for data messages available in the core.

21: 17 Number of transmission queues for Doorbell Messages (NO_TX_DBL): Number of transmission queues for doorbell available in the core, reads zero when in hybrid-queues mode.

16: 12 Number of reception queues for Doorbell Messages (NO_RX_DBL): Number of reception queues for doorbell messages available in the core.

11: 7 Number of transmission queues for IO operations (NO_TX_IO): Number of transmission queues for IO operations available in the core, reads zero when in hybrid-queues mode.

6: 4 RESERVED

3 AXI4 bus master: this bit is set to one in case an AXI4 bus master is used, otherwise if an AHB bus master is used this is set to 0.

2 Number of Partitions (NP): reads 1 if the core supports 8 memory protection partition (*g_8_in_win* = “true”), reads 0 if it supports just 4 of them.

1 Hybrid-queues Mode (HM): reads 1 if the core is instantiated in hybrid mode (*g_hyb_que* = “true”)

0 MECS Supported (MS): reads 1 if the core supports the MECS Time Synchronization Protocol (*g_mecs* = “true”)

Table 1651. CORE_RST - Core Reset Register

31	30 0
RS	RESERVED
0	0
rw	r

*Table 1651.*CORE_RST - Core Reset Register

31 GRSRIO Soft Reset (RS). Writing 1 resets the GRSRIO logical layer core. Always reads 0.

30: 0 RESERVED

*Table 1652.*TO_CONF - Timeout Configuration Register

31	0
TIMEOUT	
0xFFFFFFFF	
rw	

31: 0 Response-to-Request and Request-to-Response Timeout Value (TIMEOUT). The timeout value is expressed in clock cycles.

*Table 1653.*CORE_STAT - Core General Status Register

31 16	15 3	2	1	0
NUR	RESERVED	FE	UT	UR
0	0	0	0	0
r	r	wc	wc	wc

31: 16 Number of unexpected responses (NUR): number of received unexpected responses since last time UR was cleaned.

15: 3 RESERVED

2 Clock Domain Crossing FIFO Error (FE). This bit is set when the asynchronous FIFO overflows, which transfers credit information from the SRIO end point to the GRSRIO core. This is a fatal error that should only occur when the clock frequency of the GRSRIO is lower than the clock frequency of the SRIO end point user interface. Writing 1 clears this bit.

1 Unsupported Transaction (UT). This bit is set when the GRSRIO core receives an unsupported transaction. Writing 1 clears this bit.

0 Unexpected Response (UR). This bit is set when the GRSRIO core receives an unexpected response. Writing 1 clears this bit.

*Table 1654.*INT_TX/MSG_STAT - Data Message Transmission Queues or Hybrid Queue Interrupt Level 1 Register

31	0
TXMSGINT/TXINT	
0	
r	

31: 0 Interrupts in Message Transmission or Hybrid Queues (TXMSGINT/TXINT). A bit is set if an interrupt occurred in the message transmission queue (if g_hyb_que is set to false) or in the mixed queues (if g_hyb_que is set to true) with the index corresponding to the number of the bit. This field is read-only, the interrupts must be individually cleared in the respective TXMSG_STAT or TX_STAT.

*Table 1655.*INT_RXMSG_STAT - Data Message Reception Queues Interrupt Level 1 Register

31	0
RXMSGINT	
0	
r	

31: 0 Interrupts in Message Reception Queues (RXMSGINT). A bit is set if an interrupt occurred in the message reception queue with the index corresponding to the number of the bit. This field is read-only, the interrupts must be individually cleared for each queue in the respective RXMSG_STAT.

*Table 1656.*INT_TXDBL_STAT - Doorbell Message Transmission Queues Interrupt Level 1 Register

This register is available only when “g_hyb_que” is set to false.

31	0
TXDBELLINT	

*Table 1656.*INT_TXDBL_STAT - Doorbell Message Transmission Queues Interrupt Level 1 Register

This register is available only when “g_hyb_que” is set to false.

0
r

31: 0 Interrupts in Doorbell Transmission Buffers (TXDBELLINT). A bit is set if an interrupt occurred in the doorbell transmission buffer with the index corresponding to the number of the bit. This field is read-only, the interrupts must be individually cleared in the respective TXDBELLMSG1.

*Table 1657.*INT_RXDBL_STAT- Doorbell Messages Reception Queues Interrupt Level 1 Register

31	0
RXDBLINT	
0	
r	

31: 0 Interrupts in Doorbell Reception Buffers (RXDBLINT). A bit is set if an interrupt occurred in the doorbell reception buffer with the index corresponding to the number of the bit. This field is read-only, the interrupts must be individually cleared in the respective RXDBL_STAT.

*Table 1658.*INT_TXIO_STAT - IO Operations Transmission Queues Interrupt Level 1 Register - (0x00020)

This register is available only when g_hyb_que is set to “false”

31	0
TXIOINT	
0	
r	

31: 1 RESERVED

0 Interrupt in I/O operation transmission queue (TXIOINT). A bit is set if an interrupt occurred in the IO transmission queue with the index corresponding to the number of the bit. This field is read-only, the interrupts must be individually cleared in the respective RXDBL_STAT.

*Table 1659.*INT_RXIO_STAT - IO Operations Reception Queues Interrupt Level 1 Register

31	1	0
RESERVED		RI
0		0
r		r

31: 1 RESERVED

0 Interrupt in I/O operation reception unit (RI). This field is read-only, the interrupts must be cleared in the respective RXIO_STAT.

*Table 1660.*EXTDBL_CONF - External Doorbell Configuration Register

31	16	15	14	13	12	11	10	9	8	7	5	4	3	2	1	0
RCOUNT		RES		TT		VC	CR	PRIO		RESERVED		IB	IFB	DC	DO	DI
0		0		0		0	0	0		0		0	0	0	0	0
rw		r		rw		rw	rw	rw		r		rw	rw	rw	rw	rw

31: 16 External Outbound Doorbell Message Retry Count (RCOUNT). External outbound doorbell messages are automatically retried until the destination node either accepts the message or until the user-defined retry threshold level in this field is reached.

15: 14 RESERVED

Table 1660. EXTDBL_CONF - External Doorbell Configuration Register

13: 12	External Outbound Doorbell Message Transport Type (TT). If set to 0b00, the destination ID and source IDs of external outbound doorbell messages are 8-bit wide. If set to 0b01, the fields are 16-bit wide.
11	External Outbound Doorbell Message Virtual Channel Bit (VC). If set, the virtual channel bit of the external outbound doorbell message packet is also set.
10	External Outbound Doorbell Message Critical Request Flow Bit (CR). If set, the CRF bit of the external outbound doorbell message is also set.
9: 8	External Outbound Doorbell Message Priority (PRIO). Priority of the external outbound doorbell message. To avoid deadlock, the highest priority level is not allowed for request packets. If 0b11 is written to this field, it is automatically overwritten by 0b10.
7: 5	RESERVED
4	Ignore Doorbell Buffer (IB). If this bit is set and DO is set and the doorbell info matches DBCOMP (in case DC is also set), the GRSRIO core outputs the doorbell to the external port without storing it to any possibly set up doorbell reception buffer and therefore always returns a DONE response to the destination node. If the bit is not set, the doorbell message is also outputted to the external port, however, the GRSRIO core still tries to store it in a doorbell reception buffer. Then, the response type depends on the doorbell reception buffer. If a buffer is available and the doorbell message was successfully stored, the GRSRIO core returns a DONE response, otherwise a RETRY response.
3	Ignore Full Doorbell Buffer (IFB). In case DO is set and IB is not set but no free doorbell reception buffer is available (see above), the GRSRIO core always returns a DONE response.
2	Enable Doorbell Info Comparison (DC). If set and if DO is set, inbound doorbell messages are only forwarded to the external port if the doorbell info field matches the value in the DBCOMP field.
1	Enable Doorbell Output Signals (DO). If set, inbound doorbell messages are forwarded to the external port.
0	Enable Doorbell Input Signals (DI). If set, outbound doorbell messages can be generated through the external port.

Table 1661. EXTDBL_FLTR - Processing Core Configuration Register 2

31 16	15 0
DBMASK	DBCOMP
0	0
rw	rw

31: 16	Doorbell Message Mask Value (DBMASK). If DC and DO are set, inbound doorbell messages are compared with the compare value DBCOMP. If a bit is set in this mask field, the corresponding bit of DBCOMP is compared.
15: 0	Doorbell Message Compare Value (DBCOMP). If DC and DO are set, the doorbell message is only forwarded to the external port if the doorbell info field matches the value in this field.

Table 1662. EXTDBL_SRC_ID - Processing Core Configuration Register 4

31 0
SRC_ID
0
rw

31: 0	External Outbound Doorbell Message Destination ID (DEST_ID). Destination ID of all outbound doorbell messages generated through the external port.

Table 1663. EXTDBL_DST_ID - Processing Core Configuration Register 4

31 0
DST_ID
0
rw

31: 0	External Outbound Doorbell Message Destination ID (DEST_ID). Destination ID of all outbound doorbell messages generated through the external port.

Table 1664. ADDRTRNSL_SHIFT - Address Shift for Address Translation Register

31 4	3 0
RESERVED	ADDR_SHIFT
0	0
r	rw

31: 4 RESERVED

3: 0 ADDR_SHIFT: Address Shift for Address Translation: number of position the translated bits are right-shifted (when SHIFT = 0 the translated bits are placed in Physical Address (MSB: MSB-3))

Table 1665. TRNSLWIN_CONF- Translation Window Configuration Register

31 20	19 16	15 1	0
RESERVED	WIN_ADDR	RESERVED	TEN
0	0	0	0
r	rw*	r	rw*

31: 20 RESERVED

19: 16 Values of the translated 4 bits of the physical base address for this window. Writing to this field has only an effect when memory I/O operations are not enabled, that is, when EN is 0.

19: 16 RESERVED

0 Enable This Translation Window (TEN). Writing to this field has only an effect when memory I/O operations are not enabled, that is, when EN is 0.

Table 1666. MEMPRT_CONF - Configuration Register for Partition *i*

31	30	29 28	27 16	15 12	11 0
PRE	PWE	RES	PART_ADDR	RESERVED	PART_MASK
0	0	0	0	0	0
rw	rw	r	rw*	r	rw*

31 Enable Read for the partition (PRE): Enable Read from the partition

30 Enable Write for the partition (PWE): Enable Write in the partition

27: 16 Memory Partition Address (PART_ADDR). These 12 bits are compared to the most significant 12 bits of the corresponding physical address of the memory I/O operation address. Writing to this field has only an effect when the reception of memory I/O operations is not enabled, that is, when RXIO_CONF.EN is 0.

15: 12 RESERVED

11: 0 Memory Partition Mask (PART_MASK). The mask can be used to adjust the size of the memory partition. An incoming I/O operation's address falls into this memory partition if the following condition is true: (SRIO Address (MSB:MSB-11) XOR PART1_ADDR) AND PART1_MASK = 0x000. Writing to this field has only an effect when memory I/O operations are not enabled, that is, when EN is 0.

84.5.2 Transmission Queues Registers

Table 1667. TXMSG_CTRL/TXDBL_CTRL/TXIO_CTRL/TX_CTRL - Transmission queue configuration register

31 22	21 20	19	18	17 16	15 10	9 6	5	4	3	2	1	0
RESERVED	PRIO	CR	VC	TT	RESERVED	CBD	IM	IE	IT	ST	TP	EN
0	0	0	0	0	0	0	0	0	0	0	0	0
r	rw	rw	rw	rw	r	rw*	rw	rw	rw	rw	rw*	rw*

31: 22 RESERVED

21: 20 Priority (PRIO). Priority of the outbound message or IO operation packets. To avoid deadlock, the highest priority level is not allowed for request packets (if 0b11 is written to this field, it is automatically overwritten by 0b10)

19 Critical Request Flow Bit (CR). If set, the CRF bit of the outbound message or IO operation packet is also set.

18 Virtual Channel Bit (VC). If set, the virtual channel bit of the outbound message packet is also set.

17: 16 Transport Type (TT). If set to 0b00, the destination ID and source IDs of this transmission queue are 8-bit wide. If set to 0b01, the fields are 16-bit wide.

15: 10 RESERVED

9: 6 Circular Buffer Depth (CBD). Depth of the circular queue of this message transmission queue. A queue can be configured to store between 2 and 65,536 messages: Depth = 2^(CBD+1). Writing to this field has only an effect when the transmission queue is not enabled, that is, when EN is 0. Note that even if the size of the queue is 2^(CBD+1), setting the head point directly to this value will not enable the queue as only the least significant CBD+1 bits of the tail pointer will be compared to the CBD+1 least significant bit of the head pointer.

5 Transmission Interrupt Mode (IM). If set, a transmission interrupt is only generated after the transmission queue became empty, that is, when the tail pointer equals the head pointer. If not set, the generation of transmission interrupts depends on the setting of the corresponding TI bits in the transmission descriptor words.

4 Enable Error Interrupts (IE). If set, interrupts are generated for any kind of transmission or bus error related to this transmission queue.

3 Enable Transmission Interrupts (IT). If set, interrupts are generated for successful message or IO operation packets transmission. The way how the interrupts are generated depends on the Transmission Interrupt Mode (IM).

2 Stop on Transmission Error (ST). If set, the transmission queue is automatically disabled after a transmission error has occurred. The tail pointer is not incremented and is therefore pointing to the descriptor that was processed during the occurrence of the error, allowing the software to quickly initiate some error recovery procedure. The type of transmission error can be read from the descriptor and includes: Serial RapidIO Error, Retry Error, and Timeout Error. If not set, the DMA engine increments the tail pointer and processes the next descriptor (if available) in case of errors.

1 Enable two-outstanding-packets mode: if this bit is set to 1, when a single descriptor defines more than one packet, two packet will be transmitted without waiting for the reply to the first one. Writing to this field has only an effect when the transmission queue is not enabled, that is, when EN is 0.

0 Enable Transmission Queue (EN). If set, the DMA engine executes the operations defined in the descriptors queue until the circular buffer is empty, that is, until the tail pointer reaches the head pointer. Then, once the software increments the head pointer again, the transmission is automatically resumed. Writing 1 while the queue is disabled, enables the queue. Writing 0 while the queue is enabled disables the transmission queue. If a transmission is ongoing, the queue is disabled after processing the current descriptor. When disabled and after all memory accesses in conjunction with this descriptor have been completed, this bit clears to 0.

Table 1668. TXMSG_STAT/TXDBL_STAT/TXIO_STAT/TX_STAT - Transmission Queue Status Register

31 8	7	6	5	4	3	2	1	0
RESERVED	SR	SS	SW	TI	EI	BE	TE	TA
0	0	0	0	0	0	0	0	0
r	r	r	r	wc	wc	wc	wc	r

Table 1668. TXMSG_STAT/TXDBL_STAT/TXIO_STAT/TX_STAT - Transmission Queue Status Register

7	Two packet sent and one response received (SR): reads 1 when two packet have been sent from a descriptor defining the transmission of more packets and one response has been already received (in two-outstanding-packets mode) (two-outstanding-packets mode)
6	Two packets sent and no responses received (SS): reads 1 when two packet have been sent from a descriptor defining the transmission of more packets and no responses have been received (in two-outstanding-packets mode)
5	One packet sent and one waiting to be sent (SW): reads 1 when one packet has been sent from a descriptor defining the transmission of more packets and the second one is still waiting to be transmitted (in two-outstanding-packets mode)
4	Transmission Interrupt (TI). Indicates a transmission interrupt. Writing 1 clears this bit and the corresponding bit in the Interrupt Level 1 register.
3	Error Interrupt (EI). Indicates an error interrupt. Writing 1 clears this bit and the corresponding bit in the Interrupt Level 1 register.
2	Bus Error (BE). Set when a bus memory access error occurs while a descriptor in this queue is processed. The transmission queue is automatically disabled and the tail pointer is not incremented. Writing 1 clears the bit, writing 0 has no effect.
1	Transmission Error (TE). Set when a transmission error occurs while a descriptor in this queue is processed. The type of transmission error can be read from the descriptor and includes: Serial RapidIO Error, Retry Error, and Timeout Error. Writing 1 clears the bit, writing 0 has no effect.
0	Transmission Active (TA). Indicates that a transmission is ongoing. This bit remains high as long as the GRSRIO core has not closed the current descriptor.

Table 1669. TXMSG_SRC_ID/TXDBL_SRC_ID/TXIO_SRC_ID/TX_SRC_ID - Transmission Queue Source ID Register

31	0
SOURCE_ID	
0	
rw	

31: 0	Source Identifier (SOURCE_ID). Source ID of this transmission queue.

Table 1670. TXMSG_TLPTR/TX_TLPTR/TXDBL_TLPTR/TXIO_TLPTR - Transmission Queue Tail Pointer Register

31	16	15	0
RESERVED		TLPTR	
0		0	
r		rw*	

31: 16	RESERVED
15: 0	Message Transmission Queue Tail Pointer (TLPTR). The tail pointer is updated by the DMA engine and is therefore read-only during operation. It is incremented by one for each message that was successfully committed to the Serial RapidIO port. Writing to this field has only an effect when the transmission queue is not enabled, that is, when EN is 0. The software can flush a disabled queue by programming both the tail and head pointer to point to the same descriptor.

Table 1671. TXMSG_HDPTR/TXDBL_HDPTR/TXIO_HDPTR/TX_HDPTR - Transmission Queue Head Pointer

31	16	15	0
TLPTR_R		HDPTR	
0		0	
r		rw	

31: 16	Transmission Queue Tail Pointer (READ ONLY) (TLPTR_R): read only value of the tail pointer to allow the SW to know both head and tail pointer with just one access and take faster decisions.
15: 0	Transmission Queue Head Pointer (HDPTR). The head pointer is updated by software after setting up the messages or IO operations payload in memory and the corresponding descriptors in the circular buffer.

Table 1672. TXMSG_MADDR_MSW)/TXDBL_MADDR_MSW/TXIO_MADDR_MSW/TX_MADDR_MSW - Transmission Queue Most Significant Word Start Address Register

This register is available only when CFG_AW is set to 64.

31	0
MADDRESS_MSW	
0	
rw*	

31: 0 Most Significant Word of the Start Memory Address (MADDRESS_MSW). Address of the start of the queue containing the transmission descriptors. Writing to this field has only an effect when the transmission queue is not enabled, that is, when EN is 0. This register is reserved when *CFG_AW = 32*.

Table 1673. TXMSG_MADDR(_LSW)/TXDBL_MADDR(_LSW)/TXIO_MADDR(_LSW)/TX_MADDR(_LSW) - Transmission Queue Most Significant Word Start Address Register

31	0
MADDRESS(_LSW)	
0	
rw*	

31: 0 Least Significant Word of the Start Memory Address (MADDRESS(_LSW)). Address of the start of the queue containing the transmission descriptors. When *CFG*_AW = 64, this field indicates the LSW of such address. Writing to this field has only an effect when the transmission buffer is not enabled, that is, when EN is 0.

84.5.3 Message Reception Queues Register

*Table 1674.*RXMSG_CTRL - Message Reception Control Register, Queue i

31 23	22	21 16	15 10	9 6	5	4	3	2	1	0
MAX_SIZE	R	MBOXMSK	MBOX	CBD	IF	IE	IR	ST	R	EN
0	0	0	0	0	0	0	0	0	0	0
rw*	r	rw*	rw*	rw*	rw	rw	rw	rw	r	rw*

31: 23 Maximum allowed message size (MAX_SIZE). Only messages with a size up to MAX_SIZE are accepted in this queue. Maximum size of the message in multiples of 8 bytes: Maximum Size = (MAX_SIZE+1)*8 bytes. Writing to this field has only an effect when the reception queue is not enabled, that is, when EN is 0.

22 RESERVED

21: 16 Mailbox Mask (MBOXMSK). Using this mask field, ranges of mailbox numbers can be accepted. If a bit is set, the corresponding bit of the mailbox field MBOX is compared. The example in the data-sheet is wrong: the formula to accept a packet is: (RECEIVED ID XOR QUEUE ID) AND MASK = b000000. This means that to accept mailboxes from 0 to 3, ID must be set to b000000 and MASK must be b111100.

15: 10 Mailbox Number (MBOX). Only messages addressed to the here defined mailbox number are accepted. Writing to this field has only an effect when the reception queue is not enabled, that is, when EN is 0.

9: 6 Circular Buffer Depth (CBD). Depth of the circular buffer of this message reception queue. A queue can be configured to store between 2 and 65,536 messages: Depth = 2^(CBD+1). Writing to this field has only an effect when the reception queue is not enabled, that is, when EN is 0.

5 Enable Full Interrupts (IF). If set, an interrupt is generated each time the reception queue becomes full, that is, when also the FQ bit is set.

4 Enable Error Interrupts (IE). If set, interrupts are generated in case of reception errors or bus errors.

3 Enable Reception Interrupts (IR). If set, interrupts are generated for successful message receptions. Reception interrupts must be enabled for each descriptor individually by setting the corresponding RI bits in the message reception descriptors.

2 Stop on Reception Error (ST). If set, the reception queue is automatically disabled after a reception error has occurred. The tail pointer is not incremented and is therefore pointing to the descriptor that was processed during the occurrence of the error, allowing the software to quickly initiate some error recovery procedure. If not set, the DMA engine increments the tail pointer and processes the next descriptor (if available) in case of errors.

1 RESERVED

0 Enable Reception Queue (EN). If set, the DMA engine executes the reception of messages until the circular buffer is full, that is, until the address of the tail pointer equals the address of the head pointer - 1 (modulo buffer depth). Then, once the software increments the head pointer again, the reception is automatically resumed. Writing 1 while the queue is disabled, enables the queue. Writing 0 while the queue is enabled disables the reception queue. If a reception is ongoing, the queue is disabled after processing the current descriptor. When disabled and after all memory accesses in conjunction with this descriptor have been completed, this bit clears to 0.

*Table 1675.*RXMSG_STAT - Message Reception Status Register, Queue i

31 8	7	6	5	4	3	2	1	0
RESERVED	RI	EI	FI	WL	FQ	BE	RE	RA
0	0	0	0	0	0	0	0	0
r	wc	wc	wc	wc	wc	wc	wc	wc

31: 8 RESERVED

7 Reception Interrupt (RI). Indicates a reception interrupt. Writing 1 clears this bit and the corresponding bit in the Interrupt Level 1 register.

6 Error Interrupt (EI). Indicates an error interrupt. Writing 1 clears this bit and the corresponding bit in the Interrupt Level 1 register.

Table 1675. RXMSG_STAT - Message Reception Status Register, Queue i

5	Full Interrupt (FI). Indicates a full interrupt. Writing 1 clears this bit and the corresponding bit in the Interrupt Level 1 register.
4	Wrong Letter (WL). Set when a message with wrong letter number is received during the reception of a multi-packet message. Writing 1 clears the bit, writing 0 has no effect.
3	Full Queue (FQ). Set when the reception queue becomes full, that is, when the address of the tail pointer equals the address of the head pointer - 1 (modulo buffer depth). Writing 1 clears the bit, writing 0 has no effect.
2	Bus Error (BE). Set when a bus memory access error occurs while a descriptor in this queue is processed. The reception queue is automatically disabled and the tail pointer is not incremented. Writing 1 clears the bit, writing 0 has no effect.
1	Reception Error (RE). Set when a reception error occurs while a descriptor in this queue is processed. The type of reception error can be read from the descriptor and includes: Serial RapidIO Error and Truncation Error. Writing 1 clears the bit, writing 0 has no effect.
0	Reception Active (RA). Indicates that a reception is ongoing. This bit remains high as long as the GRSRIO core has not closed the current descriptor.

Table 1676. RXMSG_DEST_ID - Message Reception Acceptance ID Register, Queue i

31	0
DEST_ID_ACC	
0	
rw*	

31: 0	Accepted Destination ID (DEST_ID_ACC). Only messages addressed to the here defined destination ID are accepted. Writing to this field has only an effect when the reception queue is not enabled, that is, when EN is 0.

Table 1677. RXMSG_DEST_ID - Message Reception Mask ID Register, Queue i

31	0
DEST_ID_MASK	
0	
rw*	

31: 0	Accepted Destination ID Mask (DEST_ID_MASK). Using this mask field, ranges of destination IDs can be accepted. If a bit is set, the corresponding bit of the accepted destination ID field DEST_ID_ACC is compared. Writing to this field has only an effect when the reception queue is not enabled, that is, when EN is 0.

Table 1678. RXMSG_TLPTR - Message Reception Queue Tail Pointer Register, Queue i

31	16	15	0
RESERVED		TLPTR	
0		0	
r		rw*	

31: 16	RESERVED
15: 0	Message Reception Queue Tail Pointer (TLPTR). The tail pointer is updated by the DMA engine and is therefore read-only during operation. It is incremented by one for each message that was successfully stored to memory. Writing to this field has only an effect when the reception queue is not enabled, that is, when EN is 0. The software can flush a disabled queue by programming both the tail and head pointer to point to the same descriptor.

*Table 1679.*RXMSG_HDPTR - Message Reception Queue Head Pointer Register, Queue i

31 16	15 0
TLPTR_R	HDPTR
0	0
r	rw

31: 16 Message Reception Queue Tail Pointer (READ ONLY) (TLPTR_R): read only value of the tail pointer to allow the SW to know both head and tail pointer with just one access and take faster decisions.

15: 0 Message Reception Queue Head Pointer (HDPTR). The head pointer is updated by software after reserving the memory space for incoming messages and after setting up the descriptors in the circular buffer. The head pointer must be increased by one for each message that shall be received.

*Table 1680.*RXMSG_MADD_MSW - Message Reception Most Significant Word of the Start Address, Queue *i*

This register is only available when CFG_AW = 64.

31 0
MADDRESS_MSW
0
rw*

31: 0 Most Significant Word of the Reception Queue Start Memory Address (MADDRESS_MSW). Address pointing to the start of the memory block storing the descriptors of this queue. Writing to this field has only an effect when the reception queue is not enabled, that is, when EN is 0.

*Table 1681.*RXMSG_MADDR(_LSW) - Message Reception Least Significant Word of the Start Address, Queue *i*

31 0
MADDRESS(_LSW)
0
rw*

31: 0 Least Significant Word of the Reception Queue Start Memory Address (MADDRESS(_LSW)). Address pointing to the start of the memory block storing the descriptors of this queue. When CFG_AW is set to 64, it is the LSW of such address. Writing to this field has only an effect when the reception queue is not enabled, that is, when EN is 0.

84.5.4 Doorbell Message Reception Buffer Register

Table 1682. RXDBELL_CTRL - Doorbell Message Reception Configuration Register, Queue i

31 30 29 28 27 26 25 24 23 10	9 6	5	4	3	2 1	0
RESERVED	CBD	IF	IE	IR	RES	EN
0	0	0	0	0	0	0
r	rw*	rw	rw	rw	r	rw*

31: 10 RESERVED

9: 6 Circular Buffer Depth (CBD). Depth of this doorbell message reception buffer. A buffer can be configured to store between 2 and 65,536 doorbell messages: Depth = 2^(CBD+1). Writing to this field has only an effect when the reception buffer is not enabled, that is, when EN is 0.

15: 10 RESERVED

5 Enable Full Interrupts (IF). If set, an interrupt is generated each time the reception buffer becomes full, that is, when also the FB bit is set.

4 Enable Error Interrupts (IE). If set, interrupts are generated in case of bus errors.

3 Enable Reception Interrupts (IR). If set, interrupts are generated for successful doorbell message receptions. Reception interrupts must be enabled for each doorbell message individually by setting the corresponding RI bits in the reception buffer words.

2: 1 RESERVED

0 Enable Reception Buffer (EN). If set, the DMA engine executes the reception of doorbell messages until the circular buffer is full, that is, until the address of the tail pointer equals the address of the head pointer - 1 (modulo buffer depth). Then, once the software increments the head pointer again, the reception is automatically resumed. Writing 1 while the buffer is disabled, enables the buffer. Writing 0 while the buffer is enabled disables the reception buffer. If a reception is ongoing, the buffer is disabled after processing the current doorbell message. When disabled and after all memory accesses to this buffer entry have been completed, this bit clears to 0.

Table 1683. RXDBELLMSG_STAT - Doorbell Message Reception Status Register, Queue i

31 6	5	4	3	2	1	0
RESERVED	RI	EI	FI	FB	BE	RA
0	0	0	0	0	0	0
r	wc	wc	wc	wc	wc	r

5 Reception Interrupt (RI). Indicates a reception interrupt. Writing 1 clears this bit and the corresponding bit in the Interrupt Level 1 register.

4 Error Interrupt (EI). Indicates an error interrupt. Writing 1 clears this bit and the corresponding bit in the Interrupt Level 1 register.

3 Full Interrupt (FI). Indicates a full interrupt. Writing 1 clears this bit and the corresponding bit in the Interrupt Level 1 register.

2 Full Buffer (FB). Set when the reception buffer becomes full, that is, when the address of the tail pointer equals the address of the head pointer - 1 (modulo buffer depth). Writing 1 clears the bit, writing 0 has no effect.

1 Bus Error (BE). Set when a bus memory access error occurs while a doorbell message in this buffer is processed. The reception buffer is automatically disabled and the tail pointer is not incremented. Writing 1 clears the bit, writing 0 has no effect.

0 Reception Active (RA). Indicates that a reception is ongoing. This bit remains high as long as the GRSRIO core has not closed the current buffer entry.

Table 1684. RXDBL_MSK - Doorbell Reception Message Destination ID - Queue *i*

31	0
DEST_ID_ACC	
0	
rw*	

31: 0 Accepted Destination ID (DEST_ID_ACC). Only doorbell messages addressed to the here defined destination ID are accepted. Writing to this field has only an effect when the reception buffer is not enabled, that is, when EN is 0.

Table 1685. RXDBL_DST_MSK - Doorbell Reception Message ID Mask - Queue *i*

31	0
DEST_ID_MASK	
0	
rw*	

31: 0 Accepted Destination ID Mask (DEST_ID_MASK). Using this mask field, ranges of destination IDs can be accepted. If a bit is set, the corresponding bit of the accepted destination ID field DEST_ID_ACC is compared. Writing to this field has only an effect when the reception buffer is not enabled, that is, when EN is 0.

Table 1686. RXDBELL_TLPTR - Doorbell Message Reception, Queue i, Register 4

31	16	15	0
RESERVED		TLPTR	
0		0	
r		rw*	

31: 16 RESERVED

15: 0 Doorbell Message Reception Buffer Tail Pointer (TLPTR). The tail pointer is updated by the DMA engine and is therefore read-only during operation. It is incremented by one for each doorbell message that was successfully stored to memory. Writing to this field has only an effect when the reception buffer is not enabled, that is, when EN is 0. The software can flush a disabled buffer by programming both the tail and head pointer to point to the same buffer entry.

Table 1687. RXDBELL_HDPTR - Doorbell Message Reception, Queue i, Register 5

31	16	15	0
TLPTR_R		HDPTR	
0		0	
r		rw	

31: 16 Doorbell Message Reception Queue Tail Pointer (READ ONLY) (TLPTR_R): read only value of the tail pointer to allow the SW to know both head and tail pointer with just one access and take faster decisions.

15: 0 Doorbell Message Reception Queue Head Pointer (HDPTR). The head pointer is updated by software after setting up the buffer entries in the circular buffer. The head pointer must be increased by one for each doorbell message that shall be received.

*Table 1688.*RXDBL_MADDR_MSW - Doorbell Message Reception, Queue i

This register is only available when CFG_AW = 64.

31	0
MADDRESS_MSW	
0	
rw*	

31: 0 Most Significant Word of the Reception Buffer Start Memory Address (MADDRESS_MSW). Address pointing to the start of the memory block reserved for the reception of doorbell messages. Writing to this field has only an effect when the reception buffer is not enabled, that is, when EN is 0.

*Table 1689.*RXDBL_MADDR(_LSW) - Doorbell Message Reception, Queue i

31	0
MADDRESS(_LSW)	
0	
rw*	

31: 0 (Least Significant Word of the) Start Memory Address of the Doorbell Queues (MADDRESS_LSW). Address pointing to the start of the memory block reserved for the reception of doorbell messages. Writing to this field has only an effect when the reception buffer is not enabled, that is, when EN is 0.

84.5.5 I/O Operation Reception Unit Registers

*Table 1690.*RXIO_CONF - I/O Operation Reception Configuration Register

31	3	2	1	0
RESERVED		IE	IM	EN
0		0	0	0
r		rw	rw	rw*

31: 3 RESERVED

2 Enable Error Interrupts (IE). If set, interrupts are generated in case of bus errors or forbidden memory access errors.

1 Enable Memory Access Interrupts (IM). If set, interrupts are generated for successful memory I/O operations.

0 Enable Memory I/O (EN). If set, incoming memory I/O operations are executed by the DMA engine, otherwise packets are automatically rejected and an ERROR response is returned to the destination node (for those memory I/O operations that require a response). Writing 1 while memory I/O operations are disabled, enables them. Writing 0 while memory I/O operations are enabled disables them. If a memory access is ongoing, this access is allowed to finish. Then, once the access has been completed, this bit clears to 0.

*Table 1691.*RXIO_STAT - I/O Operation Reception Status Register

31	5	4	3	2	1	0
RESERVED		RI	EI	BE	ME	RA
0		0	0	0	0	0
r		wc	wc	wc	wc	r

31: 5 RESERVED

Table 1691. RXIO_STAT - I/O Operation Reception Status Register

4	Reception Interrupt (RI). Indicates a reception interrupt. Writing 1 clears this bit and the corresponding bit in the Interrupt Level 1 register.
3	Error Interrupt (EI). Indicates an error interrupt. Writing 1 clears this bit and the corresponding bit in the Interrupt Level 1 register.
2	Bus Error (BE). Set when a bus memory access error occurs while an inbound I/O operation is processed. The I/O reception unit is automatically disabled. Writing 1 clears the bit, writing 0 has no effect.
1	Forbidden Memory Access Error (ME). Set when an inbound I/O operation tries to access a forbidden memory address. Writing 1 clears the bit, writing 0 has no effect.
0	Reception Active (RA). Indicates that a reception is ongoing. This bit remains high as long as the GRSRIO core has not finished the memory access.

Table 1692. RXIO_LACCESS_MSW - Most Significant Word of the Last Access Register

This register is only available when *CFG_AW* is set to 64.

31	0
LACCESS_MSW	
0	
r	

31: 0	Most Significant Word of the Last Accessed Address Causing a Memory Error(LACCESS_MSW). The last accessed memory address can be read from this field.

Table 1693. RXIO_LACCESS(_LSW) - (Least Most Significant Word of the Last Access Register

31	0
LACCESS(_LSW)	
0	
r	

31: 0	Last Accessed Address (LACCESS(_LSW)). The last accessed memory address can be read from this field. When *CFG_AW = 64* this is the LSW of such address.

Table 1694. RXIO_LACCESS(_LSW) - (Least Most Significant Word of the Last Access Register

This register is only available when *CFG_AW* is set to 64.

31	0
LAST_MEM_ERR_ADDR_MSW	
0	
r	

31: 0	Last Accessed Address Causing a Memory Error (LACCESS(_LSW)). The last MSW of the last accessed memory address causing a memory error can be read from this field.

Table 1695. RXIO_LACCESS(_LSW) - (Least Most Significant Word of the Last Access Register

31	0
LAST_MEM_ERR_ADDR(_LSW)	
0	

*Table 1695.*RXIO_LACCESS(_LSW) - (Least Most Significant Word of the Last Access Register

r

31: 0 Last Accessed Address (LACCESS(_LSW)). The last accessed memory address causing a memory error can be read from this field. When *CFG_AW = 64* this is the LSW of such address.

*Table 1696.*RXIO_CNTRS - Reception Unit Events Counters Register

31 16	15 0
NME	NRI
0	0
r	r

31: 16 Number of Memory Error (NME): Number of memory errors occurred since last time RXIO_STAT.ME was cleaned.

15: 0 Number of Reception Interrupts Error (NRI): Number of reception interrupts occurred since last time RXIO_STAT.RI was cleaned.

84.5.6 MECS registers

The MECS registers are implemented according to RapidIO specification, Rev. 4.0 and only if *g_mecs* is set to “true”. Timestamp Generation Extension Block Header is not implemented, as MECS registers are not implemented together with the other CAR and CSR registers in the SRIOGEN2IP.

*Table 1697.*Timestamp CAR (T_CAR)- offset 0xF004

31 6	5	4	3	2	1	0
RESERVED	SS	MMS	MSS	CFS	TMS	TSS
0	0	1	1	0	0	0
r	r	r	r	r	r	r

31: 1 RESERVED

5 SMECS Supported: not supported (0)

4 MECS Master Supported: supported (1)

3 MECS Slave Supported: supported (1)

2 Common Clock Frequency supported (CFS): not supported (0)

1 Timestamp Master Supported (TMS): not supported (0)

0 Timestamp Slave Supported (TSS): not supported (0)

*Table 1698.*Timestamp Generator Status CSR (TGS_CSR) - offset 0xF008

31 4	3	2	1	0
RESERVED	WST	ST	R	CL
0	0	0	0	0
r	wc	r	r	r

31: 4 RESERVED

Table 1698. Timestamp Generator Status CSR (TGS_CSR) - offset 0xF008

3	Timestamp Generator Was Stopped: Indicates if the Timestamp Generator counter has not advanced because it has been set to an earlier time. 0b0 - Timestamp Generator has advanced continuously since this bit was last cleared 0b1 - Timestamp Generator has temporarily stopped advancing at least once since this bit was last cleared. This bit is cleared by writing "1" to it.
2	Timestamp Generator Stopped: Indicates if the Timestamp Generator counter is not advancing because it is being set to an earlier time. 0b0 - Timestamp Generator is advancing 0b1 - Timestamp Generator is temporarily not advancing
1	RESERVED
0	Timestamp Generator Clock Locked (CL): Indicates whether the Timestamp Generator counter is operating from a good clock source. 0b0 - Timestamp Generator is not operating with a good clock source. 0b1 - Timestamp Generator is operating with a good clock source.

Table 1699. MECS Tick Interval CSR (MTI_CSR) - (0xF00C)

31 8	7	6	5	4	3	2	1	0
TICK_INT	RES.		LS	LT	LSTHR		SS	SR
0	0		0	0	00		0	0
rw	r		wc	wc	rw		r	rw

31: 8	Tick Interval (TICK_INT): For a MECS Master, a MECS shall be sent when time has advanced by this many nanoseconds. For an MECS Slave, time has advanced by this many nanoseconds whenever an MECS is received. MECS transmission, and MECS timestamp synchronization for received MECS, is disabled when this register is 0.
7: 6	RESERVED
5	Lost TSG Sync Error Status: This field indicates that the device has detected at least "Lost TSG Sync Error Threshold" consecutive ticks have been lost. 0 - A Lost TSG Sync Error has not been detected 1 - A Lost TSG Sync Error has been detected This bit must be written with 1 to be cleared.
4	Lost Tick Error Status 0 This field indicates if the device has detected at least one lost tick. 0 - A Lost Tick Error has not been detected 1 - A Lost Tick Error has been detected This bit must be written with 1 to be cleared.
3: 2	Lost TSG Sync Error Threshold: controls the number of MECS/SMECS "ticks" that must be lost before declaring the timestamp generator to be out of sync. 0b00 - Lost Tick Error Threshold is disabled 0b01 - If one tick is lost, declare the timestamp generator out of sync 0b10 - If two ticks are lost, declare the timestamp generator out of sync 0b11 - If three ticks are lost, declare the timestamp generator out of sync
1	SMECS Selection (SS): The device uses MECS and doesn't support SMECS (0)
0	MECS Time synchronization Role: Controls whether a device operates as a MECS Master or MECS Slave. 0 - The device is operating as a MECS Slave 1 - The device is operating as a MECS Master

*Table 1700.*MECS Next Timestamp MSW CSR (NMSW_CSR) - offset 0xF0014

31	0
NMSW	
0	
rw	

31: 0 Most significant 32 bits for the timestamp value used to update the Timestamp Generator MSW CSR when a Multicast Event Control Symbol is received by an MECS Slave.

Most significant 32 bits of the timestamp value compared with the Timestamp Generator value to determine when a Multicast Event Control Symbol must be transmitted by an MECS Master

*Table 1701.*MECS Next Timestamp LSW CSR (NLSW_CSR) - (0xF0018)

31	0
NLSW	
0	
rw	

31: 0 Least significant 32 bits for the timestamp value used to update the Timestamp Generator LSW CSR when a Multicast Event Control Symbol is received by an MECS Slave.

Least significant 32 bits of the timestamp value compared with the Timestamp Generator value to determine when a Multicast Event Control Symbol shall be transmitted by an MECS Master

*Table 1702.*MECS Implementation Specific Settings Register (MECS_S) - (0xF0020)

31	30 28	27 16	15 11	10 8	7 0
EN	RES.	NS_LM	RESERVED	CMD	NS_CC
0	0	0xFFF	0	0	0
rw	r	rw	r	rw	rw

31 MECS Time Synchronization Protocol enabled when this field is set to 1.

30: 28 RESERVED

26: 16 Nanoseconds required to assume the MECS tick lost (NS_LM): when the MECS is set as a slave, the core assumes that the MECS tick is lost when this amount of nanoseconds is elapsed after the local timer reaches TICK_INT. When a tick is lost the Next Timestamp is incremented of MTI_CSR.TICK_INT.

15: 11 RESERVED

8: 10 CMD of the Enhanced MECS (CMD): when the node is a MECS master, this field indicates the CMD of the Enhanced MECS to be transmitted. When the node is a MECS slave, this field indicates the CMD of the last MECS received, all values are considered valid ticks.

0: 7 Increment of local timer in nanoseconds every clock cycles: the Timestamp Generator will be incremented of NS_CC every clock cycles. This field must be set accordingly to the frequency of the systems (some examples below)

*Table 1703.*MECS Timestamp Generator MSW CSR (MSW_CSR)- offset 0xF0034

31	0
MSW	
0	
rw	

31: 0 MSW Bits: Most significant 32 bits for the timestamp generator.

*Table 1704.*MECS Timestamp Generator LSW CSR (LSW_CSR)- offset 0xF0038

31	0
LSW	
0	
rw	

*Table 1704.*MECS Timestamp Generator LSW CSR (LSW_CSR)- offset 0xF0038

31: 0	LSW Bits: Least significant 32 bits for the timestamp generator.

84.6 External doorbell interface

84.6.1 Outbound doorbell messages

The following timing diagram illustrates how outbound doorbell messages can be generated through the external interface (if this feature is enabled):

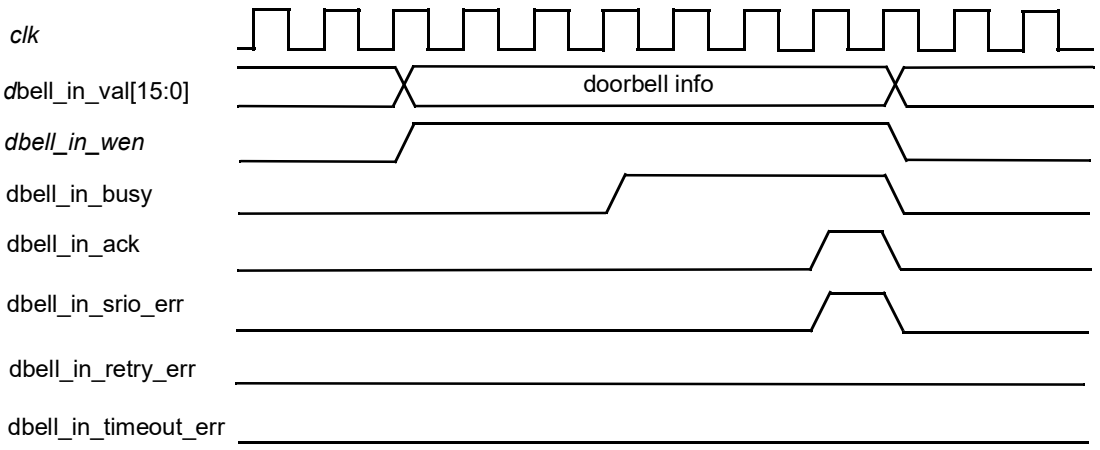


Figure 230. Skew and jitter timing waveforms

The doorbell info value must be provided on DBELL_IN_VAL and kept stable together with DBELL_IN_WEN asserted to 1 until the interface acknowledges the transmission by asserting DBELL_IN_ACK. In case of errors, DBELL_IN_SRIO_ERR, DBELL_IN_RETRY_ERR or DBELL_IN_TIMEOUT_ERR is asserted together with DBELL_IN_ACK (in the example above a SRIO error occurred during transmission, that is, an ERROR response has been received from the destination node).

84.6.2 Inbound doorbell messages

The following timing diagram illustrates how inbound doorbell messages are made available to the external interface (if this feature is enabled):

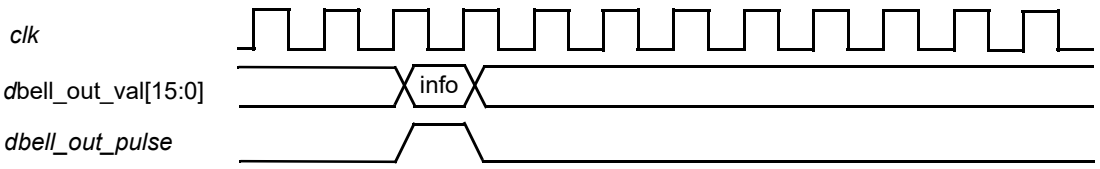


Figure 231. Skew and jitter timing waveforms

The doorbell info value is available at DBELL_OUT_VAL for one clock cycle when DBELL_OUT_-PULSE is asserted to 1. DBELL_OUT_PULSE can be used as a write enable signal to a register or memory that stores the doorbell info value for further processing.

84.7 Implementation

84.7.1 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

84.8 Configuration options

Table 1705 shows the configuration options of the core (VHDL generics).

*Table 1705.*Configuration options

Generic	Function	Allowed values	Default
g_tech	Selects technology for RAM blocks.	0 - NTECH	DEFMEMTECH
g_use_async_rst	Use asynchronous instead of synchronous resets.	FALSE, TRUE	FALSE
g_data_width	See section 84.4 for details.	32 - 128	32
g_max_burst_length	See section 84.4 for details.	2 - 256	256
g_burst_chop_mask	See section 84.4 for details.	8 - 4096 (AXI) 8 - 1024 (AHB)	4096 (AXI) 1024 (AHB)
g_be_rd_pipe	See section 84.4 for details.	0-1	1
g_unalign_load_opt	See section 84.4 for details.	0-1	0
g_bm_info_print[1]	When it is set to a value apart from 0, prints out bus master configuration information during simulation.	-	1
g_axi_bm_id_width[1]	Determines the ID width for read address/data, write address, and write response channels.	-	5
g_axi_bm_addr_width[1]	Determines the address width for both front-end and back-end of the bus master.	32 - 64	32
g_no_dbell_rx	Number of doorbell reception queues.	1 - 32	4
g_no_dbell_tx[4]	Number of doorbell transmission queues[4]	1 - 32	4
g_no_msg_rx	Number of data message reception queues	1 - 32	4
g_no_hyb_tx[3]	Number of hybrid queues[3]	1-32	4
g_no_io_tx[4]	Number of IO memory access transmission queues[4]	1-32	4
g_no_msg_tx[4]	Number of data message transmission queues[4]	1 - 32	4
g_init_nodes	Number of data nodes initially available in SRIO end point.	0 - 255	72
g_init_crq	Number of CRQ entries initially available in SRIO end point.	0 - 255	64
g_wm_p0	Watermark for priority 0.	0 - 255	72

*Table 1705.*Configuration options

Generic	Function	Allowed values	Default
g_wm_p1	Watermark for priority 1.	0 - 255	63
g_wm_p2	Watermark for priority 2.	0 - 255	54
g_wm_p3	Watermark for priority 3.	0 - 255	45
g_wm_p4	Watermark for priority 4.	0 - 255	36
g_wm_p5	Watermark for priority 5.	0 - 255	27
g_wm_p6	Watermark for priority 6.	0 - 255	18
g_wm_p7	Watermark for priority 7.	0 - 255	9
g_8_in_win	Choose between 4 (false) and 8 inbound windows (true)	FALSE,TRUE	FALSE
g_mecs	Enable support for enhanced MECS (true)	FALSE,TRUE	FALSE
g_hyb_que	If set to “true” each TX queues can handle every kind of operations.	FALSE,TRUE	FALSE

[1] Only available in grsrio_gen2axi.vhd
[2] Only available in grsrio_gen2ahb.vhd
[3] Effective only when g_hyb_que is set to “true”
[4] Effective only when g_hyb_que is set to “false”

The core is configurate as 32 or 64-bit address according to the value of CFG_AW in grsrio_conf.vhd.

84.9 Signal Descriptions

Table 1706 shows the interface signals of the core (VHDL ports).

*Table 1706.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	AHB reset	Low
CLK	N/A	Input	AHB clock	-
SRIO_RSTN	N/A	Input	SRIO physical layer reset	Low
SRIO_CLK	N/A	Input	SRIO physical layer clock	-
APBO_PRDATA[31:0][2]	N/A	Output	APB slave: Read Data Bus	-
APBO_PIRQ[2]	N/A	Output	APB slave: Interrupt Output	High
APBI_PENABLE[2]	N/A	Input	APB slave: Strobe	High
APBI_PWRITE[2]	N/A	Input	APB slave: Transfer Direction	High
APBI_PADDR[31:0][2]	N/A	Input	APB slave: Address Bus	-
APBI_PWDATA[31:0][2]	N/A	Input	APB slave: Write Data Bus	-
APBI_PSEL[2]	N/A	Input	APB slave: Select	High
AHBMO_HBUSREQ[2]	N/A	Output	AHB master: Bus Request	High
AHBMO_HLOCK[2]	N/A	Output	AHB master: Locked Transfers	High
AHBMO_HTRANS[1:0][2]	N/A	Output	AHB master: Transfer Type	-
AHBMO_HADDR[31:0][2]	N/A	Output	AHB master: Address Bus	-
AHBMO_HWRITE[2]	N/A	Output	AHB master: Transfer Direction	High
AHBMO_HSIZE[2:0][2]	N/A	Output	AHB master: Transfer Size	-
AHBMO_HBURST[2:0][2]	N/A	Output	AHB master: Burst Type	-
AHBMO_HPROT[3:0][2]	N/A	Output	AHB master: Protection Control	-

Table 1706. Signal descriptions

Signal name	Field	Type	Function	Active
AHBMO_HWDATA [g_ahb_width-1:0][2]	N/A	Output	AHB master: Write Data Bus	-
AHBMI_HGRANT[2]	N/A	Input	AHB master: Bus Grant	High
AHBMI_HREADY[2]	N/A	Input	AHB master: Transfer Done	High
AHBMI_HRESP[1:0][2]	N/A	Input	AHB master: Transfer Response	-
AHBMI_HRDATA [g_ahb_width-1:0][2]	N/A	Input	AHB master: Read Data Bus	-
AXI_AW_ID [axi_bm_id_width-1:0][1]	N/A	Output	AXI4 master write address channel: Write Address ID	-
AXI_AW_ADDR [axi_bm_addr_width-1:0][1]	N/A	Output	AXI4 master write address channel: Write Address.	-
AXI_AW_LEN [7:0][1]	N/A	Output	AXI4 Master write address channel: Burst Length	-
AXI_AW_SIZE [2:0][1]	N/A	Output	AXI4 master write address channel: Burst Size	-
AXI_AW_BURST [1:0][1]	N/A	Output	AXI4 master write address channel: Burst Type	-
AXI_AW_LOCK [1:0][1]	N/A	Output	AXI4 master write address channel: Lock Type	-
AXI_AW_CACHE [3:0][1]	N/A	Output	AXI4 master write address channel: Memory type	-
AXI_AW_PROT [2:0][1]	N/A	Output	AXI4 master write address channel: Protection type	-
AXI_AW_VALID[1]	N/A	Output	AXI4 master write address channel: Write address valid	High
AXI_AW_QOS [3:0][1]	N/A	Output	AXI4 master write address channel: Quality of Service	-
AXI_AW_READY[1]	N/A	Input	AXI4 master write address channel: Write address ready.	High
AXI_W_DATA [be_dw-1:0][1]	N/A	Output	AXI4 write data channel signals: Write data.	-
AXI_W_STRB [log2(be_dw)-1:0][1]	N/A	Output	AXI4 write data channel signals: Write strobes	High
AXI_W_LAST[1]	N/A	Output	AXI4 write data channel signalsWrite last.	High
AXI_W_VALID[1]	N/A	Output	AXI4 write data channel signalsWrite valid.	High
AXI_B_READY[1]	N/A	Output	AXI4 write response channel signals: Response ready	High
AXI_B_ID [axi_bm_id_width-1 : 0][1]	N/A	Input	AXI4 write response channel signals: Response ID tag	-
AXI_B_RESP [1 downto 0][1]	N/A	Input	AXI4 write response channel signals: Write response	-
AXI_B_VALID[1]	N/A	Input	AXI4 write response channel signals: Write response valid	High
AXI_AR_ID [axi_bm_id_width-1:0][1]	N/A	Output	AXI4 read address channel signals: Read address ID	-
axi_ar_addr [axi_bm_addr_width-1:0][1]	N/A	Output	AXI4 read address channel signals: Read address	-

Table 1706. Signal descriptions

Signal name	Field	Type	Function	Active
AXI_AR_LEN [7:0][1]	N/A	Output	AXI4 read address channel signals: Burst length.	-
AXI_AR_SIZE [2:0][1]	N/A	Output	AXI4 read address channel signals: Burst size	-
AXI_AR_BURST [1:0][1]	N/A	Output	AXI4 read address channel signals: Burst type.	-
AXI_AR_LOCK [1:0][1]	N/A	Output	AXI4 read address channel signals: Lock type.	-
AXI_AR_CACHE [3:0][1]	N/A	Output	AXI4 read address channel signals: Memory type	-
AXI_AR_PROT [2:0][1]	N/A	Output	AXI4 read address channel signals: Protection type.	-
AXI_AR_VALID[1]	N/A	Output	AXI4 read address channel signals: Read address valid	High
AXI_AR_QOS[3:0][1]	N/A	Output	AXI4 read address channel signals: Quality of Service	-
AXI_AR_READY[1]	N/A	Input	AXI4 read address channel signals: Read address ready.	High
AXI_R_READY[1]	N/A	Output	AXI4 read address channel signals: Read ready.	High
AXI_R_ID [axi_bm_id_width-1:0][1]	N/A	Input	AXI4 read address channel signals: Read ID tag.	-
AXI_R_DATA [be_dw-1:0][1]	N/A	Input	AXI4 read address channel signals	-
AXI_R_RESP [1:0][1]	N/A	Input	AXI4 read address channel signals	-
AXI_R_LAST[1]	N/A	Input	AXI4 read address channel signals	High
AXI_R_VALID[1]	N/A	Input	AXI4 read address channel signals	High
TIMESTAMP[63:0]	N/A	Input	External time-stamp value, used to timestamp descriptors if g_mecs is set to false or MECS are not enabled.	-
DBELL_OUT_VAL[15:0]	N/A	Output	Doorbell info value of inbound message, valid when DBELL_OUT_PULSE is pulsed.	-
DBELL_OUT_PULSE	N/A	Output	Pulse signaling that outputted doorbell info value is valid.	High
DBELL_IN_VAL[15:0]	N/A	Input	Doorbell info value of outbound message. Must be kept stable until DBELL_IN_ACK goes high.	-
DBELL_IN_WEN	N/A	Input	Doorbell info write enable signal. Must be asserted together with DBELL_IN_VAL and kept asserted until DBELL_IN_ACK goes high.	High
DBELL_IN_BUSY	N/A	Output	Asserted while external outbound doorbell message is processed.	High
DBELL_IN_ACK	N/A	Output	Pulse acknowledging transmission of external outbound doorbell message.	High
DBELL_IN_SRIO_ERR	N/A	Output	Indicates a SRIO error during doorbell transmission. Valid together with DBELL_IN_ACK.	High
DBELL_IN_RETRY_ERR	N/A	Output	Indicates a retry error during doorbell transmission. Valid together with DBELL_IN_ACK.	High
DBELL_IN_TIMEOUT_ERR	N/A	Output	Indicates a timeout error during doorbell transmission. Valid together with DBELL_IN_ACK.	High
MECS_CAPTURED_IN[7:0]	N/A	Input	The SRIO end point signals to this IP core the reception of an extended MECS with cmd = i when the i-th signal changes	Edge
MECS_TRIGGER_OUT[7:0]	N/A	Output	Extended MECS with cmd = i are triggered in the endpoint when the ith signal changes	Edge

Table 1706. Signal descriptions

Signal name	Field	Type	Function	Active
SRIO_TX_DATA[127:0]	N/A	Output	Outbound UDI packet data bus	-
SRIO_TX_VALID	N/A	Output	When asserted, SRIO_TX_DATA is valid.	High
SRIO_TX_EOP	N/A	Output	When asserted, data SRIO_TX_DATA is the end of the packet.	High
SRIO_TX_HWORD[2:0]	N/A	Output	Indicates which half-words of SRIO_TX_DATA are valid.	-
SRIO_TX_HALT_ACK	N/A	Output	Two-way handshake to SRIO_TX_HALT.	High
SRIO_TX_CCOUNT[4:0]	N/A	Input	The value signals the number of data nodes that have been freed by the physical layer.	-
SRIO_TX_HALT	N/A	Input	When asserted, the GRSRIO IP core stops Egress traffic after finishing the current in-flight packet.	High
SRIO_RX_FULL	N/A	Output	Used by the GRSRIO IP core to pause the transfer of packets.	High
SRIO_RX_RELEASE[5:0]	N/A	Output	Indicates the number of ingress FIFO entries that have been freed by the GRSRIO IP core.	-
SRIO_RX_DATA[127:0]	N/A	Input	Inbound UDI packet data bus.	-
SRIO_RX_VALID	N/A	Input	When asserted, SRIO_RX_DATA is valid.	High
SRIO_RX_EOP	N/A	Input	When asserted, SRIO_RX_DATA is the end of the packet	High
SRIO_RX_HWORD[2:0]	N/A	Input	Indicates which half-words of SRIO_RX_DATA are valid.	-
SRIO_RX_STOMP	N/A	Input	Asserted together with SRIO_RX_EOP to flag a packet with CRC error (not supported by GRSRIO IP core)	High

[1] Only available in grsrio_gen2axi.vhd

[2] Only available in grsrio_gen2ahb.vhd

84.10 Clocks

The GRSRIO core operates in the CLK clock domain. Clock domain crossing between the CLK clock domain and the SRIO_CLK clock domain is accomplished by various CDC techniques.

The SRIO end point provides the signal SRIO_TX_CCOUNT[4:0] to the GRSRIO core, which is synchronous to and valid on every clock cycle of SRIO_CLK. Lost samples corrupt the flow control and must therefore be avoided. As a consequence, the AHB clock must be greater than or equal the SRIO clock. Lost samples are reported in CORECONF1.FE.

84.11 Resets

To ensure proper initialisation of synchronous logic related to the clock domain crossing between the SRIO_CLK and CLK clock domain, both the SRIO_RSTN and RSTN reset must be asserted for at least three clock cycles in the slower clock domain.

84.12 Library dependencies

Table 1707 shows libraries used when instantiating the core (VHDL libraries).

*Table 1707.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions

84.13 Component declaration

For the AXI4 wrapper:

```
component grsrio_gen2axi
  generic (
    g_tech              : integer;
    g_use_async_rst     : boolean;
    g_data_width        : integer range 32 to 128;
    g_max_burst_length  : integer range 2 to 64;
    g_burst_chop_mask   : integer range 8 to 4096;
    g_be_rd_pipe        : integer range 0 to 1;
    g_unalign_load_opt  : integer range 0 to 1;
    axi_bm_id_width     : integer;
    axi_bm_addr_width   : integer;
    g_bm_info_print     : integer;
    g_no_dbell_tx       : integer range 1 to 32;
    g_no_dbell_rx       : integer range 1 to 32;
    g_no_msg_tx         : integer range 1 to 32;
    g_no_msg_rx         : integer range 1 to 32;
    g_no_io_tx          : integer range 1 to 32;
    g_no_hyb_tx         : integer range 1 to 32;
    g_init_nodes        : integer range 0 to 255;
    g_init_crq          : integer range 0 to 255;
    g_wm_p0             : integer range 0 to 255;
    g_wm_p1             : integer range 0 to 255;
    g_wm_p2             : integer range 0 to 255;
    g_wm_p3             : integer range 0 to 255;
    g_wm_p4             : integer range 0 to 255;
    g_wm_p5             : integer range 0 to 255;
    g_wm_p6             : integer range 0 to 255;
    g_wm_p7             : integer range 0 to 255;
    g_8_in_win          : boolean;
    g_mecs              : boolean;
    g_hyb_que           : boolean;
    pindex              : integer range 0 to NAPBSLV-1;
    pirq                : integer range 0 to NAHBIRQ-1;
    paddr               : integer range 0 to 16#FFF#;
    pmask               : integer range 0 to 16#FFF#);
  port (
    clk                   : in  std_logic;
    clk_lock              : in  std_logic;
    rstn                  : in  std_logic;
    srio_clk              : in  std_logic;
    srio_rstn             : in  std_logic;
    soft_reset            : out std_logic;
    timestamp             : in  std_logic_vector(63 downto 0);
    dbell_out_val         : out std_logic_vector(15 downto 0);
    dbell_out_pulse       : out std_logic;
    dbell_in_val          : in  std_logic_vector(15 downto 0);
    dbell_in_wen          : in  std_logic;
    dbell_in_busy         : out std_logic;
    dbell_in_ack          : out std_logic;
    dbell_in_srio_err     : out std_logic;
```

```
dbell_in_retry_err   : out std_logic;
dbell_in_timeout_err : out std_logic;
apbo_prdata          : out std_logic_vector(31 downto 0);
apbo_pirq            : out std_logic;
apbo_pindex          : out integer range 0 to NAPBSLV -1;
apbo_pconfig         : out apb_config_type;
apbi_penable         : in  std_logic;
apbi_pwrite          : in  std_logic;
apbi_paddr           : in  std_logic_vector(31 downto 0);
apbi_pwdata          : in  std_logic_vector(31 downto 0);
apbi_psel            : in  std_logic;
axi_aw_id            : out std_logic_vector(axi_bm_id_width-1 downto 0);
axi_aw_addr          : out std_logic_vector(axi_bm_addr_width-1 downto 0);
axi_aw_len           : out std_logic_vector(7 downto 0);
axi_aw_size          : out std_logic_vector(2 downto 0);
axi_aw_burst         : out std_logic_vector(1 downto 0);
axi_aw_lock          : out std_logic;
axi_aw_cache         : out std_logic_vector(3 downto 0);
axi_aw_prot          : out std_logic_vector(2 downto 0);
axi_aw_valid         : out std_logic;
axi_aw_qos           : out std_logic_vector(3 downto 0);
axi_aw_ready         : in  std_logic;
axi_w_data           : out std_logic_vector(g_data_width-1 downto 0);
axi_w_strb           : out std_logic_vector((g_data_width/8)-1 downto 0);
axi_w_last           : out std_logic;
axi_w_valid          : out std_logic;
axi_w_ready          : in  std_logic;
axi_b_ready          : out std_logic;
axi_b_id             : in  std_logic_vector(axi_bm_id_width-1 downto 0);
axi_b_resp           : in  std_logic_vector(1 downto 0);
axi_b_valid          : in  std_logic;
axi_ar_id            : out std_logic_vector(axi_bm_id_width-1 downto 0);
axi_ar_addr          : out std_logic_vector(axi_bm_addr_width-1 downto 0);
axi_ar_len           : out std_logic_vector(7 downto 0);
axi_ar_size          : out std_logic_vector(2 downto 0);
axi_ar_burst         : out std_logic_vector(1 downto 0);
axi_ar_lock          : out std_logic;
axi_ar_cache         : out std_logic_vector(3 downto 0);
axi_ar_prot          : out std_logic_vector(2 downto 0);
axi_ar_valid         : out std_logic;
axi_ar_qos           : out std_logic_vector(3 downto 0);
axi_ar_ready         : in  std_logic;
axi_r_ready          : out std_logic;
axi_r_id             : in  std_logic_vector(axi_bm_id_width-1 downto 0);
axi_r_data           : in  std_logic_vector(g_data_width-1 downto 0);
axi_r_resp           : in  std_logic_vector(1 downto 0);
axi_r_last           : in  std_logic;
axi_r_valid          : in  std_logic;
srio_tx_data         : out std_logic_vector(127 downto 0);
srio_tx_eop          : out std_logic;
srio_tx_hword        : out std_logic_vector(2 downto 0);
srio_tx_valid        : out std_logic;
srio_tx_halt_ack     : out std_logic;
srio_tx_halt         : in  std_logic;
srio_tx_ccount       : in  std_logic_vector(4 downto 0);
srio_rx_full         : out std_logic;
srio_rx_release      : out std_logic_vector(5 downto 0);
srio_rx_data         : in  std_logic_vector(127 downto 0);
srio_rx_valid        : in  std_logic;
srio_rx_eop          : in  std_logic;
srio_rx_hword        : in  std_logic_vector(2 downto 0);
srio_rx_stomp        : in  std_logic;
send_mecs            : out std_logic_vector(7 downto 0);
received_mecs        : in  std_logic_vector(7 downto 0));
```

```
end component;
```

For the AHB wrapper:

```
component grsrio_gen2ahb
  generic (
    g_tech              : integer;
    g_use_async_rst     : boolean;
    g_data_width        : integer range 32 to 128;
    g_addr_width        : integer range 32 to 64;
    g_max_burst_length  : integer range 2 to 64;
    g_burst_chop_mask   : integer range 8 to 1024;
    g_be_rd_pipe        : integer range 0 to 1;
    g_unalign_load_opt  : integer range 0 to 1;
    g_no_dbell_tx       : integer range 1 to 32;
    g_no_dbell_rx       : integer range 1 to 32;
    g_no_msg_tx         : integer range 1 to 32;
    g_no_msg_rx         : integer range 1 to 32;
    g_no_io_tx          : integer range 1 to 32;
    g_no_hyb_tx         : integer range 1 to 32;
    g_init_nodes        : integer range 0 to 255;
    g_init_crq          : integer range 0 to 255;
    g_wm_p0             : integer range 0 to 255;
    g_wm_p1             : integer range 0 to 255;
    g_wm_p2             : integer range 0 to 255;
    g_wm_p3             : integer range 0 to 255;
    g_wm_p4             : integer range 0 to 255;
    g_wm_p5             : integer range 0 to 255;
    g_wm_p6             : integer range 0 to 255;
    g_wm_p7             : integer range 0 to 255;
    g_8_in_win          : boolean;
    g_mecs              : boolean;
    g_hyb_que           : boolean;
    pindex              : integer range 0 to NAPBSLV-1;
    pirq                : integer range 0 to NAHBIRQ-1;
    paddr               : integer range 0 to 16#FFF#;
    pmask               : integer range 0 to 16#FFF#);
  port (
    clk                   : in  std_logic;
    clk_lock              : in  std_logic;
    rstn                  : in  std_logic;
    srio_clk              : in  std_logic;
    srio_rstn             : in  std_logic;
    soft_reset            : out std_logic;
    timestamp             : in  std_logic_vector(63 downto 0);
    dbell_out_val         : out std_logic_vector(15 downto 0);
    dbell_out_pulse       : out std_logic;
    dbell_in_val          : in  std_logic_vector(15 downto 0);
    dbell_in_wen          : in  std_logic;
    dbell_in_busy         : out std_logic;
    dbell_in_ack          : out std_logic;
    dbell_in_srio_err     : out std_logic;
    dbell_in_retry_err    : out std_logic;
    dbell_in_timeout_err  : out std_logic;
    apbo_prdata           : out std_logic_vector(31 downto 0);
    apbo_pirq             : out std_logic;
    apbo_pindex           : out integer range 0 to NAPBSLV -1;
    apbo_pconfig          : out apb_config_type;
    apbi_penable          : in  std_logic;
    apbi_pwrite           : in  std_logic;
    apbi_paddr            : in  std_logic_vector(31 downto 0);
    apbi_pwdata           : in  std_logic_vector(31 downto 0);
    apbi_psel             : in  std_logic;
    ahbmo_hbusreq         : out std_logic;
```

```
    ahbmo_hlock          : out std_logic;
    ahbmo_htrans         : out std_logic_vector(1 downto 0);
    ahbmo_haddr          : out std_logic_vector(g_addr_width-1 downto 0);
    ahbmo_hwrite         : out std_logic;
    ahbmo_hsize          : out std_logic_vector(2 downto 0);
    ahbmo_hburst         : out std_logic_vector(2 downto 0);
    ahbmo_hprot          : out std_logic_vector(3 downto 0);
    ahbmo_hwdata         : out std_logic_vector(g_data_width-1 downto 0);
    ahbmi_hgrant         : in  std_logic;
    ahbmi_hready         : in  std_logic;
    ahbmi_hresp          : in  std_logic_vector(1 downto 0);
    ahbmi_hrdata         : in  std_logic_vector(g_data_width-1 downto 0);
    srio_tx_data         : out std_logic_vector(127 downto 0);
    srio_tx_eop          : out std_logic;
    srio_tx_hword        : out std_logic_vector(2 downto 0);
    srio_tx_valid        : out std_logic;
    srio_tx_halt_ack     : out std_logic;
    srio_tx_halt         : in  std_logic;
    srio_tx_ccount       : in  std_logic_vector(4 downto 0);
    srio_rx_full         : out std_logic;
    srio_rx_release      : out std_logic_vector(5 downto 0);
    srio_rx_data         : in  std_logic_vector(127 downto 0);
    srio_rx_valid        : in  std_logic;
    srio_rx_eop          : in  std_logic;
    srio_rx_hword        : in  std_logic_vector(2 downto 0);
    srio_rx_stomp        : in  std_logic;
    send_mecs            : out std_logic_vector(7 downto 0);
    received_mecs        : in  std_logic_vector(7 downto 0));
end component;
```

84.14 Instantiation

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;

..

grsrio_gen2axi_1 : grsrio_gen2axi
    generic map (
      g_tech => CFG_MEMTECH,
      g_use_async_rst => false,
      g_data_width => 128,
      g_max_burst_length => 16,
      g_be_rd_pipe       => 0,
      g_unalign_load_opt => 1,
      axi_bm_id_width => 4,
      axi_bm_addr_width => 32,
      g_bm_info_print => 0,
      g_no_dbell_tx => 6,
      g_no_dbell_rx => 6,
      g_no_msg_tx => 6,
      g_no_msg_rx => 6,
      g_no_io_tx => 6,
      g_no_hyb_tx => 6,
      g_init_nodes => 72,
      g_init_crq => 64,
      g_wm_p0 => 72,
      g_wm_p1 => 63,
      g_wm_p2 => 54,
      g_wm_p3 => 45,
      g_wm_p4 => 36,
      g_wm_p5 => 27,
```

```
      g_wm_p6 => 18,
      g_wm_p7 => 9,
      g_8_in_win => true,
      g_mecs => true,
      g_hyb_que => true,
      pindex => 6,
      pirq => 6,
      paddr => 16#000#,
      pmask => 16#000#)
  port map (
    clk => clkm,
    clk_lock => lock,
    rstn => grsrio_rst,
    srio_clk => udi_clk,
    srio_rstn => udi_rst,
    soft_reset => soft_reset,
    timestamp => timestamp,
    dbell_out_val => dbell_out_val,
    dbell_out_pulse => dbell_out_pulse,
    dbell_in_val => dbell_in_val,
    dbell_in_wen => dbell_in_wen,
    dbell_in_busy => dbell_in_busy,
    dbell_in_ack => dbell_in_ack,
    dbell_in_srio_err => dbell_in_srio_err,
    dbell_in_retry_err => dbell_in_retry_err,
    dbell_in_timeout_err => dbell_in_timeout_err,
    apbo_prdata => apbo(6).prdata,
    apbo_pirq => apbo(6).pirq(6),
    apbo_pindex => apbo(6).pindex,
    apbo_pconfig => apbo(6).pconfig,
    apbi_penable => apbi.penable,
    apbi_pwrite => apbi.pwrite,
    apbi_paddr => apbi.paddr,
    apbi_pwdata => apbi.pwdata,
    apbi_psel => apbi.psel(6),
    axi_aw_id => grsrio_axi_awid,
    axi_aw_addr => grsrio_axi_awaddr,
    axi_aw_len => grsrio_axi_awlen,
    axi_aw_size => grsrio_axi_awsize,
    axi_aw_burst => grsrio_axi_awburst,
    axi_aw_lock => grsrio_axi_awlock,
    axi_aw_cache => grsrio_axi_awcache,
    axi_aw_prot => grsrio_axi_awprot,
    axi_aw_valid => grsrio_axi_awvalid,
    axi_aw_qos => grsrio_axi_awqos,
    axi_aw_ready => grsrio_axi_awready,
    axi_w_data => grsrio_axi_wdata,
    axi_w_strb => grsrio_axi_wstrb,
    axi_w_last => grsrio_axi_wlast,
    axi_w_valid => grsrio_axi_wvalid,
    axi_w_ready => grsrio_axi_wready,
    axi_b_ready => grsrio_axi_bready,
    axi_b_id => grsrio_axi_bid,
    axi_b_resp => grsrio_axi_bresp,
    axi_b_valid => grsrio_axi_bvalid,
    axi_ar_id => grsrio_axi_arid,
    axi_ar_addr => grsrio_axi_araddr,
    axi_ar_len => grsrio_axi_arlen,
    axi_ar_size => grsrio_axi_arsize,
    axi_ar_burst => grsrio_axi_arburst,
    axi_ar_lock => grsrio_axi_arlock,
    axi_ar_cache => grsrio_axi_arcache,
    axi_ar_prot => grsrio_axi_arprot,
    axi_ar_valid => grsrio_axi_arvalid,
```

```
axi_ar_qos => grsrio_axi_arqos,
axi_ar_ready => grsrio_axi_arready,
axi_r_ready => grsrio_axi_rready,
axi_r_id => grsrio_axi_rid,
axi_r_data => grsrio_axi_rdata,
axi_r_resp => grsrio_axi_rresp,
axi_r_last => grsrio_axi_rlast,
axi_r_valid => grsrio_axi_rvalid,
srio_tx_data => srio_tx_data,
srio_tx_eop => srio_tx_eop,
srio_tx_hword => srio_tx_hword,
srio_tx_valid => srio_tx_valid,
srio_tx_halt_ack => srio_tx_halt_ack,
srio_tx_halt => srio_tx_halt,
srio_tx_ccount => srio_tx_ccount,
srio_rx_full => srio_rx_full,
srio_rx_release => srio_rx_release,
srio_rx_data => srio_rx_data,
srio_rx_valid => srio_rx_valid,
srio_rx_eop => srio_rx_eop,
srio_rx_hword => srio_rx_hword,
srio_rx_stomp => srio_rx_stomp,
send_mecs => send_mecs,
received_mecs => received_mecs);
```

85 GRSYSMON - AMBA Wrapper for Xilinx System Monitor

85.1 Overview

The core provides an AMBA AHB interface to the Xilinx System Monitor present in Virtex-5 FPGAs. All Xilinx System Monitor registers are mapped into AMBA address space. The core also includes functionality for generating interrupts triggered by System Monitor outputs, and allows triggering of conversion start via a separate register interface.

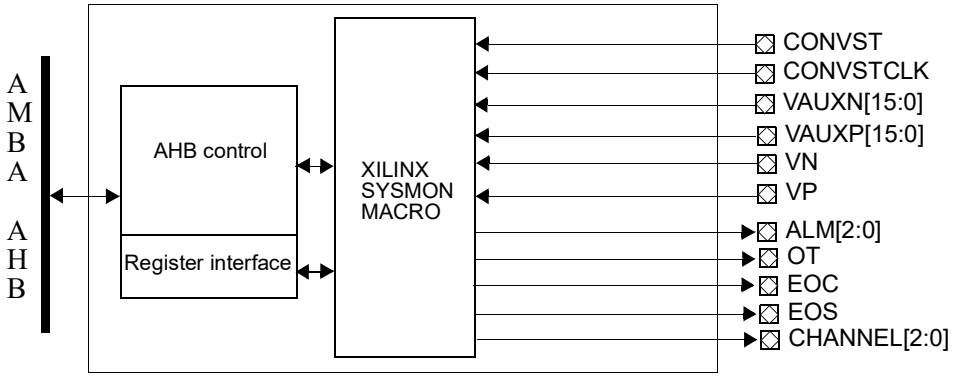


Figure 232. Block diagram

85.2 Operation

85.2.1 Operational model

The core has two I/O areas that can be accessed via the AMBA bus; the core configuration area and the System Monitor register area.

85.2.2 Configuration area

The configuration area, accessed via AHB I/O bank 0, contains two registers that provide status information and allow the user to generate interrupts from the Xilinx System Monitor's outputs. Write accesses to the configuration area have no AHB wait state and read accesses have one wait state. To ensure correct operation, only word (32-bit) sized accesses should be made to the configuration area.

85.2.3 System Monitor register area

The System Monitor register area is located in AHB I/O bank 1 and provides a direct-mapping to the System Monitor's Dynamic Reconfiguration Port. The System Monitor's first register is located at address offset 0x00000000 in this area.

Since the System Monitor documentation defines its addresses using half-word addressing, and AMBA uses byte-addressing, the addresses in the System Monitor documentation should be multiplied to get the correct offset in AMBA memory space. If the Configuration register bit WAL is '0' the address in System Monitor documentation should be multiplied by two to get the address mapped by the AMBA wrapper. A System Monitor register with address n is at AMBA offset 2*n. If the Configuration register bit WAL is '1', all registers start at a word boundary and the address in the System Monitor documentation should be multiplied by four to get the address mapped in AMBA address space. In this case, a System Monitor register with address n is at AMBA offset 4*n.

The wrapper always makes a single register access as the result of an access to the System Monitor register area. The size of the AMBA access is not checked and to ensure correct operation the mapped area should only be accessed using half-word (16-bit) accesses.

If the core has been implemented with AMBA split support, it will issue a SPLIT response to all accesses made to the mapped System Monitor registers. If the core is implemented without AMBA SPLIT support, wait states will be inserted until the System Monitor signals completion of a register access.

For a description of the System Monitor's capabilities and configuration, please refer to the Xilinx Virtex-5 FPGA System Monitor User Guide.

85.3 Registers

The core is programmed through registers mapped into AHB address space. Only 32-bit single-accesses to the registers are supported.

*Table 1708.*GRSYSMON registers

AHB address offset	Register
0x00	Configuration register
0x04	Status register

85.3.1 Configuration Register

Table 1709. 0x00 - CONF - Configuration register

31	31 13	12	11 9	8 7	6	5	4	3	2	1	0
WAL	RESERVED	OT_IEN	ALM_IEN	RESERVED	CON-VST	EOS_ IEN	EOC_ IEN	BUSY_ IEN	JB_IEN	JL_IEN	JM_IEN
*	0	0	0	0	0	0	0	0	0	0	0
r	r	rw	rw	r	rw*	rw	rw	rw	rw	rw	rw

31 Word aligned registers (WAL) - If this bit is set to '1' each System Monitor memory mapped register start at a word boundary.

30 :13 RESERVED

12 Over temperature Interrupt Enable (OT_IEN) - If this bit is set to '1' the core will generate an interrupt when the corresponding bit in the Status register is set to '1'. This bit is automatically cleared after the interrupt has been generated.

11:9 Alarm Interrupt Enable (ALM_IEN) - If a bit in this field is set to '1' the core will generate an interrupt when the corresponding bit in the Status register is set to '1'. This bit is automatically cleared after the interrupt has been generated.

8:7 RESERVED

6 Conversion Start (CONVST) - If the core has been configured, at implementation, to use the an internal source for the Xilinx System Monitor CONVST signal, this bit can be written to '1' to generate a pulse on the System Monitor's CONVST input. This bit is automatically cleared after one clock cycle.

5 End of Sequence Interrupt Enable (EOS_IEN) - If this bit is set to '1' the core will generate an interrupt when the corresponding bit in the Status register is set to '1'. This bit is automatically cleared after the interrupt has been generated.

4 End of Conversion Interrupt Enable (EOC_IEN) - If this bit is set to '1' the core will generate an interrupt when the corresponding bit in the Status register is set to '1'. This bit is automatically cleared after the interrupt has been generated.

3 Busy Interrupt Enable (BUSY_IEN) - If this bit is set to '1' the core will generate an interrupt when the corresponding bit in the Status register is set to '1'. This bit is automatically cleared after the interrupt has been generated.

2 JTAG Busy Interrupt Enable (JB_IEN) - If this bit is set to '1' the core will generate an interrupt when the corresponding bit in the Status register is set to '1'. This bit is automatically cleared after the interrupt has been generated.

1 JTAG Locked Interrupt Enable (JL_IEN) - If this bit is set to '1' the core will generate an interrupt when the corresponding bit in the Status register is set to '1'. This bit is automatically cleared after the interrupt has been generated.

0 JTAG Modified Interrupt Enable (JM_IEN) - .If this bit is set to '1' the core will generate an interrupt when the corresponding bit in the Status register is set to '1'. This bit is automatically cleared after the interrupt has been generated.

Reset value: 0x00000000

85.3.2 Status Register

Table 1710. 0x04 - STAT - Status register

31	30 13	12	11 9	8 6	5	4	3	2	1	0
WAL	RESERVED	OT	ALM	RESERVED	EOS	EOC	BUSY	JB	JL	JM
0	0	-	-	0	-	-	-	-	-	-
r	r	r	r	r	r	r	r	r	r	r

31 Word aligned registers (WAL) - If this bit is set to '1' each System Monitor memory mapped register start at a word boundary.

30 :13 RESERVED

12 Over Temperature (OT) - Connected to the System Monitor's Temperature Alarm output.

11:9 Alarm (ALM) - Connected to the System Monitor's alarm outputs.

8:6 RESERVED

Table 1710. 0x04 - STAT - Status register

5	End of Sequence (EOS) - Connected to the System Monitor's End of Sequence output.
4	End of Conversion (EOC) - Connected to the System Monitors End of Conversion output.
3	Busy (BUSY) - Connected to the System Monitor's Busy output.
2	JTAG Busy (JB) - Connected to the System Monitor's JTAG Busy output.
1	JTAG Locked (JL) - Connected to the System Monitor's JTAG Locked output.
0	JTAG Modified (JM) - Connected to the System Monitor's JTAG Modified output.

Reset value: See Xilinx System Monitor documentation

85.4 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x066. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

85.5 Implementation

85.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

85.5.2 Technology mapping

The core instantiates a SYSMON primitive.

85.5.3 RAM usage

The core does not use any RAM components.

85.5.4 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core provides registers mapped on AHB that can be accessed in both big-endian and little-endian systems.

85.6 Configuration options

Table 1711 shows the configuration options of the core (VHDL generics).

Table 1711. Configuration options

Generic name	Function	Allowed range	Default
tech	Target technology	0 - NTECH	0
hindex	AHB slave index	0 - (NAHBSLV-1)	0
hirq	Interrupt line	0 - (NAHBIRQ-1)	0
caddr	ADDR field of the AHB BAR0 defining configuration register address space.	0 - 16#FFF#	16#000#
cmask	MASK field of the AHB BAR0 defining configuration register address space.	0 - 16#FFF#	16#FFF#
saddr	ADDR field of the AHB BAR1 defining System Monitor register address space.	0 - 16#FFF#	16#001#
smask	MASK field of the AHB BAR1 defining System Monitor register space.	0 - 16#FFF#	16#FFF#

Table 1711. Configuration options

Generic name	Function	Allowed range	Default
split	If this generic is set to 1 the core will issue AMBA SPLIT responses when it is busy performing an operation with the System Monitor. Otherwise the core will insert wait states until the operation completes.	0 - 1	0
extconvst	Connect CONVST input to System Monitor. If this generic is set to '0' the System Monitor's CONVST is controlled via the configuration register, otherwise the System Monitor CONVST input is taken from the core input signal.	0 - 1	0
wrdalign	Word align System Monitor registers. If this generic is set to 1 all System Monitor registers will begin on a word boundary. The first register will be mapped at offset 0x00, the second at 0x04. To translate a register access specified in the Xilinx System Monitor register documentation the register address should be multiplied by four to get the correct offset in AMBA address space. If this generic is set to 0, the register address should be multiplied by two to get the offset in AMBA address space.	0 - 1	0
INIT_40	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_41	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_42	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	16#0800#
INIT_43	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_44	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_45	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_46	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_47	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_48	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_49	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_4A	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_4B	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_4C	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_4D	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_4E	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_4F	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_50	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_51	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_52	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_53	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_54	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_55	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_56	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
INIT_57	Xilinx System Monitor register initialization value.	0 - 16#FFFF#	0
SIM_MONITOR_FILE	Simulation analog entry file. See Xilinx System Monitor documentation for a description of use and format.	-	"sysmon.txt"

85.7 Signal descriptions

Table 1712 shows the interface signals of the core (VHDL ports).

*Table 1712.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
SYSMONI	CONVST	Input	Convert start input, connected to Xilinx System Monitor if the *extconvst* VHDL generic is set to '1'.	High
	CONVSTCLK	Input	Convert start input, connected to Xilinx System Monitor.	High
	VAUXN[15:0]	Input	Auxiliary analog input, connected to Xilinx System Monitor.	-
	VAUXP[15:0]	Input	Auxiliary analog input, connected to Xilinx System Monitor.	-
	VN	Input	Dedicated analog-input, connected to Xilinx System Monitor.	-
	VP	Input	Dedicated analog-input, connected to Xilinx System Monitor.	-
SYSMONO	ALM[2:0]	Output	Alarm outputs, connected to Xilinx System Monitor.	High
	OT	Output	Over-Temperature alarm output, connected to Xilinx System Monitor.	High
	EOC	Output	End of Conversion, connected to Xilinx System Monitor.	High
	EOS	Output	End of Sequence, connected to Xilinx System Monitor.	High
	CHANNEL[4:0]	Output	Channel selection, connected to Xilinx System Monitor.	-

* see GRLIB IP Library User's Manual

85.8 Signal definitions and reset values

The signals and their reset values are described in table 1713.

*Table 1713.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
convst	Input	Convert start input, connected to Xilinx System Monitor if the *extconvst* VHDL generic is set to '1'.	Logical 1	-
convstclk	Input	Convert start input, connected to Xilinx System Monitor.	-	-
vauxn[15:0]	Input	Auxiliary analog input, connected to Xilinx System Monitor.	-	-
vauxp[15:0]	Input	Auxiliary analog input, connected to Xilinx System Monitor.	-	-
vn	Input	Dedicated analog-input, connected to Xilinx System Monitor.	-	-
vp	Input	Dedicated analog-input, connected to Xilinx System Monitor.	-	-
alm[2:0]	Output	Alarm outputs, connected to Xilinx System Monitor.	Logical 1	-
ot	Output	Over-Temperature alarm output, connected to Xilinx System Monitor.	Logical 1	-
eoc	Output	End of Conversion, connected to Xilinx System Monitor.	Logical 1	-
eos	Output	End of Sequence, connected to Xilinx System Monitor.	Logical 1	-
channel[4:0]	Output	Channel selection, connected to Xilinx System Monitor.	-	-

85.9 Library dependencies

Table 1714 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1714.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	MISC	Component, signals	Component and signal definitions
GRLIB	AMBA	Signals	AMBA signal definitions

85.10 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.misc.all;

entity grsysmon_ex is
  port (
    clk       : in  std_ulogic;
    rstn      : in  std_ulogic
    );
end;
```

```
architecture rtl of grsysmon_ex is
  -- AMBA signals
  signal ahbsi  : ahb_slv_in_type;
  signal ahbso  : ahb_slv_out_vector := (others => ahbs_none);
  ...
  -- GRSYSMON signals
  signal sysmoni : grsysmon_in_type;
  signal sysmono : grsysmon_out_type;

begin

  -- AMBA Components are instantiated here
  ...

  -- GRSYSMON core is instantiated below
  sysm0 : grsysmon generic map (tech => virtex5, hindex => 4,
     hirq => 4, caddr => 16#002#, cmask => 16#fff#,
     saddr => 16#003#, smask => 16#fff#, split => 1, extconvst => 0)
    port map (rstn, clk, ahbsi, ahbso(4), sysmoni, sysmono);
  sysmoni <= grsysmon_in_gnd; -- Inputs are all driven to '0'

end;
```

86 GRUSBDC - USB Device controller

86.1 Overview

The Universal Serial Bus Device Controller provides a USB 2.0 function interface accessible from an AMBA-AHB bus interface. The core must be connected to the USB through an external PHY (shown in figure 235) compliant to either UTMI, UTMI+ or ULPI. Both full-speed and high-speed mode are supported.

Endpoints are controlled through a set of registers accessed through an AHB slave interface. Each of the up to 16 IN and 16 OUT endpoints can be individually configured to any of the four USB transfer types.

USB data cargo is moved to the core's internal buffers using a master or a slave data interface. The data slave interface allows access directly to the internal buffers using AHB transactions and therefore does not need external memory. This makes it suitable for slow and simple functions. The data master interface requires an additional AHB master interface through which data is transferred autonomously using descriptor based DMA. This is suitable for functions requiring large bandwidth.

These two interfaces are mutually exclusive and cannot be present in the same implementation of the core.

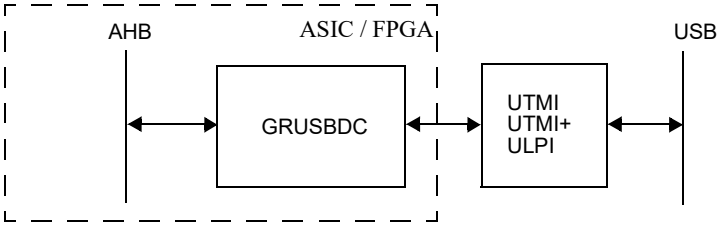


Figure 233. GRUSBDC connected to an external PHY device.

86.2 Operation

86.2.1 System overview

Figure 234 shows the internal structure of the core. This section briefly describes the function of the different blocks.

The Speed Negotiation Engine (SNE) detects connection by monitoring the VBUS signal on the USB connector. When a steady 5 V voltage is detected the SNE waits for a reset and then starts the High-speed negotiation. When the Speed negotiation and reset procedure is finished the selected speed mode (full-speed or high-speed) is notified to the Serial Interface Engine (SIE) which now can start operation. The SNE also detects and handles suspend and resume operations.

The SIE is enabled when the SNE notifies that the reset procedure has finished. It then waits for packets to arrive and processes them according to the USB 2.0 specification. The data cargo is stored to an internal buffer belonging to the recipient endpoint.

The AHB Interface Engine AIE is responsible for transferring USB data cargo from the endpoint's internal buffers to the AHB bus using descriptor based DMA through an AHB master interface when configured in master mode or by direct accesses to the AHB slave interface when configured in slave mode.

For received data it is then up to the external (to the device controller) function to continue processing of the USB data cargo after it has been transferred on the AHB bus. The function is the application specific core which determines the functionality of the complete USB device. It sets up endpoints in the device controller and notifies their existence through the appropriate USB descriptors. When the function wants to transmit a packet it either uses the slave interface to write to the endpoint buffers or establishes a DMA transfer.

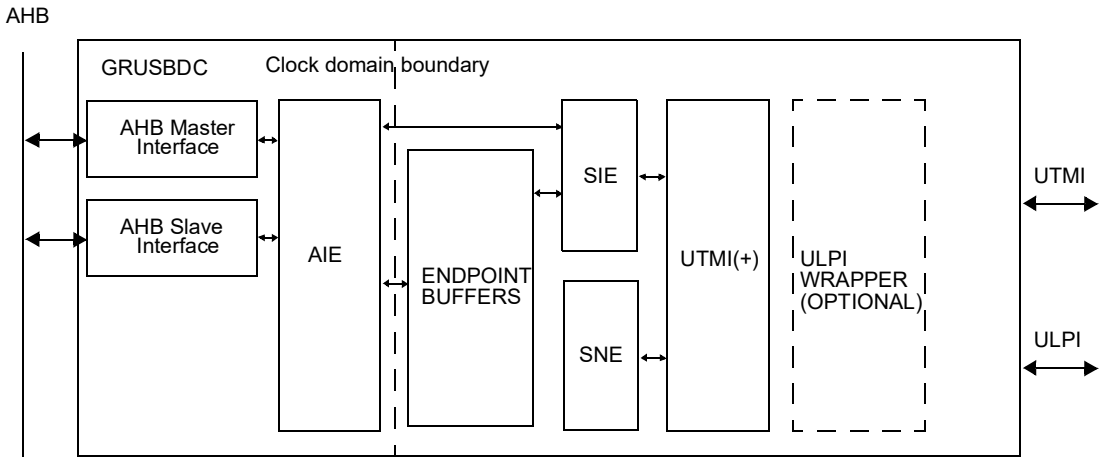


Figure 234. Block diagram of the internal structure of the core.

86.2.2 PHY interface

The core supports three different interfaces to the external PHY which is used to connect to the USB bus. The supported interfaces are UTMI, UTMI+ and ULPI. UTMI+ is an extension of the UTMI specification with optional additional support for host controllers and on-the-go devices while UTMI only supports devices. There are different so called levels in the UTMI+ specification, each with an added degree of support for hosts and on-the-go. The lowest level and common denominator for all the levels is identical to UTMI and uses the exact same signals. The core only supports devices and thus the support for UTMI+ refers to level 0. The data path to the UTMI/UTMI+ cores can be 8-bits or 16-bits wide and also uni- or bi-directional. All combinations are supported by the core.

The UTMI+ Low Pin Interface (ULPI) specifies a generic reduced pin interface and how it can be used to wrap a UTMI+ interface. The core has UTMI/UTMI+ as the main interface (they are identical) and when ULPI is used an extra conversion layer is added. When ULPI is enabled, the UTMI layer is always in 8-bit mode since this is what is required by the ULPI specification.

86.2.3 Speed Negotiation Engine (SNE)

The SNE detects attach, handles reset, high-speed handshake and suspend/resume operation. It also contains support for the various test-modes, which all USB device have to support.

The attached state is entered when a valid VBUS signal is detected. After this the core waits for a USB reset and then starts the high-speed handshake which determines whether full-speed or high-speed mode should be entered. No bus traffic will be accepted by the core until a valid reset has been detected.

The core supports soft connect/disconnect which means that the pull-up on the D+ line can be controlled from a user accessible register. The pull-up is disabled after reset and thus the function implementation has full control over when the device will be visible to the host.

The SNE also continuously monitors for the suspend condition (3 ms of idle on the USB bus) when the suspend state will be entered. The suspend state is left either through an USB reset or resume signaling. The resume signaling can come from either a downstream facing port (hub or host controller) or the device itself (Remote wakeup). The device controller core can generate remote wakeup signaling which is activated through an user accessible register. If this feature is used the function should indicate this in the descriptor returned to a device GetStatus request.

Transactions are only handled by the SIE if the SNE is in full-speed or high-speed mode. When in suspend, notattached, attached or during the reset process all transactions will be dropped.

The current status of the SNE such as VBUS valid, suspend active, USB reset received and current speed mode can be accessed through a status register. Each of these status bits have a corresponding interrupt enable bit which can be used to generate interrupts when a change occurs in the status bits.

If full-speed only mode is desired the core can be set to not perform the high-speed handshake through a register.

The different test-modes required by the USB standard are also enabled through user accessible registers. When enabled they can only be left by power-cycling the complete core or resetting the device controller (by using the rst signal to the core, not an USB reset).

The test modes are Test_SE0_NAK, Test_J, Test_K and Test_Packet.

In Test_SE0_NAK mode the high-speed receiver is enabled and only valid IN transactions (CRC correct, device and endpoint addresses match, PID is not corrupt) are responded to with a NAK.

Test_J continuously drives a high-speed J state.

Test_K continuously drives a high-speed K state.

Test_Packet repetitively sends a test packet. Please refer to the USB 2.0 standard for the packet contents. Minimum interpacket delay when device is sending two or more consecutive packets seems not to be specified in the standard. The core uses 192 bit times as its minimum delay which is the maximum value of the various minimum delays in the standard for any packet sequence and should therefore be compliant.

The core also supports a functional test-mode where all timeouts in the speed-negotiation engine have been shortened to eight clock cycles. This intended to be used in simulations and for ASIC testers where time and test-vector length respectively are important.

86.2.4 Serial Interface Engine (SIE)

The SIE handles transmission and reception of USB packets. The core will not respond to any transactions until a reset has been received and either full-speed or high-speed mode has been successfully entered.

The SIE always begins with waiting for a token packet. Depending on the type of token, data is either transferred from the core to the host or in the opposite direction. Special tokens are handled without any data transfers. The special tokens are PING and SOF which cause only a handshake to be sent or the frame number to be stored respectively. IN tokens initiate transfers to the host while SETUP and OUT tokens initiate transfers to the device. Packets received in the token stage with other PIDs than those mentioned in this paragraph are discarded.

A data packet is transmitted in the next stage if the token determined that data should be transferred to the host. When the data transmission is finished the core waits for a handshake before returning to the token stage.

If data was determined to be transferred to the device the core waits for and receives a data packet and then sends a handshake in return before entering the token stage again.

More detailed descriptions of the SIE and how it interacts with the core function are found in section 86.5.

86.2.5 Endpoint buffers

Each endpoint has two buffers to which packets are stored. The core automatically alternates between them when a packet has been received/transmitted so that data from one of the buffers can be transferred on the AHB bus while a new packet is being received/transmitted on the USB to/from the second buffer.

The state of the buffers affects the handshake sent to the host at the end of a transaction. In high-speed mode BULK OUT endpoints and CONTROL OUT endpoints which are not in the SETUP stage support the PING protocol. This means that at the end of an OUT transaction to one of these endpoints the device should return ACK if it could accept the current data and has space for another packet. For the USB device controller this is done if the second buffer for the endpoint is empty when the transaction is ready.

If the second buffer is non-empty a NYET is sent instead. If the current data could not be accepted (both buffers non-empty when the packet arrives) a NAK is returned.

For other endpoint types in high-speed mode and all endpoint types in full-speed mode an ACK is always returned if the data is accepted and a NAK if it could not be accepted.

An endpoint buffer can be configured to be larger than the maximum payload for that endpoint. For IN endpoints the writing of data larger than the maximum payload size to a buffer will result in a number of maximum sized packets being transferred ending with a packet smaller than or equal to the maximum size.

In the OUT direction larger buffers are only used for high-bandwidth endpoints where more than one transaction per microframe can occur for that endpoint. In that case the data from all packets during one microframe is stored in the order it arrives to a single buffer and is then handed over to the AHB interface. All non-high-bandwidth endpoints always store one packet data cargo to a buffer.

The endpoint buffers do not use separate physical RAM blocks in hardware instead they reside consecutively in the same memory space to avoid wasting memory.

86.2.6 AMBA Interface Engine (AIE)

The AIE can either be configured in slave mode or master mode. This is selected in synthesis process with a VHDL generic. Both cannot be present at the same time. The two interfaces will be described separately in this section.

Master interface

In master mode an AHB master interface is included in the core and handles all data transfers to and from the cores internal buffers using DMA operations. The DMA operation is described in detail in section 86.3. There is a separate DMA engine in the IN direction and the OUT direction respectively. They are multiplexed on the single master interface available for the core on the AHB bus. This scheme is used to limit the load on the AHB bus.

If both engines request the bus at the same time the owner will always be switched. That is, if the OUT direction DMA engine currently was allowed to make an access and when finished it still requests the bus for a new transaction and at the same time the IN direction engine also requests the bus, the IN direction will be granted access. If the situation is the same after the next access ownership will be switched back to the OUT engine etc.

The IN engine only reads data (note that this only applies to DATA, descriptor status is written) from the bus and always performs word transfers. Any byte alignment and length can still be used since this will only cause the core to skip the appropriate amount of leading and trailing bytes from the first and last words read.

The OUT engine writes data to the bus and performs both word and byte transfers. If the start transfer for an access is not word-aligned byte writes will be performed until a word boundary is reached. From then and onwards word writes are performed in burst mode until less than 4 bytes are left. If remaining number of bytes is not zero byte writes are performed for the last accesses. The byte accesses are always done as single accesses.

The bursts are of type incremental burst of unspecified length (refer to AMBA specification for more details). The core can only operate in big-endian mode that is the byte at the lowest address in a word is the most significant byte. This corresponds to bits 31 downto 24 in the GRLIB implementation. The

first byte received on the USB will be stored to the msb location. In a single byte the lowest bit index corresponds to the first bit transmitted on the USB.

Slave interface

The slave interface is used for accessing registers and also for data transfers when the core is configured in slave mode. Byte, half-word and word accesses are supported. At least one waitstate is always inserted due to the pipelined nature of the interface but the upper limit is not fixed due to registers being accessed across clock domains. The maximum number of waitstates will thus depend on the difference in clock frequency between the USB and AHB clock domains. An upper bound can be calculated when the clock frequencies have been determined.

86.2.7 Synchronization

There are two clock domains in the core: the AHB clock domain and the USB clock domain. The AHB clock domain runs on the same clock as the AHB bus while the USB domain runs on the UTMI or ULPI clock. The boundary is between then AIE, SIE and Endpoint buffers. All signals between the two domains are synchronized and should be declared as false paths during synthesis.

86.2.8 Reset generation

The main reset (AMBA reset) resets AHB domain registers, whereas the USB reset resets registers in USB clock domain. There is no internal reset generation or synchronization between both AMBA and USB resets, they are independent inputs. If necessary, the reset generation shall be done in a higher instance, externally to this IP.

Endpoint specific registers related to the state of the USB protocol in the SIE are reset when an USB reset is received.

86.2.9 Synthesis

All number of endpoints with up to maximum size payloads cannot be supported due to limitations in the RAM block generator size in GRLIB. The maximum size also varies with technology. Note that large buffers can also have a large timing impact at least on FPGA since the a large RAM buffer will consist of several separate physical block RAMs located at different places causing large routing delays.

As mentioned in section 86.2.7, signals between the clock domains are synchronized and should be declared as false paths.

The complete AHB domain runs at the same frequency as the AHB bus and will be completely constrained by the bus frequency requirement.

The USB domain runs on different frequencies depending on the data path width. In 8-bit mode the frequency is 60 MHz and in 16-bit mode it is 30 MHz. Input and output constraints also need to be applied to the signals to and from the PHY. Please refer to the PHY documentation and/or UTMI/ULPI specification for the exact values of the I/O constraints.

86.2.10 Functional test-mode

A functional test-mode can be enabled in the core using the functesten VHDL generic. The functional test-mode is intended to reduce the number of required test-vectors during functional testing of an ASIC chip. During normal operation it would be required to go through the whole speed detection sequence before being able to start USB transactions. Since the speed detection takes a relatively long time this would make the test-vector amount very large often making it incompatible with existing test equipment.

In functional test-mode the core shortens the speed detection thus making it possible to test the functionality without a long initial delay. The test-mode can be disabled using the FT control register bit.

86.2.11 Scan test support

The VHDL generic *scantest* enables scan test support. If the core has been implemented with scan test support it will:

- disable the internal RAM blocks when the testen and scanen signals are asserted.
- use the testoen signal as output enable signal.
- use the testrst signal as the reset signal for those registers that are asynchronously reseted.

The testen, scanen, testrst, and testoen signals are routed via the AHB slave interface.

86.3 DMA operation

DMA operation is used when the core is configured in AHB master mode. Each IN and each OUT endpoint has a dedicated DMA channel which transfers data to and from the endpoint's internal buffers using descriptor based autonomous DMA. Each direction (IN and OUT) has its own DMA engine which requests the AHB master interface in contention with the other direction. Also each endpoint in a direction contends for the usage of the DMA engine with the other endpoints in the same direction. The arbitration is done in a round-robin fashion for all endpoints which are enabled and have data to send or receive.

The operation is nearly identical in both directions and the common properties will be explained here while the differences are outlined in the two following sub-sections.

The DMA operation is based on a linked list of descriptors located in memory. Each endpoint has its own linked list. The first word in a descriptor is the control word which contains an enable bit that determines whether the descriptor is active or not and other control bits. The following word is a pointer to a memory buffer where data should be written to or read from for this descriptor. The last word is a pointer to the location of the next descriptor. A bit in the control word determines if the next descriptor pointer is valid or not. If not valid the descriptor fetching stops after the current descriptor is processed and the DMA channel is disabled.

The DMA operation is started by first setting up a list with descriptors in memory and then writing a pointer to the first descriptor to the endpoint's descriptor pointer register in the core and setting the descriptor available bit. The pointer register is updated as the list is traversed and can be read through the AHB slave interface. When the list is ended with a descriptor that has its next descriptor available bit disabled the list must not be touched until the core has finished processing the list and the channel is disabled. Otherwise a deadlock situation might occur and behavior is undefined.

Another way to use the linked list is to always set the next descriptor available bit and instead make sure that the last descriptor is disabled. This way new descriptors can be added and enabled on the fly to the end of the list as long as the descriptor available bit is always set after the new descriptors have

been written to memory. This ensures that no dead lock will occur and that no descriptors are missed. Figure 235 shows the structure of the descriptor linked list.

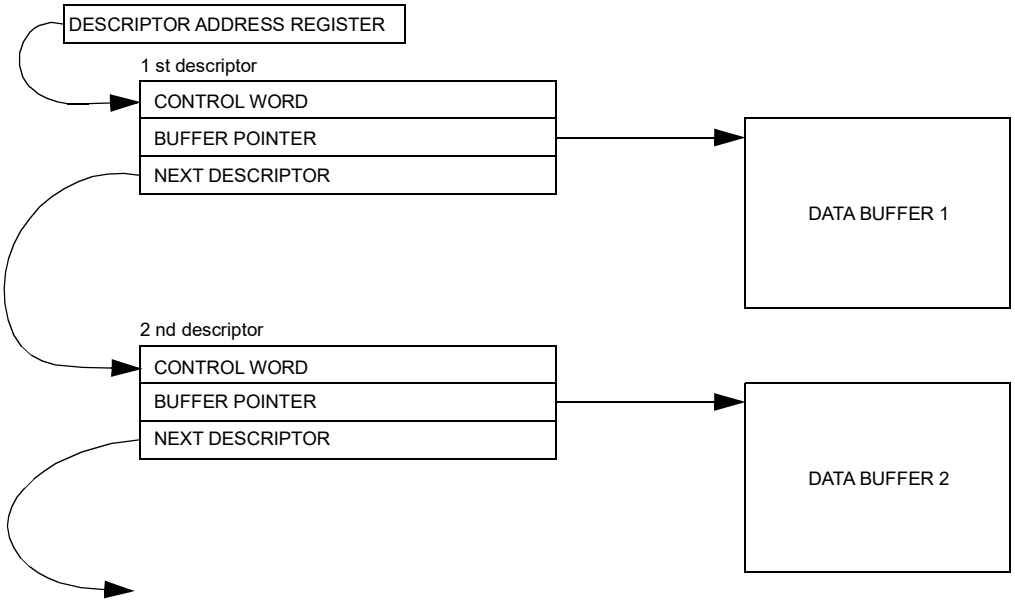


Figure 235. Example of the structure of a DMA descriptor linked list in memory.

86.3.1 OUT endpoints

The DMA operation for OUT endpoints conforms to the general description in the previous subsection. There are small differences in individual bits and the meaning of the length field. The contents of the different descriptor words can be found in the tables below.

When a descriptor has been enabled it will be fetched by the core when the descriptor available bit is set and as soon as a buffer for the corresponding endpoint contains data received from the USB it will be written to memory starting from the address specified in the buffer pointer word of the descriptor. The contents of a single internal memory buffer is always written to a single descriptor buffer. This always corresponds to a single USB packet except for high-bandwidth isochronous and interrupt endpoints. The number of bytes written is stored in the length field when writing is finished which is indicated by the enable bit being cleared. Then the SETUP status bit will also be valid. When the enable bit is cleared the memory location can be used again.

Interrupts are generated if requested as soon as the writing to memory is finished. The endpoint can also be configured to generate an interrupt immediately when a packet has been received to the internal buffers. This can not be enabled per packet since the core cannot associate a received packet with a specific descriptor in advance. This interrupt is enabled from the endpoint’s control register.

When the data has been fetched from the internal buffer it is cleared and can be used by the SIE again for receiving a new packet.

*Table 1715.*OUT descriptor word 0 (address offset 0x0) ctrl word

31	18	17	16	15	14	13	12	0
RESERVED		SE	RE	IE	NX	EN	LENGTH	

31: 18	RESERVED
17	Setup packet (SE) - The data was received from a SETUP packet instead of an OUT.
16	RESERVED

Table 1715. OUT descriptor word 0 (address offset 0x0) ctrl word

15	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when the packet from this descriptor has been read to the internal buffers and handed over to the SIE. This does not mean that packet has also been transmitted.
14	Next descriptor available (NX) - The next descriptor field is valid and points to the next descriptor.
13	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
12: 0	LENGTH - The number of bytes received. Valid when the EN bit has been cleared by the core.

Table 1716. OUT descriptor word 1 (address offset 0x4) Buffer pointer

31	0
ADDRESS	

31: 2	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.
1: 0	RESERVED

Table 1717. OUT descriptor word 2 (address offset 0x8) Next descriptor pointer

31	2	1	0
NDP		RES	

31: 2	Next descriptor pointer (NDP) - Pointer to the next descriptor.
1: 0	RESERVED

86.3.2 IN endpoints

The DMA operation for IN endpoints conforms to the general DMA description. There are small differences in individual bits and the meaning of the length field. The contents of the different descriptor words can be found in the tables below.

When a descriptor has been enabled and the descriptor available bit is set the core will start processing the descriptor and fetch the number of bytes indicated in the length field to an internal buffer belonging to the endpoint as soon as one is available. An interrupt will be generated if requested when data has been written to the internal buffer and status has been written back to the descriptor. The packet might not have been transmitted on the USB yet.

A separate interrupt is available which is generated when the packet has actually been transmitted. It needs to be enabled from the endpoint's control register and also in the descriptor (using the PI bit) for each packet that should generate the interrupt.

A descriptor with length zero will result in a packet with length zero being transmitted while a length larger than the maximum payload for the endpoint will result in two or more packets with all but the last being of maximum payload in length. The last transaction can be less than or equal to the maximum payload. If the length field is larger than the internal buffer size the data will not written to the internal buffer and status will be immediately written to the descriptor with an error bit set.

When the more bit is set the data from the current descriptor is written to the internal buffer and it then continues to the next descriptor without enabling the buffer for transmission. The next descriptor's data is also read to the same buffer and this continues until a descriptor is encountered which does not have more set.

If the total byte count becomes larger than the internal buffer size the packet is not sent (the data from the internal buffer is dropped) and the ML bit is set for the last descriptor. Then the descriptor fetching starts over again.

If the next bit is not set when the more bit is, the core will wait for a descriptor to be enabled without letting other endpoints access the AHB bus in between.

*Table 1718.*IN descriptor word 0 (address offset 0x0) ctrl word

31	19	18	17	16	15	14	13	12	0
RESERVED		MO	PI	ML	IE	NX	EN	LENGTH	

31: 19	RESERVED
18	More (MO) - The data from the next descriptor should be read to the same buffer.
17	Packet sent interrupt (PI) - Generate an interrupt when packet has been transmitted on the USB.
16	Maximum length violation (ML) - Attempted to transmit a data cargo amount larger than the buffer.
15	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when the packet from this descriptor has been read to the internal buffers and handed over to the SIE. This does not mean that packet has also been transmitted.
14	Next descriptor available (NX) - The next descriptor field is valid and points to the next descriptor.
13	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
12: 0	LENGTH - The number of bytes to be transmitted.

*Table 1719.*IN descriptor word 1 (address offset 0x4) Buffer pointer

31	0
ADDRESS	

31: 2	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.
1: 0	RESERVED

*Table 1720.*IN descriptor word 2 (address offset 0x8) Next descriptor pointer

31	2	1	0
NDP		RES	

31: 2	Next descriptor pointer (NDP) - Pointer to the next descriptor.
1: 0	RESERVED

86.4 Slave data transfer interface operation

The AHB slave interface is used for data transfers instead of the DMA interface when the core is configured in AHB slave mode. This mode is selected by setting the aiface VHDL generic to 0. In this mode the core's internal buffers containing data to/from USB packets are accessed directly using AHB read and write transfers. This is usually much slower than DMA but much simpler and does not require any external memory, thus suitable for slow devices which need to be small and simple.

As for the DMA mode each endpoint is operated separately using four registers at the same addresses. Two of them, the control and status registers, are the same while the DMA control and the descriptor address registers have been replaced with the slave control and slave write/read data registers. The slave interface does not use descriptors so these four registers provides the complete control of the endpoint.

The details for data transfers in the two different endpoint directions will be explained in separate sections.

86.4.1 OUT slave endpoint

As stated earlier, in slave mode the core's buffers are accessed directly from the AHB bus through the slave interface. For OUT endpoints it has to be checked that data is available in the selected buffer and then reserve it. This is done using the slave control register by writing a one to the CB bit. The CB bit is always automatically cleared when the write access is finished and then the BS, DA and BUFCNT

read-only bits/fields contain valid information. The BS bit is set to 0 if buffer 0 is currently selected and to 1 if buffer 1 is selected. DA is set to 1 if data is available in the buffer and in that case BUFCNT contains the number of bytes.

If data is available (DA is 1) it can be read from the slave data read register. One byte at a time is read using byte accesses, two bytes using half-word accesses and four bytes using word accesses. No other widths are supported. In each case data is available from bit 31 and downwards regardless of the value of the two least significant address bits. This is summarized in table 1721. The BUFCNT field is continuously updated when reading the buffer so that it can be monitored how many bytes are left.

Table 1721. AHB slave interface data transfer sizes

Size (byte)	AHB transfer size (HSIZE)	Data alignment (HRDATA)
1	byte (000)	31:24
2	half-word (001)	31:16
4	word (010)	31:0

When all the data has been read, a new buffer can be acquired by writing a one to CB. In this case when a buffer is currently reserved and the DA bit is set it will be released when CB is written. If a new buffer was available it will be reserved and DA is set to 1 again. If no new buffer is available DA will be 0 and the process has to be repeated. The current buffer (if one is selected) will be released regardless of whether a new one is available or not. Also, if all data has not been read yet when a buffer change request is issued the rest of the data will be lost.

The core does not have to be polled to determine whether a packet is available. A packet received interrupt is available which can be enabled from the control register and when set an interrupt will be generated each time a packet is stored to the internal buffers. The status of the buffers can also be read through the endpoint's status register without actually reserving the buffer.

One buffer consists of the data payload from one single packet except for high-bandwidth interrupt and isochronous endpoints for which up to three packet data payloads can reside in a single buffer.

A buffer does not have to be read consecutively. Buffers for several endpoints can be acquired simultaneously and read interleaved with each other.

86.4.2 IN slave endpoint

The slave operation of IN endpoints is mostly identical to that for OUT. For IN endpoints it has to be checked that a buffer is free and then reserve it before writing data to it. This is done using the slave control register by writing a one to the EB bit. The EB bit is always automatically cleared when the write access is finished and then the BS, BA and BUFCNT fields are updated. The BS bit is set to 0 if buffer 0 is currently selected and to 1 if buffer 1 is selected. BA is set to 1 if a buffer is available to write data to. BUFCNT contains the number of bytes currently written to the selected buffer. It is cleared to zero when a new buffer is acquired.

If a buffer is available (BA is 1) data can be written through the slave data write register. One byte at a time is written using byte accesses, two bytes using half-word accesses and four bytes using word accesses. No other widths are supported. In each case data should be placed from bit 31 and downwards regardless of the value of the two least significant address bits. This is summarized in table 1722.

Table 1722. AHB slave interface data transfer sizes

Size (byte)	AHB transfer size (HSIZE)	Data alignment (HWDATA)
1	byte (000)	31:24
2	half-word (001)	31:16
4	word (010)	31:0

When all the data has been written, the buffer is enabled for transmission by writing a one to EB. If a new buffer was available it will be reserved and BA is set to 1 again. If no new buffer is available BA will be 0 and the process has to be repeated. The current buffer (if one is reserved) will be enabled for transmission regardless of whether a new one is available or not.

The core does not have to be polled to determine whether a buffer is available. A packet transmitted interrupt is available which can be enabled from the endpoint control register. Then when enabling a packet for transmission the PI bit in the slave control register can be set which will be cause an interrupt to be generated when this packet has been transmitted and cleared from the internal buffers. The status of the buffers can also be read through the endpoint's status register without actually reserving the buffer.

Maximum payload size packets will be generated from the buffer until the last packet which will contain the remaining bytes.

A buffer does not have to be written consecutively. Buffers for several endpoints can be acquired simultaneously and written interleaved with each other. This will however cause additional waitstates to be inserted.

86.5 Endpoints

An endpoint needs to have both its AHB and USB function configured before usage. The AHB configuration comprises the DMA operation described in the previous section. The USB configuration comprises enabling the endpoint for USB transactions, setting up transfer type (control, bulk, isochronous, interrupt), payload size, high-bandwidth among others. The configuration options are accessed through a register available from the AHB slave interface. See section 86.8 for the complete set of options.

When setting the configuration options the endpoint valid bit should be set. This will enable transfers to this endpoint as soon as a USB reset has been received. If the endpoint is reconfigured the valid bit must first be set to zero before enabling it again. Otherwise the endpoint will not be correctly initialized. When the endpoint is enabled the toggle scheme will be reset and data buffers cleared and buffer selectors set to buffer zero. The maximum payload, number of additional transactions and transfer type fields may only be changed when endpoint valid is zero or when setting endpoint valid to one again after being disabled. Other bits in the endpoint control register can be changed at any time.

No configuration options should be changed when the endpoint is enabled except the halt, control halt and disable bits.

An endpoint can also be halted by setting the halt bit of the endpoint. This will cause all transactions to receive a STALL handshake. When clearing the halt condition the toggle scheme will also be reset as required by the USB standard.

When the endpoint is setup, data transfers from and to the endpoint can take place. There is no difference in how data is transferred on the AHB bus depending on the selected transfer type. This only affects the transfers on the USB. For control endpoints some extra handling is required by the core user during error conditions which will be explained in the control endpoint section.

Packets that are received to an endpoint (independent of endpoint type) with a larger payload than the configured maximum value for the endpoint will receive a STALL handshake and cause the endpoint to enter halt mode.

When a USB reset is detected the CS, ED and EH bits of the endpoint control register will be cleared for all endpoints. The EV bit will also be cleared except for control endpoint 0.

The CB bit in the endpoint control register is for clearing the internal buffers of an endpoint. When set the data will be discarded from the buffers, the data available bits for the corresponding buffers will be cleared and the CB bit is cleared when it is done. This will however not work if a transaction is currently active to the same endpoint so this feature must be used with caution.

86.5.1 Control endpoints

Endpoint 0 must always be a control endpoint according to the USB standard and be accessible as soon as a USB reset is received. The core does not accept any transactions until a USB reset has been received so this endpoint can be enabled directly after power up. More control endpoints can be enabled as needed with the same constraints as the default control endpoint except that they should not be accessible until after configuration. If the function controlling the core is slow during startup it might not be able complete configuration of endpoint 0 before a USB reset is received. This problem is avoided in the core because its pull-up on D+ is disabled after reset which gives the function full control of when the device will be visible on the bus.

A control endpoint is a message pipe and therefore transfers data both in the IN and OUT direction. Thus control endpoints must use the endpoint in both directions with the same number in the device controller. This requires both to be configured in the same mode (same transfer type, payload ...). Otherwise device behavior is undefined.

A control transfer is always started with a SETUP transaction which will be received to the OUT endpoint. If the control transfer is a write the subsequent data phase will be in the OUT direction and this data will also be received to the OUT endpoint. The function should read both the setup data and the other data cargo and respond correspondingly. If the request was valid the function should enable a zero length packet for the endpoint in the IN direction which will lead to a valid status stage. If an error is detected it should instead halt the endpoint. There are two alternatives for this: A non-clearing halt which will last even after the next SETUP transaction or a clearing halt which will be removed when the next SETUP is received. The latter is the recommended behavior in the USB standard since the other will require the complete core to be reset to continue operation if the permanent halt appears on the default control endpoint.

The core can detect errors in single transactions which cause the endpoint to enter halt mode automatically. In this case the clearing halt feature will be used for control endpoints.

If a SETUP transaction indicates a control read the data phase will be in the IN direction. In that case the core user should enable data for the endpoint in the IN direction if the request was accepted otherwise the halt feature should be set. The transfer is finished when the host sends a zero length packet to the OUT endpoint.

Note that when entering halt for a control endpoint both the IN and OUT endpoints halt bits should be set.

Each time a control endpoint receives a setup token the buffers in the IN direction are emptied. This is done to prevent inconsistencies if the data and status stage were missing or corrupted and thus the data never fetched. The old data would still be in the buffer and the next setup transaction would receive erroneous data. The USB standard states that this can happen during error conditions and a new SETUP is transmitted before the previous transfer finished. The core user can also clear the buffer through the IN endpoint control register and is encouraged to do this when it detects a new SETUP before finishing the previous transfer. This must be done since the user might have enabled buffers after the core cleared them when receiving the new SETUP.

Whether data received to a descriptor for an OUT endpoint was from a SETUP transaction or an OUT transaction is indicated in a descriptor status bit.

86.5.2 Bulk endpoints

Bulk endpoints are stream pipes and therefore only use a single endpoint in either the IN or OUT direction. The endpoint with the same number in the other direction can be used independently. Data is accessed normally through the AHB interface and no special consideration need to be taken apart from the general endpoint guidelines.

86.5.3 Interrupt endpoints

Interrupt data is handled in the same manner as for bulk endpoints on the AHB interface. The differences only appear on the USB. These endpoints are also of stream type.

Interrupt endpoints support a high-bandwidth state which means that more than one transaction per microframe is performed. This necessitates buffers larger than the maximum payload size. The endpoint should be configured with a buffer larger or equal to the maximum payload times the number of transactions. All transactions will be received to/transmitted from the same buffer. The endpoint is configured as a high-bandwidth endpoint by setting the number of additional transactions to non-zero in the endpoint control register.

86.5.4 Isochronous endpoints

Isochronous endpoints are of stream type are identical to other endpoints regarding the handling on the AHB bus.

A big difference between isochronous endpoints and the other types is that they do not use handshakes. If no data is available when an IN token arrives to an Isochronous endpoint a data packet with length 0 is transmitted. This will indicate to the host that no error occurred but data was not ready. If no packet is sent the host will not know whether the packet was corrupted or not.

When not in high-bandwidth mode only one transaction in the OUT direction will be stored to a single buffer. In high-bandwidth mode all transactions during a microframe are stored to the same buffer.

In the IN direction data is always transferred from the same buffer until it is out of data. For high-bandwidth endpoints the buffer should be configured to be the maximum payload times the number of transactions in size.

Isochronous high-bandwidth endpoints use PID sequencing. When an error is detected in the PID sequence in the OUT direction no data is handed over to AHB domain for the complete microframe.

86.6 Device implementation example in master mode

This section will shortly describe how the USB device controller can be used in master mode.

A function controlling the device controller and implementing the actual application specific device will be needed. It can be either hardware, software or a combination. The only requirement is that it can control the device controller through the AHB bus.

The first thing needed for successful operation is a correctly configured PHY. This is automatically done by the device controller.

After this the device controller waits for attachment to the USB bus indicated by the VBUS becoming valid. This can be notified to the function either by polling or an interrupt. The time of attachment can be controlled by the function through the pull-up enable/disable bit in the core control register. When disabled the USB host will not notice the device even when it is plugged in.

After attachment a USB reset needs to be received before transactions are allowed to be accepted. This can also notified by polling or an interrupt.

Only control endpoint 0 should be accessible after reset. The function is responsible for enabling and configuring at the right time. It can wait until a USB reset has been received but it is easier to enable it immediately after power-up. This can be done since the device controller will not accept any transactions until USB reset has been received. When enabling the endpoint descriptors should also be enabled for both the IN and OUT direction and also the descriptor available bits should be set.

Then the endpoint is ready to accept packets and the function should wait for SETUP packets arriving. It can be notified of packets arriving either through polling or interrupts. The core should process the requests and return descriptors as requested. When a Set address request is received the function should write the new address to the device controller’s global control register. It will take effect after

the next successful IN transaction for the control endpoint. This should correspond to the status stage of the Set address transfer.

When a Set configuration request is received the function should enable the appropriate interface and endpoints according to the selected configuration. This is done by writing to the various endpoint control registers. The function is responsible for advertising the configurations and interfaces through the descriptors requested by SETUP transactions.

When the endpoints for the selected configuration are enabled the function should also setup the DMA operation. Then it is ready to transmit and receive data through the application specific endpoints. Interrupts can be used to notify that new packets have transferred and then polling will determine which endpoint had a status change.

86.7 Device implementation example in slave mode

This section will shortly describe how the USB device controller can be used in slave mode.

A function controlling the device controller and implementing the actual application specific device will be needed. It can be either hardware, software or a combination. The only requirement is that it can control the device controller through the AHB bus.

The first thing needed for successful operation is a correctly configured PHY. This is automatically done by the device controller.

After this the device controller waits for attachment to the USB bus indicated by the VBUS becoming valid. This can be notified to the function either by polling or an interrupt. The time of attachment can be controlled by the function through the pull-up enable/disable bit in the core control register. When disabled the USB host will not notice the device even when it is plugged in.

After attachment an USB reset needs to be received before transactions are allowed to be accepted. This can also notified by polling or an interrupt.

Only control endpoint 0 should be accessible after reset. The function is responsible for enabling and configuring at the right time. It can wait until a USB reset has been received but it is easier to enable it immediately after power-up. This can be done since the device controller will not accept any transactions until USB reset has been received. When enabling the endpoint packet interrupts should be enabled or the function should start polling the buffer status so that it will notice when packets arrive.

Then the endpoint is ready to accept packets and the function should wait for SETUP packets arriving. When a packet arrives the core should process the requests and return USB descriptors as requested. When a Set address request is received the function should write the new address to the device controller's global control register. It will take effect immediately. It should not be written until the status stage has been finished for the Set address request. It can be determined that the request has finished if a packet transmitted interrupt is enabled for the handshake packet of the request and an interrupt is received.

When a Set configuration request is received the function should enable the appropriate interface and endpoints according to the selected configuration. This is done by writing to the various endpoint control registers. The function is responsible for advertising the configurations and interfaces through the descriptors requested by SETUP transactions.

When the endpoints for the selected configuration are enabled the function should also enable interrupts or start polling these endpoints. Then it is ready to transmit and receive data through the application specific endpoints. Interrupts can be used to notify that new packets have been transferred and then status reads will determine which endpoint had a status change.

86.8 Registers

The core is programmed through registers mapped into AHB address space. Only 32-bit single-accesses to the registers are supported.

*Table 1723.*GRUSBDC registers

AHB address offset	Register
0x00	OUT Endpoint 0 control register
0x04	OUT Endpoint 0 slave ctrl / DMA ctrl register
0x08	OUT Endpoint 0 slave data / DMA descriptor address register
0x0C	OUT Endpoint 0 status register
0x10-0x1C	OUT Endpoint 1
...	
0xF0-0xFC	OUT Endpoint 15
0x100-0x1FC	IN Endpoints 0-15
0x200	Global Ctrl register
0x204	Global Status register

86.8.1 OUT Endpoint Control Register

*Table 1724.*0x00,... - EOCTRL - OUT endpoint control register

31 21	20	19	18	17 7	6 5	4 3	2	1	0
BUFSZ	PI	CB	CS	MAXPL	NT	TT	EH	ED	EV
*	0	0	0	NR	NR	NR	0	0	0
r	rw	rw	rw	rw	rw	rw	rw	rw	rw

31: 21 Buffer size (BUFSZ) - Size/8 in bytes of one hardware buffer slot for this endpoint. Two slots are available for each endpoint.

20 Packet received interrupt (PI) - Generate an interrupt for each packet that is received on the USB for this endpoint (packet has been stored in the internal buffers). Reset value: ‘0’.

19 Clear buffers (CB) - Clears any buffers for the endpoint that contain data if the buffer is not currently active.

18 Control Stall (CS) - Return stall for data and status stages in a control transfer. Automatically cleared when the next setup token is received. Only used when the endpoint is configured as a control endpoint.

17: 7 Maximum payload (MAXPL) - Sets the maximum USB payload (maximum size of a single packet sent to/from the endpoint) size for the endpoint. All bits of the field are not always used. The maximum value for the maximum payload is determined with a generic for each endpoint. Not Reset.

6: 5 Number of transactions (NT) - Sets the number of additional transactions per microframe for high-speed endpoints and per frame for full-speed endpoints. Only valid for isochronous endpoints. Not Reset.

4: 3 Transfer type (TT) - Sets the transfer type for the endpoint. “00”=CTRL, “01” =ISOCH, “10”=BULK, “11”=INTERRUPT. Only OUT endpoints should be set to the CTRL type and then the IN endpoint with the same number will be automatically used. It is important not to use OUT endpoints that do not have a corresponding IN endpoint as a CTRL endpoint. Not Reset.

2 Endpoint halted (EH) - Halt the endpoint. If set, all transfers to this endpoint will receive a STALL handshake. Reset value: ‘0’.

1 Endpoint disabled (ED) - Disables the endpoint. If set, all transfers to this endpoint will receive a NAK handshake. Reset value: ‘0’.

0 Endpoint valid (EV) - Enables the endpoint. If not enabled, all transfers to this endpoint will be ignored and no handshake is sent. Reset value; ‘0’.

86.8.2 OUT Slave Control Register

*Table 1725.*0x04,... - EOSLUCTRL - OUT slave control register.

31 17	16	15 3	2	1	0
RESERVED	SE	BUFCNT	DA	BS	CB
0	0	0	0	0	0
r	r	r	r	r	rw

31: 17 RESERVED

16 Setup packet (SE) - The data was received from a SETUP packet instead of an OUT.

15: 3 Buffer counter (BUFCNT) - The number bytes available(OUT)

2 Data available (DA) - Set to one if a valid packet was acquired when requested using the CB. If no valid packet was available it is set to zero. Reset value: ‘0’.

1 Buffer select (BS) - Current buffer selected. Read only.

0 Change or acquire buffer (CB) - If no buffer is currently active try to acquire a new one. If one is already acquired, free it and try to acquire a new one.

86.8.3 OUT Slave Buffer Read Register

*Table 1726.*0x08,... - EOSLUDATA - OUT slave buffer read register.

31	0
DATA	
NR	
r	

31: 0 Data (DATA) - In AHB slave mode, data is fetched directly from the internal buffer by reading from this register. Data always starts from bit 31. For word accesses bits 31-0 are valid, for half-word bits 31-16 and for byte accesses bits 31-24.

86.8.4 OUT DMA Control Register

*Table 1727.*0x04,... - EODMACTRL - OUT DMA control register.

31	11	10	9	4	3	2	1	0
RESERVED		AE	RESERVED		AD	AI	IE	DA
0		0	0		0	0	0	0
r		wc	r		rw	rw	rw	rw

31: 11 RESERVED

10 AHB error (AE) - An AHB error has occurred for this endpoint.

9: 4 RESERVED

3 Abort DMA (AD) - Disable descriptor processing (set DA to 0) and abort the current DMA transfer if one is active. Reset value: ‘0’.

2 AHB error interrupt (AI) - Generate interrupt when an AHB error occurs for this endpoint.

1 Interrupt enable (IE) - Enable DMA interrupts. Each time data has been received or transmitted to/ from a descriptor with its interrupt enable bit set an interrupt will be generated when this bit is set.

0 Descriptors available (DA) - Set to indicate to the GRUSBDC that one or more descriptors have been enabled.

86.8.5 OUT Descriptor Address Register

*Table 1728.*0x08,... - EODMADESL - OUT descriptor address register.

31	2	1	0
DESCADDR		RES	
NR		00	
rw		r	

31: 2 Descriptor table address (DESCADDR) - Address to the next descriptor.Not Reset.

1: 0 RESERVED

86.8.6 OUT Endpoint Status Register

*Table 1729.*0x0C,... - EOSTAT - OUT endpoint status register

31 30	29	28 16	15 3	2	1	0
RES	PR	B1CNT	B0CNT	B1	B0	BS
0	0	0	0	0	0	0
r	wc	r	r	r	r	r

31: 30 RESERVED.

29 Packet received (PR) - Set each time a packet has been received (OUT) and stored in the internal buffers. Cleared when written with a '1'.

28: 16 Buffer 1 byte count (B1CNT) - Number of bytes in buffer one.

15: 3 Buffer 0 byte count (B0CNT) - Number of bytes in buffer zero.

2 Buffer 1 data valid (B1) - Set when buffer one contains valid data.

1 Buffer 0 data valid (B0) - Set when buffer zero contains valid data.

0 Buffer select (BS) - The currently selected buffer.

86.8.7 IN Endpoint Control Register

*Table 1730.*0x000 - EICTRL - IN endpoint control register

31 21	20	19	18	17 7	6	5	4	3	2	1	0
BUFSZ	PI	CB	CS	MAXPL	NT		TT		EH	ED	EV
*	0	0	0	NR	NR		NR		0	0	0
r	rw	rw	rw	rw	rw		rw		rw	rw	rw

31: 21 Buffer size (BUFSZ) - Size/8 in bytes of one hardware buffer slot for this endpoint. Two slots are available for each endpoint.

20 Packet transmitted interrupt (PI) - Generate an interrupt each time a packet has been transmitted on the USB and the internal buffer is cleared. Reset value: '0'.

19 Clear buffers (CB) - Clears any buffers for the endpoint that contain data if the buffer is not currently active.

18 Control Stall (CS) - Return stall for data and status stages in a control transfer. Automatically cleared when the next setup token is received. Only used when the endpoint is configured as a control endpoint.

17: 7 Maximum payload (MAXPL) - Sets the maximum USB payload (maximum size of a single packet sent to/from the endpoint) size for the endpoint. All bits of the field are not always used. The maximum value for the maximum payload is determined with a generic for each endpoint. Not Reset.

6: 5 Number of transactions (NT) - Sets the number of additional transactions per microframe for high-speed endpoints and per frame for full-speed endpoints. Only valid for isochronous endpoints. Not Reset.

4: 3 Transfer type (TT) - Sets the transfer type for the endpoint. "00"=CTRL, "01" =ISOCH, "10"=BULK, "11"=INTERRUPT. Only OUT endpoints should be set to the CTRL type and then the IN endpoint with the same number will be automatically used. It is important not to use OUT endpoints that do not have a corresponding IN endpoint as a CTRL endpoint. Not Reset.

2 Endpoint halted (EH) - Halt the endpoint. If set, all transfers to this endpoint will receive a STALL handshake. Reset value: '0'.

1 Endpoint disabled (ED) - Disables the endpoint. If set, all transfers to this endpoint will receive a NAK handshake. Reset value: '0'.

0 Endpoint valid (EV) - Enables the endpoint. If not enabled, all transfers to this endpoint will be ignored and no handshake is sent. Reset value; '0'.

86.8.8 IN Slave Control Register

*Table 1731.*0x104 - EISLVCTRL - IN slave control register.

31 17	16 4	3	2	1	0
RESERVED	BUFCNT	PI	BA	BS	EB
0	0	0	0	0	0
r	r	rw	r	r	rw

31: 17 RESERVED

16: 4 Buffer counter (BUFCNT) - The number of bytes written in the current buffer.

3 Packet interrupt enable (PI) - Generate interrupt when the activated packet has been transmitted. Should be set together with EB when enabling a packet for transmission. Reset value: ‘0’.

2 Buffer active (BA) - A free buffer was acquired and is available for use.

1 Buffer select (BS) - Current buffer selected. Read only.

0 Enable (EB) - Enable current buffer for transmission if one has been acquired and try to acquire a new buffer. If no data has been written to the buffer a zero length packet will be transmitted.

86.8.9 IN Slave Buffer read/write Register

*Table 1732.*0x108 - EISLVDATA - IN slave buffer read/write register.

31 0
DATA
NR
r

31: 0 Data (DATA) - Data written to this register is placed into the current buffer for transmission. Byte, Halfword and word sizes are allowed but only a word aligned address should be used. This means that data is always placed on 31-0 for word, 31-16 for half-word and 31-24 for byte.

86.8.10 IN DMA Control Register

*Table 1733.*0x104 - EIDMACTRL - IN DMA control register.

31 11	10	9 4	3	2	1	0
RESERVED	AE	RESERVED	AD	AI	IE	DA
0	0	0	0	0	0	0
r	wc	r	rw	rw	rw	rw

31: 11 RESERVED

10 AHB error (AE) - An AHB has occurred for this endpoint.

9: 4 RESERVED

3 Abort DMA (AD) - Disable descriptor processing (set DA to 0) and abort the current DMA transfer if one is active. Reset value: ‘0’.

2 AHB error interrupt (AI) - Generate interrupt when an AHB error occurs for this endpoint.

1 Interrupt enable (IE) - Enable DMA interrupts. Each time data has been received or transmitted to/ from a descriptor with its interrupt enable bit set an interrupt will be generated when this bit is set.

0 Descriptors available (DA) - Set to indicate to the GRUSBDC that one or more descriptors have been enabled.

86.8.11 IN Descriptor Address Register

*Table 1734.*0x108 - EIDMADESC - IN descriptor address register.

31	2	1	0
DESCADDR		RES	

31: 2 Descriptor table address (DESCADDR) - Address to the next descriptor.Not Reset.

1: 0 RESERVED

86.8.12 IN Endpoint Status Register

*Table 1735.*0x10C - EISTAT - IN endpoint status register

31	30	29	28	16	15	3	2	1	0
RES		PT	B1CNT		B0CNT		B1	B0	BS
0		0	0		0		0	0	0
r		wc	r		r		r	r	r

31: 30 RESERVED.

29 Packet transmitted (PT) - Packet has been transmitted and cleared from the internal buffers. Cleared when written with a ‘1’.

28: 16 Buffer 1 byte count (B1CNT) - Number of bytes in buffer one.

15: 3 Buffer 0 byte count (B0CNT) - Number of bytes in buffer zero.

2 Buffer 1 data valid (B1) - Set when buffer one contains valid data.

1 Buffer 0 data valid (B0) - Set when buffer zero contains valid data.

0 Buffer select (BS) - The currently selected buffer.

86.8.13 CTRL Register

*Table 1736.*0x200 - GCTRL - ctrl register

31	30	29	28	27	26	16	15	14	13	12	11	9	8	7	1	0
SI	UI	VI	SP	FI	RESERVED		FT	EP	DH	RW	TS		TM	UA		SU
0	0	0	0	0	0		0	0	0	0	0		0	0		0
rw	rw	rw	rw	rw	r		rw	rw	rw	rw*	rw		rw*	rw*		w

31: Suspend interrupt (SI) - Generate interrupt when suspend status changes. Reset value: ‘0’.

30 USB reset (UI) - Generate interrupt when USB reset is detected. Reset value: ‘0’.

29 VBUS valid interrupt (VI) - Generate interrupt when VBUS status changes. Reset value: ‘0’.

28 Speed mode interrupt (SP) - Generate interrupt when Speed mode changes. Reset value: ‘0’.

27 Frame number received interrupt (FI) - Generate interrupt when a new Start of frame (SOF) token is received. Reset value: ‘0’.

26: 16 RESERVED

15 Functional test mode (FT) - Enables functional test-mode which shortens all timer such as reset and chirp timers to 8 clock cycles.

14 Enable pull-up (EP) - Enable pull-up on the D+ line signaling a connect to the host. Reset value: ‘0’.

13 Disable High-speed (DH) - Disable high-speed handshake to make the core full-speed only.

12 Remote wakeup (RW) - Start remote wakeup signaling. It is self clearing and will be cleared when it has finished transmitting remote wakeup if it was currently in suspend mode. If not in suspend mode when set it will self clear immediately. Writes to this bit when it is already asserted are ignored. Reset value: ‘0’.

11: 9 Testmode selector (TS) - Select which testmode to enter. “001”= Test_J, “010”= Test_K, “011”= Test_SE0_NAK, “100”= Test_Packet.

8 Enable test mode (TM) - Set to one to enable test mode. Note that the testmode cannot be left without resetting or power-cycling the core and cannot be entered if hsdis is set to ‘1’. Reset value: ‘0’.

Table 1736. 0x200 - GCTRL - ctrl register

7: 1	USB address (UA) - The address assigned to the device on the USB bus.
0	Set USB address (SU) - Write with a one to set the usb address stored in the USB address field.

86.8.14 Status Register

Table 1737. 0x204 - GSTAT - status register

31 28	27 24	23	22 18	17	16	15	14	13 11	10 0
NEPI	NEPO	DM	RESERVED	SU	UR	VB	SP	AF	FN
*	*	*	0	0	0	0	0	0	0
r	r	r	r	r	wc	r	r	r	r

31: 28	Number of implemented IN endpoints (NEPI) - The number of configurable IN endpoints available in the core (including endpoint 0) minus one.
27: 24	Number of implemented OUT endpoints (NEPO) - The number of configurable OUT endpoints available in the core (including endpoint 0) minus one.
23	Data mode (DM) - 0 = core uses slave mode for data transfers, 1 = core uses master mode (DMA) for data transfers.
22: 18	RESERVED
17	Suspended (SU) - Set to ‘0’ when the device is suspended and ‘1’ when not suspended.
16	USB reset (UR) - Set each time an USB reset has been detected. Cleared when written with a ‘1’.
15	Vbus valid (VB) - Set to one when a valid voltage has been detected on the USB vbus line.
14	Speed (SP) - The current speed mode of the USB bus. ‘0’ = high-speed, ‘1’ = full-speed.
13: 11	Additional frames (AF) - Number of additional frames received with the current frame number.
10: 0	Frame number (FN) - The value of the last SOF token received.

86.9 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x021. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

86.10 Implementation

86.10.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User’s Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. A subset of the registers in the USB clock domain make use of asynchronous reset.

The IP core has two different reset inputs: the AMBA reset and the USB reset. They work independently on their respective clock domains. If reset generation/synchronization is desired, it shall be done in a higher instance, out of the IP.

See also documentation of *prst* VHDL generic.

86.10.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User’s Manual). The core is designed for big-endian systems.

86.11 Configuration options

Table 1738 shows the configuration options of the core (VHDL generics). Buffer sizes are given in bytes and determines the sizes of one hardware buffer for the endpoint. Two buffers are available for each endpoint. The buffer size must be equal to or larger than the desired maximum payload size. In the case of high-bandwidth endpoints the buffer size must be equal to or larger than the maximum payload times the number of transactions per (micro) frame.

*Table 1738.*Configuration options

Generic	Function	Allowed range	Default
hsindex	AHB slave index.	0 - NAHBSLV-1	0
hirq	AHB interrupt number	0 - NAHBIRQ-1	0
haddr	AHB slave address	-	0
hmask	AHB slave address mask	-	16#FFF#
hmindex	AHB master index	0 - NAHBMST-1	
aiface	0 selects the AHB slave interface for data transfer while 1 selects the AHB master interface.	0 - 1	0
memtech	Memory technology used for blockrams (endpoint buffers).	0 - NTECH	0
uiface	0 selects the UTMI interface while 1 selects ULPI.	0 - 1	0
dwidth	Selects the data path width for UTMI.	8 - 16	8
blen	Maximum number of beats in burst accesses on the AHB bus.	4 - 128	16
nepi	Number of IN endpoints	1 - 16	1
nepo	Number of OUT endpoints	1 - 16	1
i0	Buffer size for IN endpoint 0.	8, 16, 24, ... , 3072	1024
i1	Buffer size for IN endpoint 1.	8, 16, 24, ... , 3072	1024
i2	Buffer size for IN endpoint 2.	8, 16, 24, ... , 3072	1024
i3	Buffer size for IN endpoint 3.	8, 16, 24, ... , 3072	1024
i4	Buffer size for IN endpoint 4.	8, 16, 24, ... , 3072	1024
i5	Buffer size for IN endpoint 5.	8, 16, 24, ... , 3072	1024
i6	Buffer size for IN endpoint 6.	8, 16, 24, ... , 3072	1024
i7	Buffer size for IN endpoint 7.	8, 16, 24, ... , 3072	1024
i8	Buffer size for IN endpoint 8.	8, 16, 24, ... , 3072	1024
i9	Buffer size for IN endpoint 9.	8, 16, 24, ... , 3072	1024
i10	Buffer size for IN endpoint 10.	8, 16, 24, ... , 3072	1024
i11	Buffer size for IN endpoint 11.	8, 16, 24, ... , 3072	1024
i12	Buffer size for IN endpoint 12.	8, 16, 24, ... , 3072	1024
i13	Buffer size for IN endpoint 13.	8, 16, 24, ... , 3072	1024
i14	Buffer size for IN endpoint 14.	8, 16, 24, ... , 3072	1024
i15	Buffer size for IN endpoint 15.	8, 16, 24, ... , 3072	1024
o0	Buffer size for OUT endpoint 0.	8, 16, 24, ... , 3072	1024
o1	Buffer size for OUT endpoint 1.	8, 16, 24, ... , 3072	1024
o2	Buffer size for OUT endpoint 2.	8, 16, 24, ... , 3072	1024
o3	Buffer size for OUT endpoint 3.	8, 16, 24, ... , 3072	1024
o4	Buffer size for OUT endpoint 4.	8, 16, 24, ... , 3072	1024
o5	Buffer size for OUT endpoint 5.	8, 16, 24, ... , 3072	1024
o6	Buffer size for OUT endpoint 6.	8, 16, 24, ... , 3072	1024
o7	Buffer size for OUT endpoint 7.	8, 16, 24, ... , 3072	1024

*Table 1738.*Configuration options

Generic	Function	Allowed range	Default
o8	Buffer size for OUT endpoint 8.	8, 16, 24, ... , 3072	1024
o9	Buffer size for OUT endpoint 9.	8, 16, 24, ... , 3072	1024
o10	Buffer size for OUT endpoint 10.	8, 16, 24, ... , 3072	1024
o11	Buffer size for OUT endpoint 11.	8, 16, 24, ... , 3072	1024
o12	Buffer size for OUT endpoint 12.	8, 16, 24, ... , 3072	1024
o13	Buffer size for OUT endpoint 13.	8, 16, 24, ... , 3072	1024
o14	Buffer size for OUT endpoint 14.	8, 16, 24, ... , 3072	1024
o15	Buffer size for OUT endpoint 15.	8, 16, 24, ... , 3072	1024
oepol	Select polarity of output enable signal. 1 selects active high and 0 selects active low.	0 - 1	0
keepclk	This generic determines wheter or not the USB transceiver will be suspended and have its clock turned off during USB suspend. Set this generic to 1 if the clock should not be turned off. This might be needed for some technologies that can't handle that the USB clock is turned off for long periods of time.	0 - 1	0
sepirq*	Set this generic to 1 if three separate interrupt lines should be used, one for status related interrupts, one for IN endpoint related interrupts, and one for OUT endpoint related interrupts. The irq number for the three different interrupts are set with the hirq (status), irqi (IN), and irqo (OUT) generics. If sepirq = 0 then only interrupts with irq number hirq will be generated.	0 - 1	0
irqi*	Sets the irq number for IN endpoint related interrupts. Only used if sepirq generic is set to 1.	0 - NAHBIRQ-1	1
irqo*	Sets the irq number for OUT endpoint related interrupts. Only used if sepirq generic is set to 1.	0 - NAHBIRQ-1	2
functesten	Enable functional test mode. This is used to skip the USB high-speed detection sequence to reduce the number of test vectors during functional testing.	0 - 1	0
scantest	Set this generic to 1 if scan test support should be implemented.	0 - 1	0
nsync	Number of stages for the internal clock-domain-crossing synchronization registers.	1 - 2	1

* The values of these generics are stored in the first User-Defined word of the core's AHB plug-n-play area as follows: bit 0 = sepirq, bits 7:4 = irqi, bits 11:8 = irqo. Please see the AHBCTRL section of GRLIB IP Core User's Manual.

86.12 Signal descriptions

Table 1739 shows the interface signals of the core (VHDL ports).

*Table 1739.*Signal descriptions

Signal name	Field	Type	Function	Active
UCLK	N/A	Input	USB UTMI/ULPI Clock	-
URST		Input	USB Reset	Low
HCLK		Input	AMBA Clock	-
HRST		Input	AMBA Reset	Low
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
AHBSI	*	Input	AHB slave input signals	-

Table 1739. Signal descriptions

Signal name	Field	Type	Function	Active
AHBSO	*	Output	AHB slave output signals	-
USBI	datain[15:0]	Input	UTMI/UTMI+/ULPI. Bits 15:8 are only used in 16-bit UTMI/UTMI+ mode.	-
	rxactive	Input	UTMI/UTMI+	High
	rxvalid	Input	UTMI/UTMI+	High
	rxvalidh	Input	UTMI/UTMI+ 16-bit	High
	rxerror	Input	UTMI/UTMI+	High
	txready	Input	UTMI/UTMI+	High
	linestate[1:0]	Input	UTMI/UTMI+	-
	nxt	Input	ULPI	High
	dir	Input	ULPI	High
	vbusvalid	Input	UTMI+	High
	urstdrive	Input	This input determines if the cores should drive the transceiver data lines low during USB transceiver reset, even if the dir input is High. This is needed for some transceivers, such as the NXP ISP1504. When this input is low the direction of the transceiver data lines are exclusively controlled by the dir signal from the transceiver. When this input is high the core will drive the data lines low during transceiver reset. Only applicable for ULPI transceivers.	High
USBO	dataout[15:0]	Output	UTMI/UTMI+/ULPI. Bits 15:8 are only used in 16-bit UTMI/UTMI+ mode.	-
	txvalid	Output	UTMI+	High
	txvalidh	Output	UTMI+ 16-bit	High
	opmode[1:0]	Output	UTMI+	-
	xcvrselect[1:0]	Output	UTMI/UTMI+. Bit 1 is constant low.	-
	termselect	Output	UTMI/UTMI+	-
	suspendm	Output	UTMI/UTMI+	Low
	reset	Output	Transceiver reset signal. Asserted asynchronously and deasserted synchronously to the USB clock.	**
	stp	Output	ULPI	High
	oen	Output	Data bus direction control for ULPI and bi-directional UTMI/UTMI+ interfaces.	***
	databus16_8	Output	UTMI+. Constant high for 16-bit interface, constant low for 8-bit interface.	-
	dppulldown	Output	UTMI+. Constant low.	High
	dmpulldown	Output	UTMI+. Constant low.	High
	idpullup	Output	UTMI+. Constant low.	High
	drvvbus	Output	UTMI+. Constant low.	High
	dischrgvbus	Output	UTMI+. Constant low.	High
	chrgvbus	Output	UTMI+. Constant low.	High
	txbitstuffenable	Output	UTMI+. Constant low.	High
	txbitstuffenableh	Output	UTMI+. Constant low.	High
	fslsserialmode	Output	UTMI+. Constant low.	High
	tx_enable_n	Output	UTMI+. Constant high.	Low

*Table 1739.*Signal descriptions

Signal name	Field	Type	Function	Active
	tx_dat	Output	UTMI+. Constant low.	High
	tx_se0	Output	UTMI+. Constant low.	High

* See GRLIB IP Library User's Manual.

** Depends on transceiver interface. Active high for UTMI/UTMI+ and active low for ULPI.

*** Implementation dependent.

86.13 Library dependencies

Table 1740 shows libraries used when instantiating the core (VHDL libraries).

*Table 1740.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	GRUSB	Signals, component	GRUSBDC component declarations, USB signals

86.14 Instantiation

This example shows how the core can be instantiated.

```
usbdc0: GRUSBDC
    generic map(
      hsindex       => 4,
      hirq          => 0,
      haddr         => 16#001#,
      hmask         => 16#FFF#,
      hmindex       => 14,
      aiface        => 1,
      memtech       => memtech,
      uiface        => 0,
      dwidth        => 8,
      nepi          => 16,
      nepo          => 16)
    port map(
      uclk          => uclk,
      urst          => urstn,
      usbi          => usbi,
      usbo          => usbo,
      hclk          => clkm,
      hrst          => rstn,
      ahbmi         => ahbmi,
      ahbmo         => ahbmo(14),
      ahbsi         => ahbsi,
      ahbso         => ahbso(4)
     );

usb_d_pads: for i in 0 to 15 generate
  usb_d_pad: iopad generic map(tech => padtech, slew => 1)
    port map (usb_d(i), usbo.dataout(i), usbo.oen, usbi.datain(i));
end generate;

usb_h_pad:iopad generic map(tech => padtech, slew => 1)
  port map (usb_validh, usbo.txvalidh, usbo.oen, usbi.rxvalidh);

usb_i0_pad : inpad generic map (tech => padtech) port map (usb_txready,usbi.txready);
usb_i1_pad : inpad generic map (tech => padtech) port map (usb_rxvalid,usbi.rxvalid);
usb_i2_pad : inpad generic map (tech => padtech) port map (usb_rxerror,usbi.rxerror);
usb_i3_pad : inpad generic map (tech => padtech) port map (usb_rxactive,usbi.rxactive);
usb_i4_pad : inpad generic map (tech => padtech) port map
(usb_linestate(0),usbi.linestate(0));
```

```
usb_i5_pad : inpad generic map (tech => padtech) port map
(usb_linestate(1),usbi.linestate(1));
usb_i6_pad : inpad generic map (tech => padtech) port map (usb_vbus, usbi.vbusvalid);

usb_o0_pad : outpad generic map (tech => padtech, slew => 1) port map (usb_reset,usbo.reset);
usb_o1_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_suspend,usbo.suspendm);
usb_o2_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_termsel,usbo.termselect);
usb_o3_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_xcvrsel,usbo.xcvrselect(0));
usb_o4_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_opmode(0),usbo.opmode(0));
usb_o5_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_opmode(1),usbo.opmode(1));
usb_o6_pad : outpad generic map (tech => padtech, slew => 1) port map
(usb_txvalid,usbo.txvalid);

usb_clk_pad : clkpad generic map (tech => padtech, arch => 2) port map (usb_clkout, uclk);

usbi.urstdrive <= '0';
```

87 GRUSB_DCL - USB Debug Communication Link

87.1 Overview

The Universal Serial Bus Debug Communication Link (GRUSB_DCL) provides an interface between a USB 2.0 bus and an AMBA-AHB bus. The core must be connected to the USB through an UTMI, UTMI+, or ULPI compliant PHY. Both full-speed and high-speed mode are supported. The GRUSB_DCL rely on the GRUSBDC core for handling the USB communication and communication with the PHY. The GRUSB_DCL implements the minimum required set of USB requests to be Version 2.0 compliant and a simple protocol for performing read and write accesses on the AHB bus. Figure 236 show how the GRUSB_DCL can be connected to a PHY. For more information on the GRUSBDC and the connection to the USB PHY please refer to the GRLIB IP Core User's Manual.

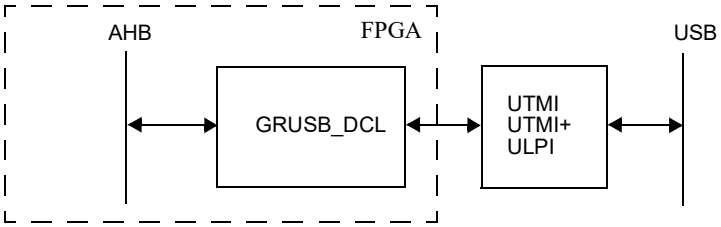


Figure 236. USBDCL connected to an external UTM.

87.2 Operation

87.2.1 System overview

The internal structure of the GRUSB_DCL can be seen in figure 237. The GRUSB_DCL is constructed with two internal AHB busses for communication with the GRUSBDC and one external AHB master interface for reading and writing the external AHB bus. Since the GRUSBDC is connected with point-to-point links there is no need for a conventional AHB arbiter on the internal bus.

The GRUSBDC is configured with two bidirectional endpoints with endpoint zero (EP0) being the default USB control endpoint and endpoint one (EP1) the communication endpoint for the DCL protocol. The GRUSBDC is configured to use DMA and its descriptors as well as the DMA buffers are stored in a local memory with separate read and write ports (SYNCRAM_2P). The two ports makes it possible for the GRUSBDC and the internal workings of the GRUSB_DCL to access the memory in parallel. Arbitration for the read and write port is implemented as two separate procedures. The main functionality of the GRUSB_DCL is implemented in the main FSM procedure. The FSM can be seen in figure 238.

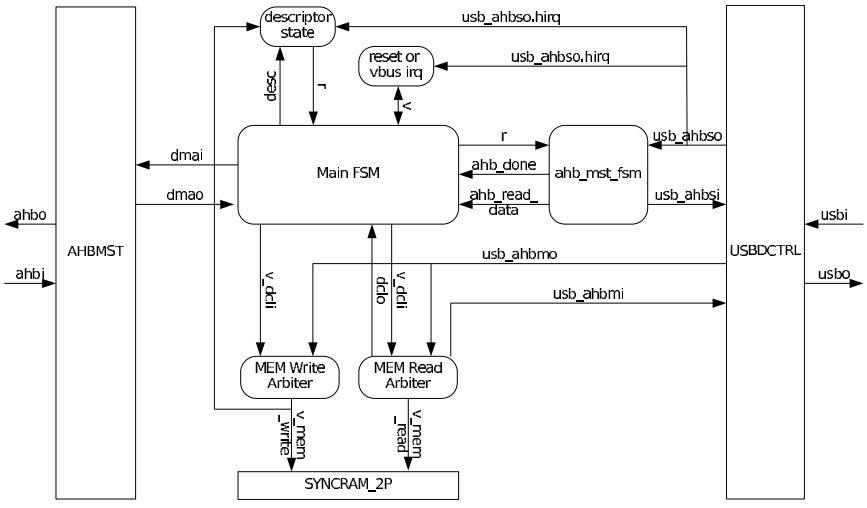


Figure 237. Block diagram of the internal structure of the GRUSBDCL. Blocks with rounded corners are implemented as VHDL procdures while squares represent VHDL entities.

Upon reset the FSM begins with setting up the DMA descriptors in the local memory and then configures the GRUSBDC such that it becomes active. The FSM then waits for incoming requests on either EP0 or EP1. For EP0 each request is validated and then appropriate action is taken according to the USB Version 2.0 standard. For undefined requests the GRUSB_DCL returns an error by stalling EP0. For EP1 the DCL request is fetched and either data is written to the AHB buss from the local memory or data is read from the AHB and stored in the local memory. In the case of the AHB is being read the data is then sent on EP1 IN. To keep track of the state of the DMA descriptors the FSM has help from the descriptor-state procedure. The descriptor-state procedure uses the IRQ from the GRUSBDC to identify incoming packets on either EP0 or EP1. By storing the last accessed address by the GRUSBDC to the local memory the descriptor-state procedure can tell which EP that has been updated. The FSM in turn signals the procedure telling it when a DMA descriptor/buffer has been read/write. A small FSM (ahb_mst_fsm) is also used when the main FSM wants to update the state of the GRUSBDC through the AHB slave interface. Finally, the reset-or-vbus-irq procedure listens on irqs from the USBDC that informs the core that a USB reset or that a change on the VBUS has occurred. In that case the core stops what ever it was doing and moves into the USB default state (no address set and not configured).

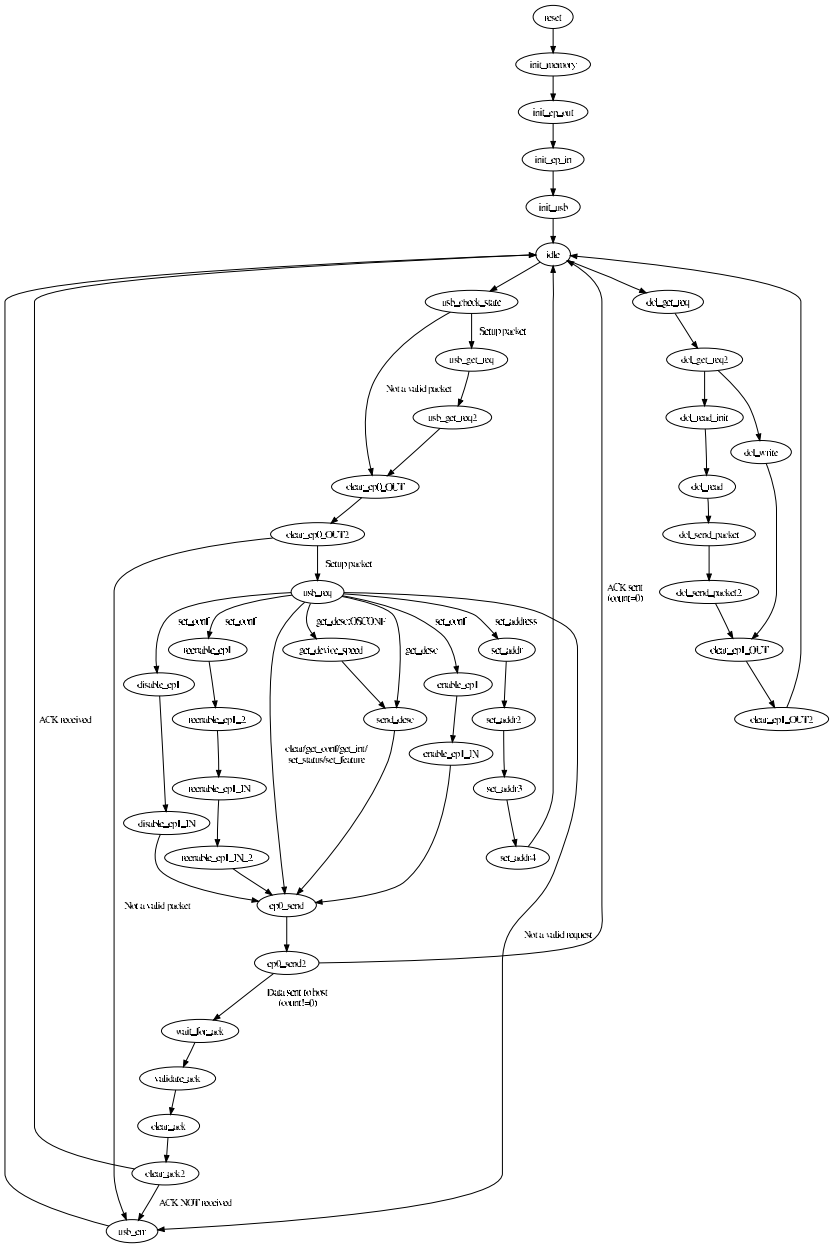


Figure 238. The main FSM of the GRUSBDCL. The FSM can be divided into three major parts i) the initialization (reset to idle), ii) handling of USB requests (seen to the left of idle), and iii) handling of DCL requests (seen to the right of idle).

87.2.2 Protocol

The protocol used for the AHB commands is very simple and consists of two 32-bit control words. The first word consists of the 32-bit AHB address and the second consists of a read/write bit at bit 31 and the number of words to be written at bits 16 downto 2. All other bits in the second word are reserved for future use and must be set to 0. The read/write bit must be set to 1 for writes.

Figure 239 shows the layout of a write command. The command should be sent as the data cargo of an OUT transaction to endpoint 1. The data for a command must be included in the same packet. The maximum payload is 512 B when running in high-speed mode and 64 B in full-speed mode. Since the control information takes 8 B the maximum number of bytes per command is 504 B and 56 B respectively. Subword writes are not supported so the number of bytes must be a multiple of four between 0 and 504.

The words should be sent with the one to be written at the start address first. Individual bytes should be transmitted msb first, i.e. the one at bits 31-24.

There is no reply sent for writes since the USB handshake mechanism for bulk writes guarantees that the packet has been correctly received by the target.

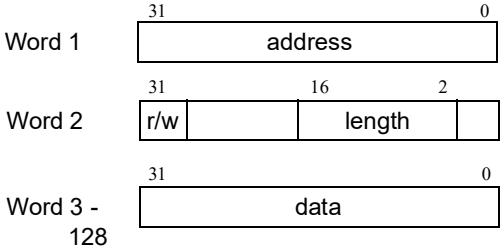


Figure 239. Layout of USBDCL write commands.

Figure 240 shows the layout of read commands and replies. In this case the command only consists of two words containing the same control information as the two first words for write commands. However, for reads the r/w bit must be set to 0.

When the read is performed data is read to the buffer belonging to IN endpoint 1. The reply packet is sent when the next IN token arrives after all data has been stored to the buffer. The reply packets only contains the read data (no control information is needed) with the word read from the start address transmitted first. Individual bytes are sent with most significant byte first, i.e. the byte at bit 31 downto 24.

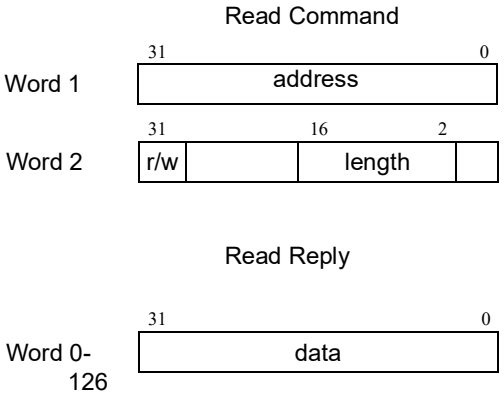


Figure 240. Layout of USBDCL read commands and replies.

87.2.3 AHB operations

All AHB operations are performed as incremental bursts of unspecified length. Only word size accesses are done.

87.3 Registers

The core does not contain any user accessible registers.

87.4 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x022. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

The USB vendor identifier is 0x1781 and product identifier is 0x0AA0.

87.5 Implementation

87.5.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. A subset of the registers in the USB clock domain make use of asynchronous reset.

The IP core has two different reset inputs: the AMBA reset and the USB reset. They work independently on their respective clock domains. If reset generation/synchronization is desired, it shall be done in a higher instance, out of the IP.

See also documentation of *syncprst* VHDL generic.

87.5.2 Scan test support

The VHDL generic *scantest* enables scan test support for both the GRUSB_DCL and GRUSBDC. When the scanen and testen signals in the AHB master input record are high the GRUSB_DCL will disable the internal RAM blocks.

See GRUSBDC section of GRLIB IP Core User's Manual for details on the scan support for GRUSBDC.

87.5.3 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

87.6 Configuration options

Table 1741 shows the configuration options of the core (VHDL generics).

*Table 1741.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
memtech	Memory technology used for blockrams (endpoint buffers).	0 - NTECH	0
uiface	Please see GRUSBDC section in the GRLIB IP Core User's Manual.		
dwidth	Please see GRUSBDC section in the GRLIB IP Core User's Manual.		
oepol	Please see GRUSBDC section in the GRLIB IP Core User's Manual.		
keepclk	Please see GRUSBDC section in the GRLIB IP Core User's Manual.		
functesten	Please see GRUSBDC section in the GRLIB IP Core User's Manual. If this generic is non-zero, the core will sample the value of its functesten input signal during reset. This value will then be used when assigning the Functional Testmode field in the GRUSBDC control register. The functesten input can be useful during netlist simulation as functional test mode reduces simulation time. If this generic is set to zero, the value of the functesten input will be disregarded and the Functional Testmode field will always be written with '0'.		
burstlength	Sets the maximum burst length in 32-bit words. The core will not burst over a burstlength word boundary.	8	1 - 512
scantest	Set this generic to 1 if scan test support should be implemented.	0 - 1	0

87.7 Signal descriptions

Table 1742 shows the interface signals of the core (VHDL ports).

*Table 1742.*Signal descriptions

Signal name	Field	Type	Function	Active
UCLK	N/A	Input	USB UTMI/ULPI Clock	-
URST	N/A	Input	USB Reset	Low
USBI	*	Input	USB Input signals	-
	functesten	Input	Functional test enable. If the core has been implemented with support for functional test mode (VHDL generic *functesten*), this signal will be sampled during core reset. Its value will then be used to set the functional testmode enable bit in the GRUSBDC core's control register.	High
USBO	*	Output	USB Output signals	-
HCLK		Input	AMBA Clock	-
HRST		Input	AMBA Reset	Low
AHBMI	**	Input	AHB master input signals	-
AHBMO	**	Output	AHB master output signals	-

* see GRUSBDC section og GRLIB IP Core User's Manual

** see GRLIB IP Library User's Manual

87.8 Library dependencies

Table 1743 shows libraries used when instantiating the core (VHDL libraries).

*Table 1743.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	GRUSB	Signals, components	GRUSB_DCL and GRUSBDC component declarations, USB signals

87.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.grusb.all;

entity usbdcl_ex is
  port (
    clk : in std_ulogic; --AHB Clock
    rstn : in std_ulogic;

    -- usb signals
    usb_clkout  : in std_ulogic;
    usb_rst     : in std_ulogic;
    usb_d       : inout std_logic_vector(7 downto 0);
    usb_nxt     : in  std_ulogic;
    usb_stp     : out std_ulogic;
    usb_dir     : in  std_ulogic;
    usb_resetn  : out std_ulogic
end;

architecture rtl of usbdcl_ex is
  constant padtech : integer := inferred;
  constant memtech : integer := inferred;

  -- AMBA signals
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
  -- USB signals
  signal usbi : grusb_in_type;
  signal usbo : grusb_out_type;
  signal uclk : std_ulogic;
  signal urstn : std_ulogic;

begin

  -- AMBA Components are instantiated here
  ...

  -- GRUSB_DCL
  usb_d_pad: iopadv
    generic map(tech => padtech, width => 8)
    port map (usb_d, usbo.dataout, usbo.oen, usbi.datain);
  usb_nxt_pad : inpad generic map (tech => padtech)
    port map (usb_nxt, usbi.nxt);
  usb_dir_pad : inpad generic map (tech => padtech)
    port map (usb_dir, usbi.dir);
  usb_resetn_pad : outpad generic map (tech => padtech)
    port map (usb_resetn, usbo.reset);
  usb_stp_pad : outpad generic map (tech => padtech)
```

```
      port map (usb_stp, usbo.stp);

    usb_clkout_pad : clkpad
      generic map (tech => padtech)
      port map (usb_clkout, uclk);

    usb_input_rst_pad : inpad
      generic map (tech => padtech)
      port map (usb_rst, urstn);

    usbi.urstdrive <= '0';

    usbdcl0: grusb_dcl
      generic map (
          hindex  => 0,
          memtech => memtech,
          uiface  => 1,
          dwidth  => 8,
          oepol   => 0)
      port map (
          uclk => uclk,
          urst => urstn,
          usbi => usbi,
          usbo => usbo,
          hclk => clk,
          hrst => rstn,
          ahbi => ahbmi,
          ahbo => ahbmo(0));
end;
```

88 GRUSBHC - USB 2.0 Host Controller

88.1 Overview

The Frontgrade Gaisler USB 2.0 Host Controller provides a link between the AMBA AHB bus and the Universal Serial Bus. The host controller supports High-, Full-, and Low-Speed USB traffic. USB 2.0 High-Speed functionality is supplied by an enhanced host controller implementing the Enhanced Host Controller Interface revision 1.0. Full- and Low-Speed traffic is handled by up to 15 (USB 1.1) companion controllers implementing the Universal Host Controller Interface, revision 1.1. Each controller has its own AMBA AHB master interface. Configuration and control of the enhanced host controller is done via the AMBA APB bus. Companion controller registers are accessed via an AMBA AHB slave interface. Figure 241 shows a USB 2.0 host system and the organization of the controller types. Figure 242 shows an example with both host controller types present.

The controller supports both UTMI+ and ULPI transceivers and can handle up to 15 ports.

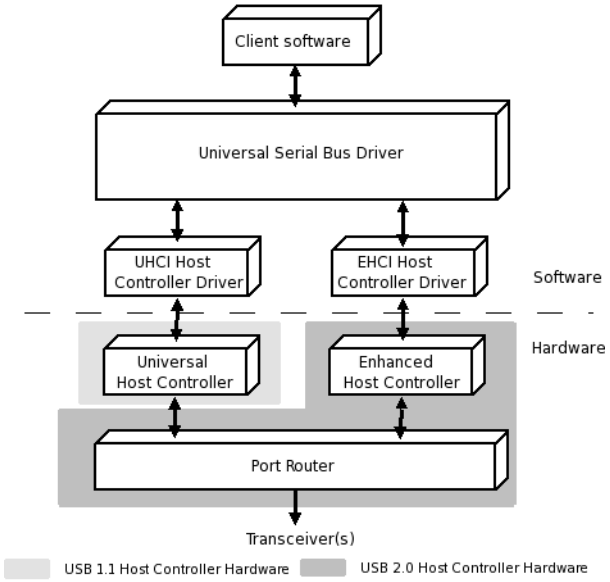


Figure 241. Block diagram of USB 2.0 host system

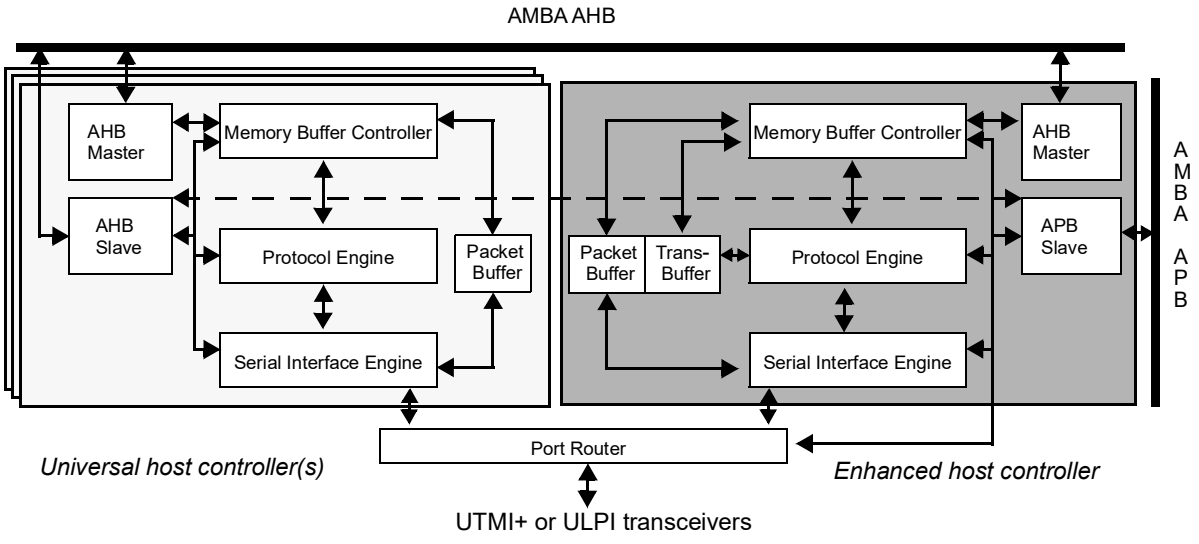


Figure 242. Block diagram of both host controller types

88.2 Operation

88.2.1 System overview

Depending on the core's configuration it may contain both controller types, one enhanced host controller, or up to 15 standalone universal host controllers. If both controller types are present, each universal host controller acts as a companion controller to the enhanced host controller.

The enhanced host controller complies with the Enhanced Host Controller Interface with the exception of the optional Light Host Controller Reset, which is not implemented.

The universal host controller complies with the Universal Host Controller Interface, with exceptions. The HCHalted field in the USB Command register is implemented as Read Only instead of Read/Write Clear. The Port Status/Control registers have been extended with Over Current and Over Current Change fields. Changes to both registers have been done in accordance with contemporary implementations of the interface. Both changes match the description of corresponding bits in the EHCI specification.

88.2.2 Protocol support

The enhanced host controller has full support for High-Speed traffic as defined in the USB Specification, revision 2.0. In addition Asynchronous Park Mode is supported, and the controller has a NAK counter.

The universal host controller supports Full- and Low-Speed traffic.

88.2.3 Descriptor and data buffering

The enhanced host controller prefetches one frame of isochronous descriptors. All payload data for a transaction is fetched before the transaction is executed. The enhanced host controller has a 2048 byte buffer for descriptors and a 2048 byte buffer for payload data, which can hold data for two transactions.

The universal host controller does not prefetch descriptors. Depending on controller configuration a transaction on the bus may be initiated before all payload data has been fetched from memory. Each universal host controller has a 1024 byte buffer for payload data. A transfer descriptor in UHCI may describe a transaction that has a payload of 1280 bytes. The USB specification limits the maximum allowed data payload to 1023 bytes and the controller will not transfer a larger payload than 1023 bytes. If a descriptor has a, legal, larger payload than 1023 bytes, the controller will only attempt to transfer the first 1023 bytes before the transaction is marked as completed.

In the event that the host controller has just one port, the universal host controller and the enhanced host controller will share the data payload buffer. Thus only two 2048 byte buffers are required.

88.2.4 Endianness

The core always accesses the least significant byte of a data payload at offset zero. Depending on the core's configuration, registers may be big endian, little endian, or byte swapped little endian.

88.2.5 RAM test facilities

The VHDL generic *ramtest* adds the possibility to test the RAM by mapping the core's internal buffers into the register space. If the core is implemented with RAM test facilities the universal host controller maps the packet buffer at offset 0x400 - 0x7FF. An enhanced host controller will map the packet buffer at offset 0x1000 - 0x17FF and the transaction buffer at 0x1800 - 0x1FFF. Note that the VHDL generics *uhchmask* and *ehcpmask* must be modified to allow access to the increased number of registers. The three least significant bits of the universal host controller's mask must be set to zero. The enhanced host controller's mask must have its five least significant bits set to zero.

When the *ramtest* generic is set to one an extra register called *RAM test control* register is added to both the universal and the enhanced controller. This register is described in section 88.8.1 and 88.8.2. To perform the RAM tests the user should first make sure that both the universal host controller and enhanced host controller are in their respective idle state. Note that if the core has only one port then the enhanced controller and the universal controller share the packet buffer. The shared buffer can be tested through both of the controllers but the controller performing the test must be the current owner of the port. For information on how to enter idle state and change ownership of the port please see section 88.8. When the controllers are in their idle states the enable bit in the *RAM test control* register should be set to one.

Once RAM test is enabled the whole RAM can be tested by first filling the buffers by writing to the corresponding register addresses and then setting the start bit in the *RAM test control* register. When the start bit is set the controller will, in order to access to the RAM from all its ports, read and write the whole packet buffer from the USB domain. If the core uses dual port RAM (see section 88.10.2 for more information on RAM usage) the same thing is done with the transaction buffer (if it is the enhanced controller that is performing the test). When the core uses double port RAM both the read and write port of the transaction buffer is located in the AHB domain, and therefore both the read and write port is tested during read and write accesses to register space. When the transfers are finished the core will clear the start bit and the data can then be read back through register space and be compared with the values that were written. When the core is implemented with dual port RAM individual addresses in the packet buffer and transaction buffer can be written and read without using the start bit. When using double port RAM only the transaction buffer can be read without using the start bit. However when individual addresses are accessed the buffers are only read/written from the AHB domain.

88.3 Port routing

Port routing is implemented according to the EHCI specification but functions regardless of whether the core is configured with or without an enhanced host controller. The VHDL generic prr enables or disables Port Routing Rules. With Port Routing Rules enabled, each port can be individually routed to a specific universal host controller via the VHDL generics portroute1 and portroute2. If Port Routing Rules is disabled the n_pcc lowest ports are routed to the first companion controller, the next n_pcc ports to the second companion controller, and so forth. The HCSP-PORTROUTE array is communicated via the portroute VHDL generics, which are calculated with the following algorithm:

$$\text{portroute1} = 2^{26}*CC_8 + 2^{22}*CC_7 + 2^{18}*CC_6 + 2^{14}*CC_5 + 2^{10}*CC_4 + 2^{6}*CC_3 + 2^{2}*CC_2 + CC_1 / 4$$

$$\text{portroute1} = 2^{26}*CC_{15} + 2^{22}*CC_{14} + 2^{18}*CC_{13} + 2^{14}*CC_{12} + 2^{10}*CC_{11} + 2^{6}*CC_{10} + 2^{2}*CC_9 + CC_1 \bmod 4$$

where CC_P is the companion controller that port P is routed to. Companion controllers are enumerated starting at 1.

When the enhanced host controller has not been configured by software, or when it is nonexistent, each port is routed to its companion controller. This allows a universal host controller to function even if the host system does not have support for the enhanced host controller. Please see the EHCI specification for a complete description of port routing.

88.4 DMA operations

Both host controller types have configurable DMA burst lengths. The burst length in words is defined by the VHDL generic bwrd. The value of bwrd limits how many words a controller may access in memory during a burst and not the number of memory operations performed after bus access has been granted. When writing a data payload back to memory that requires half-word or byte addressing the number of memory operations may exceed bwrd by one before the bus is released. If a host controller

is given a byte-aligned data buffer its burst length may exceed the bwrd limit with one word when fetching payload data from memory.

The universal host controller uses a burst length of four words when fetching descriptors. This descriptor burst length is not affected by the bwrd VHDL generic. The universal host controller may be configured to start transactions on the USB before all data has been fetched from memory. The VHDL generic uhcblo specifies the number of words that must have been fetched from memory before a USB transaction is started. Since the USB traffic handled by the universal host controller can be expected to have significantly lower bandwidth than the system memory bus, this generic should be set to a low value.

88.5 Endianness

The core works internally with little endian. If the core is connected to a big endian bus, endian conversion must be enabled. When the VHDL generic endian_conv is set, all AMBA data lines are byte swapped. With endian_conv correctly set the core will start accessing data payloads from byte offset zero in the buffer, this is the first byte that is moved on the USB. The VHDL generic endian_conv must be set correctly for byte and halfword accesses to work. Therefore it is not possible to change the byte order of the buffer by configuring the controller for a little endian bus when it is connected to a big endian bus or vice versa.

The VHDL generics be_regs and be_desc are used to place the controller into big endian mode when endian conversion is enabled. These configuration options have no effect when the core is connected to a little endian bus, as determined by the value of VHDL generic endian_conv. The VHDL generic be_regs arranges the core's registers in accordance with big endian addressing. In the enhanced host controller this will only affect the placement of the register fields CAPLENGTH and HCIVERSION, the HCSP-PORTROUTE array, and - if implemented - the PCI registers *Serial Bus Release Number Register* and *Frame Length Adjustment Register*. In the universal host controller be_regs will affect the placement of all registers. When be_regs is set, the bus to the register interface is never byte swapped. Tables 1744 - 1746 below illustrate the difference between big endian, little endian, and little endian layout with byte swapped (32 bit) WORDs on two 16 bit registers. Register R1 is located at address 0x00 and register R2 is located at address 0x02.

*Table 1744.*R1 and R2 with big endian addressing

31	16	15	0
R1(15:0)		R2(15:0)	

*Table 1745.*R1 and R2 with little endian addressing

31	16	15	0
R2(15:0)		R1(15:0)	

*Table 1746.*R1 and R2 with little endian layout and byte swapped DWORD

31	24	23	16	15	8	7	0
R1(7:0)		R1(15:8)		R2(7:0)		R2(15:8)	

The VHDL generic be_desc removes the byte swapping of descriptors on big endian systems. Tables 1747 and 1748 below list the effects of endian_conv and be_regs on a big endian and a little endian system respectively.

*Table 1747.*Effect of endian_conv, be_regs, and be_desc on a big endian system

endian_conv	be_regs	be_desc	System configuration
0	-	-	Illegal. DMA will not function.
1	0	0	Host controller registers will be arranged according to little endian addressing and each DWORD will be byte swapped. In-memory transfer descriptors will also be byte swapped. This is the correct configuration for operating systems, such as Linux, that swap the bytes on big endian systems.
1	0	1	Host controller registers are arranged according to little endian addressing and will be byte swapped. Transfer descriptors will not be byte swapped.
1	1	0	Host controller registers will be arranged according to big endian addressing and will not be byte swapped. In memory transfer descriptors will be byte swapped.
1	1	1	Host controller registers will be arranged according to big endian addressing. In memory transfer descriptors will not be byte swapped.

*Table 1748.*Effect of endian_conv, be_regs and be_desc on a little endian system

endian_conv	be_regs	be_desc	System configuration
0	-	-	Host controller registers will be placed as specified in the register interface specifications.
1	-	-	Illegal. DMA will not function.

88.6 Transceiver support

The controller supports UTMI+ 8-bit, UTMI+ 16-bit, and ULPI transceivers. All connected transceivers must be of the same type. Note that the transceiver type is fixed and the core can therefore not change between 8-bit and 16-bit UTMI+ interface during operation. Transceiver signals not belonging to the selected transceiver type are not connected and do not need to be driven. When using ULPI transceivers the default, and recommended, configuration is to use an external source for USB bus power (VBUS) as well as external VBUS fault detection. However the core can be configured to support configurations where the ULPI transceiver handles VBUS generation and fault detection internally, and configurations where VBUS generation is external to transceiver but fault detection is handled internally. Also the active level of the VBUS fault indicator can be configured. The configuration is handled by the vbusconf generic. If UTMI+ transceivers are used it does not matter to the core how VBUS generation and fault detection is handled as long as the VBUS enable signal and VBUS fault indicator are connected to the core's drvvbus and vbusvalid signals respectively. The UTMI+ specification defines these two signals to be active high, however in order to support different types of USB power switches and fault detectors the core can be configured to have active low drvvbus and vbusvalid signals. This configuration is also handled by the vbusconf generic. The UTMI+ interface is described in *USB 2.0 Transceiver Macrocell Interface (UTMI) Specification* and *UTMI+ Specification Revision 1.0*. The ULPI interface is described in *UTMI+ Low Pin Interface (ULPI) Specification Revision 1.1*.

88.7 PCI configuration registers and legacy support

The VHDL generic *pcidev* is used to configure the core to be used as a PCI device. If the core is configured to be used as a PCI device then the PCI registers *Serial Bus Release Number Register* and *Frame Length Adjustment Register* are implemented in the enhanced controller. The *Serial Bus Release Number Register* is also implemented for the universal controller. See *Enhanced Host Controller Interface Specification (EHCI) for Universal Serial Bus revision 1.0* and *Universal Host Controller Interface (UHCI) Design Guide revision 1.1* for details.

Legacy support is not implemented.

88.8 Registers

88.8.1 Enhanced host controller

The core is programmed through registers mapped into APB address space. The contents of each register is described in the *Enhanced Host Controller Interface Specification (EHCI) for Universal Serial Bus revision 1.0. A* register called *RAM test control* is added when the VHDL generic *ramtest* is set to one. The *RAM test control* register is not part of the EHCI interface and is described below. Also registers mapped to the packet buffer and transaction buffer are added if RAM test facilities are implemented.

*Table 1749.*Enhanced Host Controller capability registers

APB address offset	Register
0x00	Capability Register Length
0x01	Reserved
0x02	Interface Version Number
0x04	Structural Parameters
0x08	Capability Parameters
0x0C	Companion Port Route Description

*Table 1750.*Enhanced Host Controller operational registers

APB address offset	Register
0x14	USB Command*
0x18	USB Status
0x1C	USB Interrupt Enable
0x20	USB Frame Index
0x24	4G Segment Selector (Reserved)
0x28	Frame List Base Address
0x2C	Next Asynchronous List Address
0x54	Configured Flag Register
0x58 - 0x90	Port Status/Control Registers**

*Light Host Controller reset is not implemented.

**One 32-bit register for each port

*Table 1751.*Enhanced Host Controller RAM test registers

APB address offset	Register
0x100 - 0xFFF	RAM test control*
0x1000 - 0x17FF	Packet buffer**
0x1800 - 0x1FFF	Transaction buffer**

*Register is only present if *ramtest* generic is set to one. Accessible through any of the offsets specified.

**Registers are only present if *ramtest* generic is set to one.

*Table 1752.*0x100 - RAMTEST - RAM test control

31 ... 2	1	0
RESERVED	ST	EN
0	0	0
r	w	w

*Table 1752.*0x100 - RAMTEST - RAM test control

31: 2	R (Reserved): Always reads zero.
1	ST (Start): Starts the automatic RAM test. Can only be written to '1'. Cleared by the core when test is finished.
0	EN (Enable): Enable RAM test. Need to be set to '1' in order to access the buffers.

*Table 1753.*Enhanced Host Controller PCI registers

APB address offset	Register
0x2060	PCI registers *Serial Bus Release Number Register* and *Frame Length Adjustment Register**

*Only implemented if configured to be used as a PCI device.

88.8.2 Universal host controller

The core is programmed through registers mapped into AHB I/O address space. The contents of each register is described in the *Universal Host Controller Interface (UHCI) Design Guide revision 1.1. A* register called *RAM test control* is added when the VHDL generic *ramtest* is set to one. The *RAM test control* register is not part of the UHCI interface and is described below. Also registers mapped to the packet buffer are added when RAM test facilities are implemented.

*Table 1754.*Universal Host Controller I/O registers

AHB address offset	Register
0x00	USB Command
0x02	USB Status*
0x04	USB Interrupt Enable
0x06	Frame Number
0x08	Frame List Base Address
0x0C	Start Of Frame Modify
0x10 - 0x2C	Port Status/Control**

*The HCHalted bit is implemented as Read Only and has the default value 1.

**Over Current and Over Current Change fields have been added. Each port has a 16-bit register.

*Table 1755.*Changes to USB Status register

15 … 6	5	4 … 0
UHCI compliant	HCH	UHCI compliant
	1	
	r	

15: 6	UHCI compliant
5	Host Controller Halted (HCH) - Same behaviour as specified in the UHCI specification but the field has been changed from Read/Write Clear to Read Only and is cleared when Run/Stop is set. The default value of this bit has been changed to 1.
4:0	UHCI compliant

*Table 1756.*Changes to Port Status/Control registers

15 … 12	11	10	9 … 0
UHCI compliant	OCC	OC	UHCI compliant
	0	0	
	wc	r	

15: 12	UHCI compliant

Table 1756. Changes to Port Status/Control registers

11	Over Current Change (OCC) - Set to 1 when Over Current (OC) toggles. Read/Write Clear.
10	Over Current Active (OC) - Set to 1 when there is an over current condition. Read Only.
9:0	UHCI compliant

Table 1757. Universal Host Controller RAM test registers

AHB address offset	Register
0x100 - 0x3FF	RAM test control*
0x400 - 0x7FF	Packet buffer**

*Register is only present if *ramtest* generic is set to one. Accessible through any of the offsets specified.

**Registers are only present if *ramtest* generic is set to one.

Table 1758. 0x100 - RAMTEST - RAM test control

31	2	1	0
RESERVED		ST	EN
*		0	0
r		w	rw

31: 2	R (Reserved): Always reads zero.
1	ST (Start): Starts the automatic RAM test. Can only be written to '1'. Cleared by the core when test is finished.
0	EN (Enable): Enable RAM test. Need to be set to '1' in order to access the buffers.

Table 1759. Universal Host Controller PCI registers

AHB address offset	Register
0x60	PCI register *Serial Bus Release Number Register**

*Only implemented if configured to be used as a PCI device.

88.9 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler), the enhanced host controller has device identifier 0x026, the universal host controller has device identifier 0x027. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

88.10 Implementation

88.10.1 Clocking and reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set. The core does not support *grlib_async_reset_enable*.

The core has two clock domains; a system clock domain and a USB clock domain. The USB clock domain always operates in either 60 MHz or 30 MHz, depending on the transceiver interface. All signals that cross a clock domain boundary are synchronized to prevent meta-stability.

The IP core has two different reset inputs, according to the two existing clock domains: the AMBA reset and the USB reset. They work independently on their respective clock domains. If reset generation/synchronization is desired, it shall be done in a higher instance, out of the IP, as this is not performed internally.

The reset input can be asserted and deasserted asynchronously or synchronously. All registers that in the system clock domain that require a reset value are synchronously reseted. It is assumed that the

reset input is held asserted until the system clock input is stable. In order to insure that no unwanted USB activity take place, a few registers in the USB clock domain are asynchronously reseted. Once the USB clock starts to toggle, all other registers in the USB domain that require a reset value will be reset as well.

Some ULPI transceivers require the data bus to be kept low by the core during transceiver reset, this behaviour is controlled by the *urstdrive* input signal.

88.10.2 RAM usage

The core maps all usage of RAM on either the *syncram_dp* component (dual port) or the *syncram_2p* component (double port), both from the technology mapping library (TECHMAP). Which component that is used can be configured with generics. The default, and recommended, configuration will use *syncram_dp.* A universal host controller requires one 256x32 *syncram_dp*, or two 256x32 *syncram_2p* for its packet buffer. The extra amount of *syncram_2p* needed comes from the fact that the packet buffer is both read and written from the AHB clock domain and the USB clock domain. It can not be guaranteed that a synchronization scheme would be fast enough and therefore one buffer for data beeing sent and one buffer for data beeing received are needed. An enhanced host controller requires one 512x32 *syncram_dp* (or two 512x32 *syncram_2p*, for the same reason discussed above) for its packet buffer and two 512x16 *syncram_dp/syncram_2p* for its transaction buffer. The transaction buffer is not doubled when using *syncram_2p*, instead synchronization and arbitration logic is added. When the core is instantiated with only one port, the enhanced host controller and universal host controller will share the packet buffer and the core only requires one 512x32 *syncram_dp* (or two 512x32 *syncram_2p*) for the packet buffer. Table 1760 below shows RAM usage for all legal configurations.

*Table 1760.*RAM usage for USB Host Controller core

Enhanced Host Controller present	Number of Universal Host Controllers	Number of ports	RAM component	RAM 256x32	RAM 512x32	RAM 512x16
No	x*	Don’t care	syncram_dp	x*	0	0
No	x*	Don’t care	syncram_2p	x**	0	0
Yes	1	1	syncram_dp	0	1	2
Yes	1	1	syncram_2p	0	2	2
Yes	x*	> 1	syncram_dp	x*	1	2
Yes	x*	> 1	syncram_2p	x**	2	2

* The number of required 256x32 *syncram_dp* equals the number of instantiated universal host controllers.

** The number of required 256x32 *syncram_2p* equals the double amount of instantiated universal host controllers.

88.10.3 ASIC implementation details

When synthesizing the core for ASIC it might be required to use DC Ultra to reach the desired performance of the AMBA interface.

88.10.4 Scan test support

The VHDL generic *scantest* enables scan test support. If the core has been implemented with scan test support it will:

- disable the internal RAM blocks when the testen and scanen signals are asserted.
- use the testoen signal as output enable signal.
- clock all registers with the clk input (i.e. not use the USB clock).
- use the testrst signal as the reset signal for those registers that are asynchronously reseted.

The testen, scanen, testrst, and testoen signals are routed via the AHB master interface.

88.10.5 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

88.11 Configuration options

Table 1761 shows the configuration options of the core (VHDL generics).

*Table 1761.*Configuration options

Generic name	Function	Allowed range	Default
ehchindex	Enhanced host controller AHB master index	0 - NAHBMST-1	0
ehcpindex	Enhanced host controller APB slave index	0 - NAPBSLV-1	0
ehcpaddr	Enhanced host controller ADDR field of the APB BAR.	0 - 16#FFF#	0
ehcpmask	Enhanced host controller MASK field of the APB BAR. Note that if the *ramtest* generic is set to 1 then the allowed range for this generic changes to 0 - 16#FE0#. If the pcidev generic is set to 1 then the allowed range for *ehcpmask* is 0 - 16#FC0#	0 - 16#FFF#	16#FFF#
ehcpirq	Enhanced host controller interrupt line	0 - NAHBIRQ-1	0
uhchindex	Universal host controller AHB master index. If the core contains more than one universal host controller the controllers will be assigned indexes from uhchindex to uhchindex+n_cc-1.	0 - NAHBMST-1	0
uhchsindex	Universal host controller AHB slave index. If the core contains more than one universal host controller the controllers will be assigned indexes from uhc_hsindex to uhchsindex+n_cc-1.	0 - NAHBSLV-n_cc	0
uhchaddr	Universal host controller ADDR field of the AHB BAR. If the core contains more than one universal host controller the controllers will be assigned the address space uhchaddr to uhchaddr + n_cc.	0 - 16#FFF#	0
uhchmask	Universal host controller MASK field of the AHB BAR. Note that if the *ramtest* generic is set to 1 then the allowed range for this generic changes to 0 - 16#FF8#	0 - 16#FFF#	16#FFF#
uhchirq	Universal host controller interrupt line. If the core contains more than one universal host controller the controller will be assigned interrupt lines uhc_hirq to uhchirq+n_cc-1.	0 - NAHBIRQ-1	0
tech	Technology for clock buffers	0 - NTECH	inferred
memtech	Memory Technology used for buffers.	0 - NTECH	inferred
nports	Number of USB ports	1 - 15	1
ehcgen	Enable enhanced host controller	0 - 1	1
uhcgen	Enable universal host controller(s)	0 - 1	1
n_cc	Number of universal host controllers. This value must be consistent with nports and n_pcc, or portroute1 and portroute2, depending on the value of the generic prr. This value must be at least 1, regardless the value of generic uhcgen.	1 - 15	1

*Table 1761.*Configuration options

Generic name	Function	Allowed range	Default
n_pcc	Number of ports per universal host controller. This value must be consistent with n_cc and nports: nports <= (n_cc * n_pcc) < (nports + n_pcc) when Port Routing Rules is disabled. The only allowed deviation is if (nports mod n_cc) < n_pcc in which case the last universal host controller will get (nports mod n_cc) ports. This generic is not used then Port Routing Rules (prr) is enabled.	1 - 15	1
prr	Port Routing Rules. Determines if the core's ports are routed to companion controller(s) with n_cc and n_pcc or with the help of portroute1 and portroute2.	0 - 1	0
portroute1	Defines part of the HCSP-PORTROUTE array	-	0
portroute2	Defines part of the HCSP-PORTROUTE array	-	0
endian_conv	Enable endian conversion. When set, all AMBA data lines are byte swapped. This generic must be set to 1 if the core is attached to a big endian bus, it must be set to 0 if the core is attached to a little endian bus.	0 - 1	1
be_regs	Arrange host controller registers according to big endian addressing. When set no endian conversion is made on the AMBA data lines connected to the host controller registers, regardless of endian_conv. Valid when endian_conv is enabled.	0 - 1	0
be_desc	Disable byte swapping of in-memory descriptors. Valid when endian_conv is enabled.	0 - 1	0
uhcblo	Universal Host Controller Buffer Limit Out. A universal host controller will start OUT bus transactions when uhcblo words of payload data has been fetched from memory. Note that if the core uses the UTMI+ 16 bit interface this generic must have a value larger than 2.	1 - 255	2
bwrd	Burst length in words. A universal host controller has a fixed, not affected by bwrd, burst length of four words when fetching transfer descriptors. See comments under section 88.2 DMA operations.	0 - 256	16
utm_type	Transceiver type: 0: UTMI+ 16 bit data bus 1: UTMI+ 8 bit data bus 2: ULPI	0 - 2	2

*Table 1761.*Configuration options

Generic name	Function	Allowed range	Default
vbusconf*	Selects configuration for USB power source and fault detection (external and internal below is from the USB transceivers point of view): ULPI transceivers: 0: ULPI transceiver generates VBUS internally and no external fault indicator present 1: External power source but no external fault indicator. Transceiver implement the optional ULPI pin DrvVbusExternal but not ExternalVbusIndicator. 2: External power source and external active high fault indicator. Transceiver implement both the optional ULPI signals DrvvbusExternal and ExternalVbusIndicator. 3: External power source and external active low fault indicator. Transceiver implement both the optional signals DrvvbusExternal and ExternalVbusIndicator. 4: External power source, but transceiver does not implement the optional ULPI signal DrvVbusExternal. Active low drvvbus output from Host Controller will be used. Don't care if ExternalVbusIndicator is implemented, not used. 5: External power source, but transceiver does not implement the optional ULPI signal DrvVbusExternal. Active high drvvbus output from Host Controller will be used. Don't care if ExternalVbusIndicator is implemented, not used. UTMI+ transceivers: 0: vbusvalid and drvvbus are both active low 1: vbusvalid is active low, drvvbus is active high 2: vbusvalid is active high, drvvbus is active low 3: vbusvalid and drvvbus are both active high	0 - 3	3
ramtest	When set each controller maps its internal buffers into the controller's register space.**	0 - 1	0
oepol	The polarity of the output enable signal for the data input/output buffers, 0 means active low and 1 means active high.	0 - 1	0
scantest	Scan test support will be included if this generic is set to 1.	0 - 1	0
memsel	Selects if dual port or double port memories should be used for the host controllers' internal buffers. Dual port memories are used if this generic is set to 0 (or MEMSEL_DUALPORT). Double port memories are used if this generic is set to 1 (or MEMSEL_DOUBLEPORT). It is strongly recommended to use dual port memories in the host controllers. Double port memories should only be used on technologies that lack support for dual port memories or where the area overhead for dual port memories preventively large.***	0 - 1	0
pcidev	This generic should be set to one if the core is to be used as a PCI device. If set to 1 the core will hold its interrupt signal(s) high until cleared by software. Also a few PCI registers will be added. See section 88.7 and 88.8 for details on the registers. Note that if this generic is set to 1 then the allowed range for *ehcpmask* changes to 0 - 16#FC0#.	0 - 1	0

*Table 1761.*Configuration options

Generic name	Function	Allowed range	Default

*see section 88.6 Transceiver support for more information

**see section 88.2.5 RAM test facilities for more information

***see section 88.10.2 RAM usage for more information

88.12 Signal descriptions

Table 1762 shows the interface signals of the core (VHDL ports).

*Table 1762.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	AMBA clock	-
UCLK	N/A	Input	USB clock	-
RST	N/A	Input	AMBA reset	Low
URST	N/A	Input	USB reset	Low
APBI	*	Input	APB slave input signals	-
EHC_APBO	*	Output	APB slave output signals	-
AHBMI	*	Input	AHB master input signals	-
AHBSI	*	Input	AHB slave input signals	-
EHC_AHBMO	*	Output	AHB master output signals.	-
UHC_AHBMO[]	*	Output	AHB master output vector.	
UHC_AHBSO[]	*	Output	AHB slave output vector.	-
O[]	xcvrselect[1:0]	Output	UTMI+	-
	termselect	Output	UTMI+	-
	suspendm	Output	UTMI+	Low
	opmode[1:0]	Output	UTMI+	-
	txvalid	Output	UTMI+	High
	dataout[15:0]	Output	UTMI+/ULPI. Bits 15:8 are only used in 16-bit UTMI+ mode.	-
	txvalidh	Output	UTMI+ 16-bit	High
	stp	Output	ULPI	High
	reset	Output	Transceiver reset signal. Asserted asynchronously and deasserted synchronously to the USB clock.	**
	oen	Output	Data bus direction control for ULPI and bi-directional UTMI+ interfaces.	***
	databus16_8	Output	UTMI+ Constant high for 16-bit interface, constant low for 8-bit interface.	-
	dppulldown	Output	UTMI+ Constant high.	High
	dmpulldown	Output	UTMI+ Constant high.	High
	idpullup	Output	UTMI+ Constant low.	High
	drvvbus	Output	UTMI+/ULPI	***
	dischrgvbus	Output	UTMI+ Constant low.	High
	chrgvbus	Output	UTMI+ Constant low.	High
	txbitstuffenable	Output	UTMI+ Constant low.	High
	txbitstuffenableh	Output	UTMI+ Constant low.	High
	fslsserialmode	Output	UTMI+ Constant low.	High

Table 1762. Signal descriptions

Signal name	Field	Type	Function	Active
	tx_enable_n	Output	UTMI+ Constant high.	High
	tx_dat	Output	UTMI+ Constant low.	High
	tx_se0	Output	UTMI+ Constant low.	High
I[]	linestate[1:0]	Input	UTMI+	-
	txready	Input	UTMI+	High
	rxvalid	Input	UTMI+	High
	rxactive	Input	UTMI+	High
	rxerror	Input	UTMI+	High
	vbusvalid	Input	UTMI+	***
	datain[15:0]	Input	UTMI+/ULPI. Bits 15:8 are only used in 16-bit UTMI+ interface.	-
	rxvalidh	Input	UTMI+ 16-bit	High
	hostdisconnect	Input	UTMI+	High
	nxt	Input	ULPI	High
	dir	Input	ULPI	-
	urstdrive	Input	This input determines if the cores should drive the transceiver data lines low during USB transceiver reset, even if the dir input is High. This is needed for some transceivers, such as the NXP ISP1504. When this input is low the direction of the transceiver data lines are exclusively controlled by the dir signal from the transceiver. When this input is high the core will drive the data lines low during transceiver reset. Only applicable for ULPI transceivers.	High

* See GRLIB IP Library User's Manual.

** Depends on transceiver interface. Active high for UTMI+ and active low for ULPI.

*** Implementation dependent.

88.13 Signal definitions and reset values

The signals and their reset values are described in table 1763.

Table 1763. Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
xcvrselect[1:0]	Output	UTMI+	-	-
termselect	Output	UTMI+	-	-
suspendm	Output	UTMI+	Logical 0	Logical 1
opmode[1:0]	Output	UTMI+	-	-
txvalid	Output	UTMI+	Logical 1	Logical 0
dataout[15:0]	Output	UTMI+/ULPI. Bits 15:8 are only used in 16-bit UTMI+ mode.	-	-
txvalidh	Output	UTMI+ 16-bit	Logical 1	Logical 0
stp	Output	ULPI	Logical 1	Logical 0
reset	Output	Transceiver reset signal. Set asynchronously and cleared synchronously to the USB clock.	*	*
oen	Output	Data bus direction control for ULPI and bi-directional UTMI+ interfaces.		

*Table 1763.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
databus16_8	Output	UTMI+ Constant high for 16-bit interface, constant low for 8-bit interface.	-	**
dppulldown	Output	UTMI+ Constant high.	Logical 1	Logical 1
dmpulldown	Output	UTMI+ Constant high.	Logical 1	Logical 1
idpullup	Output	UTMI+ Constant low.	Logical 1	Logical 0
drvvbus	Output	UTMI+	**	**
dischrgvbus	Output	UTMI+ Constant low.	Logical 1	Logical 0
chrgvbus	Output	UTMI+ Constant low.	Logical 1	Logical 0
txbitstuffenable	Output	UTMI+ Constant low.	Logical 1	Logical 0
txbitstuffenableh	Output	UTMI+ Constant low.	Logical 1	Logical 0
fslsserialmode	Output	UTMI+ Constant low.	Logical 1	Logical 0
tx_enable_n	Output	UTMI+ Constant low.	Logical 1	Logical 0
tx_dat	Output	UTMI+ Constant low.	Logical 1	Logical 0
tx_se0	Output	UTMI+ Constant low.	Logical 1	Logical 0
linestate[1:0]	Input	UTMI+	-	-
txready	Input	UTMI+	Logical 1	-
rxvalid	Input	UTMI+	Logical 1	-
rxactive	Input	UTMI+	Logical 1	-
rxerror	Input	UTMI+	Logical 1	-
vbusvalid	Input	UTMI+	***	-
datin[15:0]	Input	UTMI+/ULPI. Bits 15:8 are only used in 16-bit UTMI+ mode.	-	-
rxvalidh	Input	UTMI+ 16 bit interface	Logical 1	-
hostdisconnect	Input	UTMI+	Logical 1	-
nxt	Input	ULPI	Logical 1	-
dir	Input	ULPI	-	-

* Depends on transceiver interface. UTMI+ is active high, ULPI is active low.

** Implementation dependent

88.14 Library dependencies

Table 1764 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1764.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	GRUSB	Signals, component	Component declaration, USB signals

88.15 Instantiation

This example shows how the core can be instantiated.

```
library ieee, grlib, gaisler;
use ieee.std_logic_1164.all;
use grlib.amba.all;
use gaisler.grusb.all;

-- USB Host controller with 2 ports. One enhanced
```

```
-- host controller and two universal host controllers. Note that not all generics are set
-- in this example, many are kept at their defaulr values.

entity usbhc_ex is
  generic (
    tech    => tech;
    memtech => memtech;
    padtech => padtech);
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- USBHC signals
    usbh_clkin     : in std_ulogic;
    usbh_rstin     : in std_ulogic;
    usbh_d         : inout std_logic_vector(15 downto 0);
    usbh_reset     : out std_logic_vector(1 downto 0);
    usbh_nxt       : in std_logic_vector(1 downto 0);
    usbh_stp       : out std_logic_vector(1 downto 0);
    usbh_dir       : in std_logic_vector(1 downto 0)
    );
end;

architecture rtl of usbhc_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);

  -- USBHC signals
  signal usbhci : grusb_in_vector(1 downto 0);
  signal usbhco : grusb_out_vector(1 downto 0);
  signal uhclk : std_ulogic;
  signal uhrstn : std_ulogic;

begin

  -- AMBA Components are instantiated here
  ...

  -- Instantiate pads, one iteration for each port
  multi_pads: for i in 0 to 1 generate
    usbh_d_pad: iopadv
      generic map(tech => padtech, width => 8)
      port map (usbh_d((i*8+7) downto (i*8)),
                usbhco(i).dataout(7 downto 0), usbhco(i).oen,
                usbhci(i).datain(7 downto 0));
    usbh_nxt_pad : inpad generic map (tech => padtech)
      port map (usbh_nxt(i),usbhci(i).nxt);
    usbh_dir_pad : inpad generic map (tech => padtech)
      port map (usbh_dir(i),usbhci(i).dir);
    usbh_reset_pad : outpad generic map (tech => padtech)
      port map (usbh_reset(i),usbhco(i).reset);
    usbh_stp_pad : outpad generic map (tech => padtech)
      port map (usbh_stp(i),usbhco(i).stp);

    -- No need to drive ULPI data bus during USB reset
    usbhci(i).urstdrive <= '0';
  end generate multi_pads;

  usbh_clkin_pad : clkpad:
    generic map (tech => padtech)
    port map(usbh_clkin, uhclk);

  usbh_rstin_pad : inpad:
    generic map (tech => padtech)
    port map(usbh_rstin, uhrstn);
```

```
  usbhostcontroller0: grusbhc
    generic map (
      ehchindex => 5,
      ehcpindex => 14,
      ehcpaddr => 14,
      ehcpirq => 9,
      ehcpmask => 16#fff#,
      uhchindex => 6,
      uhchsindex => 3,
      uhchaddr => 16#A00#,
      uhchmask => 16#fff#,
      uhchirq => 10,
      tech => tech,
      memtech => memtech,
      nports => 2,
      ehcgen => 1,
      uhcgen => 1,
      n_cc => 2,
      n_pcc => 1,
      endian_conv => 1,
      utm_type => 2,
      vbusconf => 3)
    port map (
      clk => clk,
      uclk => uhclk,
      rst => rstn,
      urst => uhrstn,
      apbi => apbi,
      ehc_apbo => apbo(14),
      ahbmi => ahbmi,
      ahbsi => ahbsi,
      ehc_ahbmo => ahbmo(5),
      uhc_ahbmo => ahbmo(7 downto 6),
      uhc_ahbso => ahbso(4 downto 3),
      o => usbhco,
      i => usbhci);
end;
```

89 GRVERSION - Version and Revision information register

89.1 Overview

The GRVERSION provides a register containing a 16 bit version field and a 16 bit revision field. The values for the two fields are taken from two corresponding VHDL generics. The register is available via the AMBA APB bus.

89.2 Registers

The core is programmed through registers mapped into APB address space.

Table 1765. GRVERSION registers

APB address offset	Register
0x00	Configuration Register

89.2.1 Configuration Register

Table 1766. 0x00 - CONFIG - Configuration Register

31 16	15 0
VERSION	REVISION
*	*
r	r

31-16: VERSION Version number
15- 0: REVISIONRevision number

89.3 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x03A. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

89.4 Implementation

89.4.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

89.5 Configuration options

Table 1767 shows the configuration options of the core (VHDL generics).

Table 1767. Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
versionnr	Version number	0 - 2^16-1	0
revisionnr	Revision number	0 - 2^16-1	0

89.6 Signal descriptions

Table 1768 shows the interface signals of the core (VHDL ports).

Table 1768. Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-

* see GRLIB IP Library User's Manual

89.7 Library dependencies

Table 1769 shows the libraries used when instantiating the core (VHDL libraries).

Table 1769. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

90 GRWATCHDOG - Watchdog module with separate clock domain

90.1 Overview

This core implements a watchdog timer intended to detect when a processor has crashed or become non-responsive. The timer consists of a down-counter, that runs in a separate clock domain from the on-chip bus system. Running it in another clock domain allows catching cases where the system clock stops functioning (for example, due to PLL malfunction).

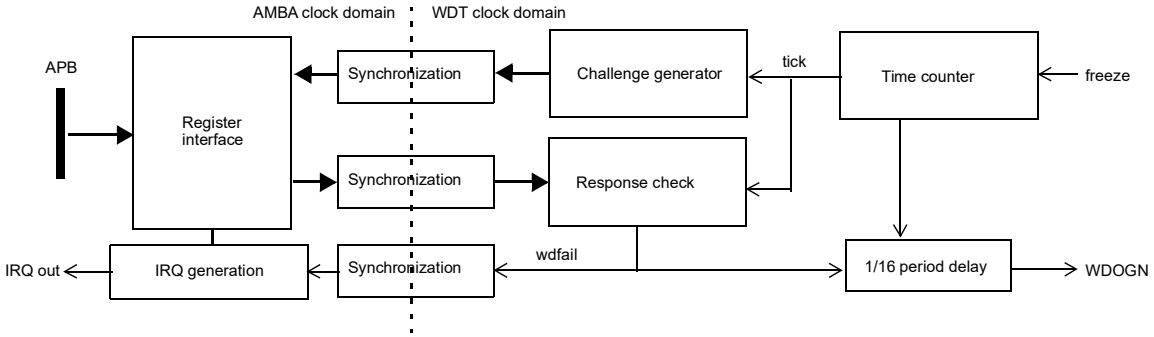


Figure 243. Functional block diagram of GRWATCHDOG

90.2 Operation

90.2.1 Challenge / Response protocol

The watchdog operates by a continuous cycle of posting an value (challenge), waiting for a pre-determined amount of time, and then checking that the correct response value was written in by software. If the value is correct, another challenge value is produced and the cycle repeats. If the value is incorrect, the watchdog will assert a watchdog triggered signal.

Both the challenge and response are 8-bit values. The response is formed by rotating the value two bit positions to the left and inverting the lower two bits. The formula for producing the correct response for a certain challenge is, in C syntax:

resp = ((chal & 0x3f) << 2) | ~((chal & 0xc0) >> 6);

The intent of this challenge/response protocol is to ensure that the writes to the watchdog are made intentionally by software and not by stray DMA accesses or through accidental writes by software via corrupted pointers. While a stray write to the watchdog's register space could by accident get the response right once it is very unlikely to produce the right response several times.

90.2.2 Watchdog triggered signal

When a challenge/response cycle fails, the watchdog will trigger. In order to allow for a controlled shutdown whenever possible, the watchdog triggers in two stages. First, a watchdog triggered indication is sent to the AMBA clock domain. This can be read out through the status register, and optionally an interrupt can be generated. The second stage occurs after an additional 1/16:th of the watchdog period time has passed, then the active-low watchdog triggered signal (wdogn) will go low, and will stay low until the core is reset.

The watchdog module itself does not take any further action beyond asserting the watchdog output signal. The signal would normally be used to trigger a system reset, either on-chip or through an external signal on board level.

90.2.3 Timer freeze

The core has a timer freeze control signal that suspends the watchdog counter. This is intended to be used to stop the watchdog timer during software debugging. Note that accidentally raising this signal could defeat the watchdog functionality, it is therefore recommended to ensure that this signal is low when not debugging (for example by gating it with a debug-enable input signal).

90.3 Implementation

90.3.1 Synchronization

The core uses the syncreg component from the techmap library for the synchronization registers. For all signals from the system clock domain into the watchdog clock domain, three-stage synchronization registers are used. For the transfer of data from the watchdog clock domain into the system, a handshaking protocol is used and the cdcbus component is used for the data bus capture.

The timer freeze signal is internally synchronized into the wdt_clk domain.

90.3.2 Endianness

This core has only an APB register interface that supports one access size (32 bits), and is therefore not impacted by endianness issues.

90.3.3 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_async_reset_enable* is set.

90.4 Registers

The bridge provides a configuration register area accessed over the APB bus. For forward compatibility, reserved fields should be written either with 0 or with the last read-out value, and reserved registers should not be accessed at all.

Table 1770. Configuration registers for GRWATCHDOG

Register address offset	Name	R/W	Reset value	Notes
0x00	Hardware configuration register	R	-	1
0x04	Control register	R/W	0x00000000	
0x08	Status register	R	0x00000001	
0x0C	Response register	R/W	0x00000000	
0x10 - 0xFC	RESERVED			

Note 1: Reset value depends on IP configuration options

90.4.1 Hardware configuration register

Table 1771. 0x00 - GRWATCHDOG hardware configuration register

31	16
RESERVED	
0	
r	

15	7	6	5	0
RESERVED		SMCLK	TOBITS	
0		*	*	
r		r	r	

31 : 7 Reserved

6 Set to 1 if the watchdog timer is in the same clock domain as the system

5 : 0 Size of timeout counter (TOBITS), specifies the number of bits in the watchdog time conter, minus one. The timeout is $2^{(TOBITS+1)}$ cycles in the wdt clock domain

90.4.2 Control register

Table 1772. 0x04 - GRWATCHDOG control register

31	16
RESERVED	
0	
r	

15	1	0
RESERVED		IRQEN
0		0
r		rw

31 : 1 Reserved

0 Watchdog triggered interrupt enable (IRQEN) - set to 1 to enable delivery of the watchdog interrupt when the watchdog triggered status (WDT field in the status register) goes from 0 to 1.

90.4.3 Status register

Table 1773. 0x08 - GRWATCHDOG status register

31	16
RESERVED	
0	
r	

15	10	9	8	7	0
RESERVED		WDT	CHTOG	CHALLENGE	
0		0	0	00000001	
r		r	r	r	

31 : 10 Reserved

9 Watchdog timer triggered status, set to 1 after a failed challenge/response cycle.

8 Challenge toggle (CHTOG). This status bit changes value between 0 and 1 each time there is a new challenge value.

7 : 0 Current challenge value

90.4.4 Response register

Table 1774. 0x00 - GRWATCHDOG response register

31	16
RESERVED	

Table 1774. 0x00 - GRWATCHDOG response register

0	
r	

15	8	7	0
RESERVED		RESPONSE	
0		00000000	
r		rw	

31 : 8 Reserved

7 : 0 Response value to the current challenge. Should be written in by software to avoid triggering the watchdog.

90.5 Vendor and device identifiers

This core is identified with vendor ID 0x001 (Frontgrade Gaisler) and device ID 0x0C9. The version described in this document is version 0.

90.6 Configuration options

Tables 1775 shows the configuration options (VHDL generics) of the core.

*Table 1775.*Configuration options for GRWATCHDOG

Generic	Function	Allowed range	Default
fabtech	Technology constant (passed on to techmap components)	0 - NTECH	0
pindex	APB index for APB slave port	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR for the register port.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR for the register port.	0 - 16#FFF#	0
pirq	Interrupt number for the PIRQ bus	0 - 16#FFF#	0
tobits	Number of bits in the watchdog timer counter, determines the timeout period of the watchdog.	4 - 64	16
sameclk	Set to 1 if the wdt and apb clocks are the same. Removes synchronization registers between the clock domains.	0 - 1	0

90.7 Signal descriptions

Table 1776 shows the interface signals (VHDL ports) of the of the core.

*Table 1776.*Signal descriptions for GRWATCHDOG

Signal name	Field	Type	Function	Active
APB_CLK	N/A	Input	APB (system/processor) clock	Rising
APB_RSTN	N/A	Input	Reset input for APB clock domain	Low
WDT_CLK	N/A	Input	Watchdog timer clock	Rising
WDT_RSTN	N/A	Input	Reset input for WDT clock domain	Low
APBI	*	Input	Inputs for APB port	-
APBO	*	Output	Outputs for APB port	-
WDOGN	N/A	Output	Watchdog triggered indication	Low
TFREEZE	N/A	Input	Timer freeze	High

* see GRLIB IP Library User's Manual

90.8 Library dependencies

Tables 1777 shows libraries used when instantiating the core (VHDL libraries).

*Table 1777.*Library dependencies for instantiating ftaddr_gr top level

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	Signal record definitions
GAISLER	MISC	Component	Component declaration
TECHMAP	GENCOMP	Constants	Constants for tech generic

90.9 Component declaration

The component declaration for the core is provided below.

```
component grwatchdog is
   generic (
     fabtech : integer               := inferred;
     pindex  : integer               := 0;
     paddr   : integer               := 0;
     pmask   : integer               := 16#FFF#;
     pirq    : integer               := 0;
     tobits  : integer range 4 to 64 := 4;
     sameclk : integer range 0 to 1  := 0
     );
   port (
     apb_clk  : in  std_ulogic;
     apb_rstn : in  std_ulogic;
     wdt_clk  : in  std_ulogic;
     wdt_rstn : in  std_ulogic;
     apbi     : in  apb_slv_in_type;
     apbo     : out apb_slv_out_type;
     wdogn    : out std_ulogic;
     tfreeze  : in  std_ulogic
     );
 end component;
```

91 GRWIZL - WizardLink codec with DMA engine and Bus Master Interface

91.1 Overview

GRWIZL provides an interface between the system bus (such as AHB or AXI) and a High Speed Serial Link (HSSL). The IP has been designed to inter-operate with the WizardLink transceiver family from Texas Instrument, in particular the TLK2711 transceiver. However, the IP can be used with other SerDes devices (Serializer / Deserializer).

The IP core consists of a WizardLink codec and a DMA engine for autonomous data transfers. Compared to SpaceFibre, the codec is much simpler and most of the higher-layer functionality has been removed, since WizardLink is not a closed standard. Therefore, it is the responsibility of upper layers (i.e. configurable commands in the DMA engine or software) to assign a specific functionality to the control words supported in the serial communication.

The design of the WizardLink codec is based on the lowest layer of the SpaceFibre codec, namely the interface layer. It handles the clock-domain-crossing synchronization mechanism between the core clock, the transmitter clock and the clock recovered by the receiver. If enabled, the codec can additionally perform 8b10b encoding (transmission) and decoding (reception), as well as comma detection and word synchronization.

The DMA engine is in charge of autonomously transferring packets between the codec and external working memory via a generic bus master interface. The core is configured and controlled via a set of registers accessed over an AHB slave interface. The generic bus master interface can easily be converted into the desired bus standard. There are wrappers available adapting the generic bus master interface to either AMBA AHB 2.0 or AXI4.

GRWIZL requires an external SerDes for the connection to the high speed serial link, either on the chip or external to the chip. In total, there are up to four clock domains: the system clock for the AMBA and DMA layers, the WizardLink codec clock, the WizardLink transmitter clock (optional) and the WizardLink recovered clock for the receiver.

The block diagram of GRWIZL can be seen in the figure below:

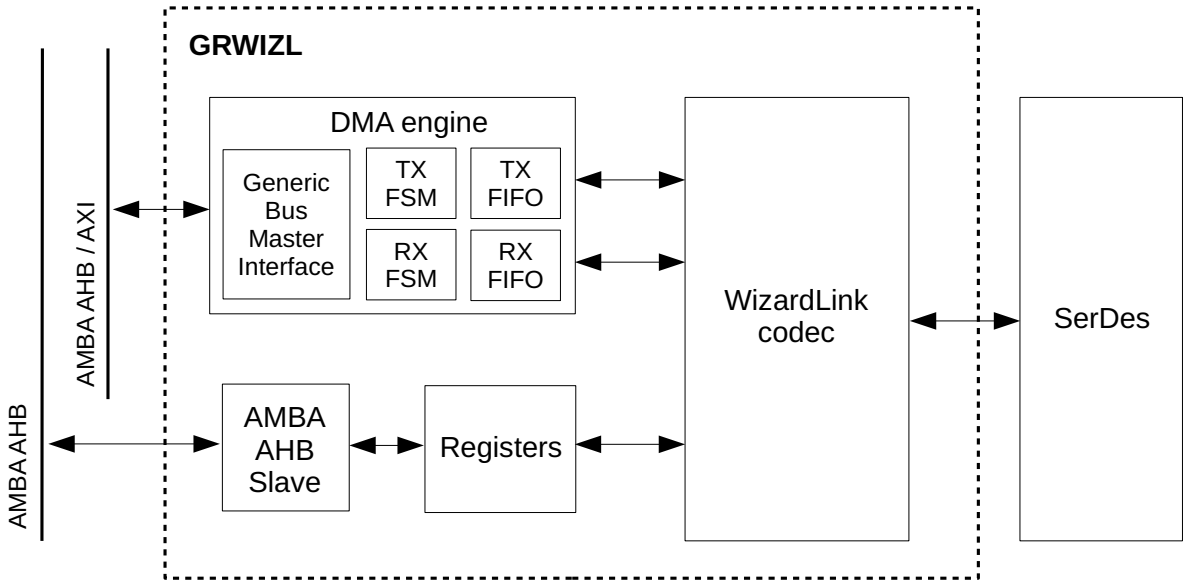


Figure 244. GRWIZL block diagram

91.2 Operation

91.2.1 Functional description

The main sub-blocks of the IP are the WizardLink codec, the DMA engine and the IP registers accessed through the AHB slave interface. Note that the SerDes is not part of the IP, but it is always required in order to achieve the typical high bit-rates of the High Speed Serial Links.

The core of the IP is the WizardLink codec. It has been designed to inter-operate the TLK2711 transceiver, but it supports other models of transceivers. The codec can be split into two basic layers:

- Interface Layer. This layer serves as the interface between the Physical and the Lane layers. It is in charge of the symbol synchronization, 8B/10B encoding, cross-domain crossing and polarity inversion, among other tasks.
- Lane Layer. Functions here are those typically related to the Data-Link layer, according to the OSI model. Idle words are inserted and extracted, and reception errors are also detected. Note that, due to the open nature of the standard, the codec does not follow any specific sequence to establish the link.

Additionally, the codec features internal TX and RX buffers to hold data words and K-flags for both transmission and reception. The depth of these buffers is configurable at implementation time via VHDL generics. The TX and RX buffers are directly interfaced by the Lane Layer.

Transmission and reception of packets are based on descriptors. There are two descriptor tables: one for transmission and another for reception. Once a descriptor has been read and verified by the DMA channel, the packet can be fetched from external memory and transmitted using the codec (transmitter) or read from the codec and stored into external memory (receiver). Packets and descriptors are read and written through the generic bus master interface.

The DMA engine also contains an RX and a TX FIFO for 32-bit word alignment and for burst support. The FIFOs serve as the interface between the FSMs and the codec. Once a complete packet has been read by the WizardLink codec (transmission) or written to external memory (reception), the FIFO is emptied, ready for the next packet. As is the case of the codec buffers, the size of these FIFOs can be configured at implementation time via generics.

The bus master interface present in the IP is generic, so that the user can connect a wrapper in order to adapt it to their custom protocol. Wrappers for AMBA 2.0 AHB and AXI4 are readily available from Frontgrade Gaisler.

In order to enhance protocol support in hardware, the DMA engine includes programmable commands that allow the user to transmit and receive control words without needing to set up descriptors. The core can generate interrupts when these commands are transmitted or received. This is an optional feature that can be used to easily insert and extract control words that do not necessarily belong to a WizardLink packet, for instance when starting or stopping the link, or as part of a custom flow control mechanism. The core currently supports up to 8 programmable commands.

GRWIZL includes the registers to configure and control the IP. These registers are accessible through the AMBA AHB slave interface.

GRWIZL will generate AMBA interrupts under certain circumstances:

- Link events
- Errors during read/write accesses through the bus master interface
- Transmission and reception of packets
- Transmission and reception of programmable commands

The user can configure which events shall cause interrupts both at IP level (by an AHB register) and per individual packet (a specific bit in the descriptor).

91.2.2 Protocol support

The WizardLink codec has been designed to interface the TLK2711 transceiver with minimal protocol overhead. Other SerDes models are supported by the IP.

91.2.3 Endianness

The core is designed for big-endian and little-endian systems.

When connected to an AHB bus, GRWIZL automatically detects the endianness of the system by reading a dedicated sideband signal included in the AMBA records. If the AXI wrapper is used instead, the endianness is configured via the VHDL generic *lendian*. When accessing the bus, the data may be swapped depending on the selected endianness and the type of data involved in the transfer:

- Words belonging to TX and RX descriptors are never swapped and shall be read and written using 32-bit accesses. These words shall be located at the same address offsets (0x0, 0x4, 0x8 and 0xC) regardless of the endianness.
- Data and K-flags in WizardLink packets are treated as a stream of bytes, i.e. the bytes will be located in the same memory positions in both big- and little-endian systems. If *addr* is the address configured in the corresponding field of a descriptor, the first byte shall be located at *addr*, the second byte at *addr+1*, and so on. This implies that GRWIZL will internally swap the data words in little-endian configurations when accessing the AMBA bus.

91.3 WizardLink codec

This section provides guidelines for configuring and controlling the internal WizardLink codec.

91.3.1 Configuration

The generics of the WizardLink codec are mainly related to the interface with the SerDes.

Communication over High Speed Serial Links normally requires 8B/10B encoding. However, some SerDes may already incorporate the encoding; otherwise this can be performed inside the codec by setting the generic *use_8b10b*.

Depending on the SerDes interface, the IP can be configured to have a 20- or 40-bit wide interface with the generic *sel_16_20_bit_mode*. If the narrow interface of 20 bits is chosen, the generic *use_sep_txclk* shall be enabled, and the transmitter clock input signal (wizl_txclk) shall be provided.

The lane layer contains TX and RX FIFOs used to buffer the words to transmit or received, respectively. The memory data width is fixed (36 bits: 32 of data plus 4 of K-codes), whereas the depth of the FIFOs is configurable via the generics *depth_tx_buf* and *depth_rx_buf*.

91.3.2 Control

The WizardLink communication is controlled by means of the Codec Control Register (CTRL). This register allows to enable or disable the transmitter and receiver side of the codec and the SerDes, as well as the internal loopback mode of the codec.

In loopback mode, the IP loops the data on both sides: codec (lane layer) and SerDes (interface layer). This implies that any data transmitted by the lane layer is automatically looped back to the receiver but not propagated to the SerDes, whereas any data received by the SerDes is automatically retransmitted over the High Speed Serial Link but not passed to the lane layer of the codec.

Since the control characters and words of WizardLink are not specified, there are registers to customize the protocol according to the needs of a particular application. Some of the most important registers are listed below:

- Synchronization comma. Since the 8b10b encoding supports different possibilities for the comma character, the user shall specify the pattern to be used by the codec to perform comma detection. It is assumed that the comma is always a control character (i.e. its K-flag is set to 1).
- EOP (End of Packet) and EEP (Error Packet) markers. Defining at least the EOP is mandatory in order to establish boundaries between consecutive packets. If EEP is not desired it can simply be set to the same value as EOP, so it will never be detected by the IP (EOP takes precedence). It is assumed that EOP and EEP are always control characters (i.e. their K-flags are always set to 1).
- IDLE words. When the IP does not have any word to transmit, it will continue sending IDLE words in order to keep the link active. The user shall specify both the data and the corresponding K-codes of this word.

The control characters and words need to be configured by the user before asserting the transmitter and receiver enable bits of the Codec Control Register in order to avoid transmitting invalid words via the high speed serial link.

91.3.3 Status

The codec provides several signals to inform about its internal status. These signals are mapped to the Codec Status Register (STS). Among others, the following signals are available:

- RX Error counter and counter overflow flag.
- RX overrun.

Note that, unlike in SpaceWire or SpaceFibre, there is no built-in mechanism for flow control. This means that the internal RX FIFO of the codec may get full if the DMA engine cannot keep up with the high speed serial link. As stated before, it is responsibility of the user to configure the application to deal with these situations, either by adding a flow control mechanism in software or by ensuring that the DMA engine can handle the incoming traffic at the target speed.

The capability registers contain information regarding how the IP was generated. Most generics can be read back by reading the Codec Capability Register.

91.4 DMA engine

91.4.1 Transmit channel

A WizardLink packet consists of data bytes and their corresponding K-flags. Data and K-flags are located in different areas in memory, and both shall be provided by the user. Since WizardLink does not associate any functionality with the control characters (bytes with their K-flag asserted) and words, by default this shall be done in software in order to support any application using WizardLink with a custom set of commands.

Transmission of packets is handled by means of descriptors. TX descriptors indicate where the packet is to be fetched from and control several aspects of the transmission process, such as interrupt generation and how the hardware pointers handling the descriptors shall be updated.

The core reads descriptors from an area in memory pointed to by the Transmit Descriptor Table Address Register. The register consists of a base address and a descriptor selector. The base address points to the beginning of the area and must start on an address that is aligned to the size of the descriptor table.

The number of descriptors in the table can be configured by the generic *num_txdesc*. Accepted values are 64, 128, 256 and 512. The number of bits allocated to the base address and descriptor selector in the Transmit Descriptor Table Address Register depends on the value of this generic. A TX descriptor consists of four 32-bit words (16 bytes). Therefore, the size of the descriptor table can be determined as *num_txdesc * 16* bytes. The number of TX descriptors can also be obtained by reading the DMA Capabilities Register.

To transmit packets, one or more descriptors have to be initialized in memory. Each descriptor shall specify the number of words to transmit and the location of both the data and K-flags. Note that not only the data, but also the K-flags shall be provided by the user in order to support any WizardLink application with custom control words and characters, even if the packet only consists of data characters with their K-flags deasserted.

The TX descriptor contains two different address fields for the data and the K-flags, respectively. The data length is unique, as there shall be a K-flag (one bit) per data byte. If the length field is zero, the packet is skipped and the descriptor is updated. The maximum data length is 32 MiB - 1. When the pointer and length fields have been configured the descriptor enable bit should be set to validate the descriptor. This must always be done last. The other control bits must also be set before enabling the descriptor.

Once the TX descriptors have been written to the descriptor table, the user shall flag this by setting the New Transmit Descriptor bit (NTD) in the Descriptor Control Register. The transmission will start as soon as the transmitter is enabled in the DMA Control Register. The user can also clear the TX descriptor table by setting the Clear Transmit Descriptor bit (CTD) in the Descriptor Control Register. Note that this will not abort any ongoing transmission if a previous descriptor had already been validated by the IP.

The transmission finite-state machine controls the exchange of data between external memory via the AMBA bus and the WizardLink codec. Firstly the FSM fetches the TX descriptor from memory and decodes it. The descriptor is validated, meaning that the enable bit shall be set. If so, the FSM proceeds to fetch the data bytes and K-flags provided that the packet length is not zero. If the enable bit of the descriptor was 0, the FSM automatically clears the TX descriptor available bit in the Descriptor Control Register.

When fetching the content of the packet, the FSM will interleave a read access of 128 K-flags (16 bytes) with 128 bytes of data. This process is repeated until the complete packet has been read out from memory. The FSM will then wait until the WizardLink codec has finished the transmission.

Once the transmission ends, GRWIZL overwrites the first word of the TX descriptor in order to clear the descriptor enable bit, which indicates that the IP has already used the descriptor. An AMBA interrupt will be generated if enabled.

The DMA Status Register contains valuable information regarding the transmission process. The TX Active bit indicates that the transmit FSM is enabled and may be in a status other than idle: a transmission may be ongoing. Disabling the transmit channel will not have any effect until the ongoing operation ends (if there is any). The TX packet bit indicates that a packet has been successfully fetched from memory and transmitted, and the descriptor has been updated accordingly.

The transmit channel will continue reading descriptors and transmitting packets until it encounters a disabled TX descriptor, or until the transmit descriptor table is cleared by the user. New descriptors can be activated in the table on the fly (while transmission is active). Each time a set of descriptors is added the NTD bit shall be set again, as the TX Descriptor Available (TDA) bit is automatically cleared when the DMA engine finds a descriptor disabled.

The internal pointer used to keep the current position in the descriptor table can be read and written through the AHB interface. This pointer is set to zero during reset and is incremented each time a descriptor is used. It wraps automatically when the limit for the descriptor table is reached, or it can be set to wrap earlier by setting a bit in the current descriptor.

The descriptor table register can be updated with a new table anytime when the transmitter is not active. A status bit is available in the DMA Status Register. A value of ‘0’ in the TX Active bit (TA) shall be read to consider it safe to modify the TX descriptor table. Once that happens, the user may change the location of the descriptor table, the content of the current table or the pointer to the next descriptor to be fetched by the core.

91.4.2 TX descriptors

Transmit descriptors consist of four 32-bit words that set up the transmission of the packets. The tables below depict each word in a TX descriptor and further explains every field:

Table 1778. GRWIZL TX descriptor word 0 (offset 0x0)

31	30	29	28 ... 0
IE	WR	EN	RESERVED

31	Interrupt enable (IE) - If set, an interrupt will be generated when the packet has been transmitted and the transmission interrupt enable bit in the Interrupt control register is set.
30	Wrap (WR) - If set, the descriptor pointer will wrap and the next descriptor read will be the first one in the table (at the base address). Otherwise the pointer is increased with 0x10 to use the descriptor at the next higher memory location.
29	Enable TX descriptor (EN) - When all control fields (address, length, wrap) are set, this bit should be set. While the bit is set the descriptor should not be touched since this might corrupt the transmission. The core clears this bit when the transmission has finished.
28: 0	RESERVED

Table 1779. GRWIZL TX descriptor word 1 (offset 0x4)

31 ... 25	24 ... 2	1	0
RESERVED	DATALEN		RES

31: 25	RESERVED
24: 2	Data length (DATALEN) - Number of 32-bit words of packet data. If set to zero, no packet will be sent. This also defines the number of K-flags to fetch: four bits per data word.
1: 0	RESERVED

Table 1780. GRWIZL TX descriptor word 2 (offset 0x8)

31 ... 0
KFLAGSADDR

31: 0	K-flags address (KFLAGSADDR) - Address from where K-flags are read. No alignment restriction.

Table 1781. GRWIZL TX descriptor word 3 (offset 0xC)

31 ... 0
DATAADDR

31: 0	Data address (DATAADDR) - Address from where data is read. No alignment restriction.

91.4.3 Receive channel

A WizardLink packet consists of data bytes and their corresponding K-flags. After receiving a WizardLink packet, the core will store the data and K-flags in different areas of memory. Since WizardLink does not associate any functionality with the control characters (bytes with their K-flag asserted) and words, by default the incoming data shall be interpreted in software in order to support any application using WizardLink with a custom set of commands.

As in transmission, reception of WizardLink packets is handled by means of descriptors. In this case, the descriptor indicates where the packet is to be stored, both the data and the K-flags. Some additional control bits are available as well to handle the interrupt generation and how the descriptor pointer shall be updated once the next reception finishes. Optionally, the user may specify the expected length of the data by writing the data length field. Otherwise, the packet length is determined once the EOP or EEP markers are received.

The core reads the descriptors from an area in memory pointed to by the Receive Descriptor Table Address Register. The register consists of a base address and a descriptor selector. The base address points to the beginning of the area and must start on an address that is aligned to the size of the descriptor table.

The number of descriptors in the table can be configured by the generic *num_rxdesc*. Accepted values are 128, 256, 512 and 1024. The number of bits allocated to the base address and descriptor selector in the Receive Descriptor Table Address Register depends on the value of this generic. An RX descriptor consists of four 32-bit words (16 bytes). Therefore, the size of the descriptor table can be determined as *num_rxdesc * 16* bytes. The number of RX descriptors can also be obtained by reading the DMA Capabilities Register.

WizardLink packets are received by the codec regardless of whether a descriptor has been fetched. The data and K-flags are internally written to a temporary buffer, waiting for the DMA channel to read its content and store it in external memory. For the latter to happen, RX descriptors shall be initialized so that they can be fetched and validated by the DMA engine.

Similarly to the transmitter, RX descriptors include two pointers to the locations where the data and K-flags shall be stored, respectively. Once the pointers are set and the control bits (wrap bit, interrupt enable, data length) configured, the enable bit should be set to validate the descriptor. This must always be done last to avoid reading a corrupt descriptor.

In this case, however, the length of the packet determines how the reception is terminated. If set to 0, the reception only finishes once either EOP or EEP are detected by the IP, and this field is subsequently updated by the core with the resulting packet length. Otherwise, the core will ignore the EOP and EEP markers and will store as many words as specified via the data length before ending the reception and updating the descriptor. Note that the number of K-flags stored in memory can easily be inferred from the total length as four bits per word of data.

Once the RX descriptors have been written to the descriptor table, the user shall set the New Receive Descriptor bit (NRD) in the Descriptor Control Register. The storage of the packet will then start as soon as the receiver is enabled in the DMA Control Register. The user can also clear the RX descriptor table by setting the Clear Receive Descriptor bit (CRD) in the Descriptor Control Register. Note that this will not abort any ongoing reception if a previous descriptor had already been validated by the IP.

The reception finite-state machine controls the exchange of data between the WizardLink codec and the external memory, which is accessed via the AMBA bus. Firstly the FSM fetches the RX descriptor from memory and decodes it. The descriptor is validated, meaning that the enable bit shall be set. If so, the DMA engine checks if the codec has received data. If the enable bit of the descriptor was 0, the FSM automatically clears the RX descriptor available bit in the Descriptor Control Register.

When the codec output FIFO contains at least 128 bytes of data, or the EOP/EEP has already been detected by the codec, the DMA engine starts a burst access of up to 128 data bytes. This is followed by another access to memory in order to store the K-flags associated with the data just written. These accesses are interleaved until the complete packet has been written to external memory.

Once the reception ends, GRWIZL overwrites the first word of the RX descriptor with the corresponding status bits: length of the packet received, EEP detected and whether the packet was truncated due to maximum length violation. It also clears the descriptor enable bit, which indicates that the IP has already used the descriptor. An AMBA interrupt will be generated if enabled.

The DMA Status Register contains valuable information regarding the reception process. The RX Active bit indicates that the receive FSM is enabled and may be in a status different from idle: it may have already started processing a packet. Disabling the receive channel will not have any effect until the ongoing operation ends (if there is any). The RX packet bit indicates that a packet has been received and stored in external memory, and the descriptor has been updated accordingly.

The receive channel will continue reading descriptors and storing packets until it encounters a disabled RX descriptor, or until the receive descriptor table is cleared by the user. New descriptors can be

activated in the table on the fly (while reception is active). Each time a set of descriptors is added the NRD bit shall be set again, as the RX Descriptor Available (RDA) bit is automatically cleared when the DMA engine finds a descriptor disabled.

The internal pointer used to keep the current position in the descriptor table can be read and written through the AHB interface. This pointer is set to zero during reset and is incremented each time a descriptor is used. It wraps automatically when the limit for the descriptor table is reached, or it can be set to wrap earlier by setting a bit in the current descriptor.

The descriptor table register can be updated with a new table anytime when the receiver is not active, i.e. the RX Active (RA) bit in the DMA Status Register shall be set to 0. In that case, the user may safely change the location of the descriptor table, the content of the current table or the pointer to the next descriptor to be fetched by the core.

91.4.4 RX descriptors

Receive descriptors consist of four 32-bit words that set up the storage of the packets received by the codec. The tables below depict each word in an RX descriptor and further explains every field:

Table 1782. GRWIZL RX descriptor word 0 (offset 0x0)

31	30	29	28	27	26 25	24 ... 2	1 0
IE	WR	EN	TR	EP	RES	PKTLEN	RES

31	Interrupt enable (IE) - If set, an interrupt will be generated when a packet has been received and the reception interrupt enable bit in the Interrupt control register is set.
30	Wrap (WR) - If set, the descriptor pointer will wrap and the next descriptor read will be the first one in the table (at the base address). Otherwise the pointer is increased with 0x10 to use the descriptor at the next higher memory location.
29	Enable RX Descriptor (EN) - Set to one to activate this descriptor. This means that the descriptor contains valid control values and the memory area pointed to by the packet address field can be used to store a packet. When all control fields are set, this bit should be set. While the bit is set the descriptor should not be touched since this might corrupt the reception. The core clears this bit when the reception has finished.
28	Packet truncated (TR) - Set by the core when the reception has finished if the length of the packet exceeds the maximum length. The rest of the packet is spilled.
27	EEP termination (EP) - Set by the core to indicate that the packet ended with an Error End of Packet control character.
26: 25	RESERVED
24: 2	Packet length (PKTLEN) - Number of 32-bit words of packet data, excluding the K-flags. When the descriptor is set up, this field determines how the reception shall terminate. If set to 0 by the user, the reception finishes whenever the IP detects EOP or EEP within the packet data. Otherwise, the reception finalizes once the DMA engine has received the number of words programmed in this field. The core updates this field with the resulting packet length when setting the EN bit to 0, once the reception has finished.
1: 0	RESERVED

Table 1783. GRWIZL RX descriptor word 1 (offset 0x4)

31 ... 0
RESERVED

31: 0	RESERVED

Table 1784. GRWIZL RX descriptor word 2 (offset 0x8)

31 ... 0
KFLAGSADDR

Table 1784. GRWIZL RX descriptor word 2 (offset 0x8)

31: 0	K-flags address (KFLAGSADDR) - The address pointing to the buffer which will be used to store the K-flags of the packet received.

Table 1785. GRWIZL RX descriptor word 3 (offset 0xC)

31	0
DATAADDR	

31: 0	Data address (DATAADDR) - The address pointing to the buffer which will be used to store the data bytes of the packet received.

91.4.5 Packet maximum length filter (RX only)

GRWIZL features a mechanism to limit the size of the incoming packets that will be stored in memory.

The RX Maximum Length Register defines the maximum number of 32-bit words that will be accepted by the core for a given WizardLink packet. If the length of a packet exceeds the value of the register, the Truncate bit in the RX descriptor is set and the rest of the packet is spilled, i.e. not stored in external memory. It is mandatory to configure this register prior to starting a reception, otherwise the default length is zero and no words are accepted.

Note that this filter is only active when the IP expects the WizardLink packet to be terminated with either EOP or EEP, meaning that the user shall set the data length to 0 when initializing the RX descriptor.

91.4.6 Address filters (RX only)

GRWIZL features an optional mechanism to filter packets based on their address. When a WizardLink packet is received, the first data byte with its K-flag deasserted is interpreted as the packet local address. An acceptance filter is applied to this address in order to decide whether the packet shall be accepted (stored in memory) or ignored (read from the WizardLink codec but never written to external memory).

The filter consists of 2 bytes: an address code and a mask, which are configured in the Address Register. The WizardLink packet address is compared with the address code after applying the mask: those bits set to 1b in the corresponding position of the mask are ignored. The following condition must be fulfilled in order to accept a WizardLink packet:

$$PKTADDR \text{ and not } ADDR.MASK = ADDR.CODE \text{ and not } ADDR.MASK$$

Setting all bits of the mask to 1b disables the filter, and all packet addresses are therefore accepted.

91.4.7 Programmable commands

GRWIZL features a built-in mechanism to transmit and receive custom commands without the need of setting up descriptors. These commands can be configured using the AHB registers, both their data bytes and associated K-flags. The transmission and reception is controlled via AHB registers as well. This mechanism can be used to map control commands that are typically not part of the data payload of a packet. Examples of this are commands used during link establishment, packet acknowledgment, timestamps and flow control.

The first step is to configure the commands relevant to a given application. The IP supports up to 8 programmable commands, although the user does not need to initialize all of them. After configuring their data and K-flag patterns (similarly to how it is done for the IDLE control word, which is mandatory), GRWIZL is ready to transmit the commands and filter them in reception.

The transmission of a command can be triggered by setting the corresponding bit in the Programmable Command TX Register. The TX Request field (PROGTX.REQ) consists of as many bits as programmable commands. The user may set one or more bits simultaneously. If multiple bits are set, commands in lower indices take precedence and are transmitted first. Once a command is transmitted, the corresponding bit is cleared by the IP.

The reception of custom commands works similarly. After configuring the commands, the user shall enable which commands can be automatically detected and filtered by the IP. This is done via the Programmable Command RX Enable Register. The Enable field (PROGEN.EN) has one bit per programmable command. Only commands whose enable bit is set are filtered by the IP. The status can be monitored by checking the Programmable Command RX Register. The flags in the RX.STS field are cleared when written to 1b. Note that when a custom command is received, it is removed from the data stream and therefore not stored in external memory.

Optionally, the transmission and reception of programmable commands can trigger an interrupt. The Programmable Command IRQ Enable Register can be used to specify which commands shall generate interrupts in both transmission and reception. Additionally, the user can separately enable transmission and reception interrupts by means of the TXC and RXC fields in the Interrupt Control Register, respectively.

The table below shows the correspondence between the indices in the control registers (PROGTX, PROGRX, PROGEN and PROGIRQ) and the registers used to configure each programmable command:

Table 1786. Correspondence between indices in the control registers and programmable command registers

Index in control registers (PROGTX, PROGRX, PROGEN, PROGIRQ)	Command data register	Command K-flags register field
0	PROGCMD0 (AHB offset 0x40)	PROGKF.KF0
1	PROGCMD1 (AHB offset 0x44)	PROGKF.KF1
2	PROGCMD2 (AHB offset 0x48)	PROGKF.KF2
3	PROGCMD3 (AHB offset 0x4C)	PROGKF.KF3
4	PROGCMD4 (AHB offset 0x50)	PROGKF.KF4
5	PROGCMD5 (AHB offset 0x54)	PROGKF.KF5
6	PROGCMD6 (AHB offset 0x58)	PROGKF.KF6
7	PROGCMD7 (AHB offset 0x5C)	PROGKF.KF7

91.4.8 Bandwidth throttling (TX only)

There may be cases where the remote node cannot process the incoming data at the rated link speed, which is typically in the range of gigabits per second. To alleviate this situation the Idle Counter Register can be used to interleave IDLE control words between two consecutive data words, thus throttling the bandwidth in transmission.

When set to a value other than 0, this register defines the number of IDLE control words inserted between data words. The receiver in the remote end is expected to filter these IDLE words so that the actual data can be handled at the desired rate. When the register is set to 0, this mechanism is disabled and IDLE control words are only sent when the IP does not have anything else to transmit.

91.4.9 Deactivating the TX and RX channels

A prerequisite for transmitting and receiving packets is to set TX enable (TE) and RX enable (RE) in the DMA control register, respectively. However, it is important to point out that these control bits are only checked when the FSM is in idle state, meaning that once a transmission or reception starts, the

DMA engine will attempt to finish it even if the corresponding channel is disabled in the middle of the transfer.

It is therefore essential to monitor the DMA Status Register in order to know whether the TX and RX channels are effectively switched off before the IP may be safely reconfigured. Two bits are available for this purpose: TX Active (TA) and RX Active (RA). Note that these status bits will be set to 1 if the corresponding channel (TX or RX) is enabled, even if the FSM is idle.

Therefore, the procedure to safely change the configuration of the IP is:

- Disable the TX and/or RX channel by setting the Transmitter Stop (TS) or Receiver Stop (RS) bits in the DMA Control Register.
- Monitor the TA and/or RA bit of the DMA Status Register until cleared.
- Reconfigure the IP, including the address and content of the descriptor tables.
- Re-enable the TX and/or RX channel by setting the Transmitter Enable (TE) / Receiver Enable (RE) bits in the DMA Control Register.

91.5 External FIFO interfaces

GRWIZL can implement an external FIFO interface to directly expose the WizardLink codec data to the ports of the IP. This functionality is enabled via the VHDL generic *numextwc*. When set to 1, this FIFO interface allows the user to directly access the data interface of the WizardLink codec inside GRWIZL, thus bypassing the DMA engine altogether.

By default, this functionality is disabled and the codec is managed by the DMA layer by means of hardware descriptors. To turn it on, the Bypass DMA bit shall be set in the WizardLink Codec Control Register (CCTRL.BD). The user may dynamically configure this feature, meaning that the DMA bypass mode can be turned off and on in run time.

To reconfigure this feature in run time it is always recommended to keep both DMA transmitter and receiver off while switching in order to guarantee that ongoing packets are handled properly.

The transmission and reception of programmable commands is automatically disabled when the DMA engine is bypassed, which means that all commands and data words except for the IDLE command must be handled externally. IDLE commands are still inserted and extracted by the internal WizardLink codec in order to keep the link active.

This feature is identical to the one present in the SpaceFibre IP, so please refer to the GRSPFI section of this user's manual for further information regarding timing of the FIFO signals.

91.6 AMBA interfaces

This section explains the AMBA interfaces present in the GRWIZL codec, as well as the AMBA interrupts that the IP may generate, and how to configure them.

91.6.1 Bus master interface

The DMA engine has a generic bus master interface to fetch and store descriptors and WizardLink packets from/to external memory. The generic approach is chosen for maximum flexibility, so that the user can connect a wrapper adapting the bus master to any custom protocol. Frontgrade Gaisler provides wrappers for AHB 2.0 and AXI4.

Both transmitter and receiver share the bus master interface. The receiver has priority over the transmitter when it comes to accessing the bus, in order not to delay the reception of packets and thus preventing potential loss of data due to buffer overrun.

The size of the read or write access depends on the operation being performed:

- **Fetching a descriptor**: the descriptor size is the same for both transmission and reception. The IP will initiate a read access of 16 bytes to get a complete descriptor.

- **Storing the data bytes of a WizardLink packet** (RX): when the packet is being read from the codec by the DMA engine, a write access will be performed over the bus master interface as soon as there are at least 128 bytes available or the end of the packet has been detected. This means that the size of the write access may vary between 1 and 128 bytes.
- **Storing the K-flags of a WizardLink packet** (RX): after writing a block of up to 128 bytes of data to memory, the core stores their corresponding K-flags. Since each data byte is associated with a single K-flag bit, the write access consists of up to 16 bytes (128 K-flags).
- **Fetching the data bytes of a WizardLink packet** (TX): the data is read from memory in blocks of up to 128 bytes. If there are less than 128 bytes pending, then the size is set to the remaining bytes. Therefore the size of the read burst may vary between 1 and 128 bytes.
- **Fetching the K-flags of a WizardLink packet** (TX): in this case the core reads the flags in a fixed read burst of 16 bytes (128 K-flags, one per data byte). If less than 128 data bytes shall be transmitted, the K-flags read in excess are simply discarded by the core.
- **Updating a descriptor**: once the transmission or reception finishes, the descriptor is updated with the status of the transaction. However, only the first word of the descriptor is overwritten: that implies a fixed write access of 4 bytes for both transmission and reception.

91.6.2 AHB slave interface

GRWIZL has a single AHB slave interface to access the registers of the core. These registers are used to configure and control the IP. The size of write and read accesses over the AHB interface is always 32 bits.

The IP has 2 major clock domains: the AMBA clock, used by the DMA engine and the AMBA interface, and the WizardLink core clock, which relates to the internal codec and its interface with the DMA FIFOs. Therefore, the internal registers are split into both clock domains. All accesses to registers implemented in the WizardLink core clock domain are properly handled by means of cross domain crossing synchronization techniques.

The register address offsets can help determine if the register is implemented in the AMBA clock domain or in the WizardLink clock domain:

- 0x00 - 0x7F: registers implemented in the WizardLink clock domain. These registers are closely related to the control and status of the codec, including the mapping for control characters and words defined for the current application (comma, EOP, EEP, IDLE command, etc.).
- 0x80 - 0xFF: registers implemented in the AMBA clock domain. These registers are related to the DMA channel, interrupt handling and external memory management (i.e. descriptor tables).

Due to the internal cross domain crossing techniques, there will be a delay when accessing registers implemented in the WizardLink clock domain; waitstates are inserted in order to delay the reply from the internal core to guarantee the integrity of the signals.

91.6.3 AMBA interrupts

GRWIZL triggers AMBA interrupts upon certain events. The IP is highly configurable when it comes to selecting which events shall produce an interrupt. This section describes how the interrupts are handled and monitored.

The DMA Control Register shall be used to configure 3 types of interrupts:

- WizardLink packet transmitted.
- WizardLink packet received.
- Error while reading or writing over the bus master interface of the DMA channel.
- Link event: overrun, RX error counter overflow
- Transmission of a programmable command

- Reception of a programmable command

In addition to the control bits above, the TX and RX descriptors include an Interrupt Enable bit used to mask / unmask the generation of interrupts only for the current packet. This means that a TX / RX interrupt is only generated if enabled in both the DMA Control Register and the TX / RX descriptor. This does not apply to the other interrupts, as those are solely enabled in the DMA Control Register.

The user can read the DMA Status Register to know which event has caused an interrupt. In case it was a bus master error, further information is provided: which descriptor was being processed as well as a 4-bit code to identify when the bus error occurred (fetching the descriptor, storing the packet, etc.). Please refer to the chapter covering the Register definition to learn how these fields are encoded.

91.7 Implementation

91.7.1 Reset

The core changes its reset behavior depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

GRWIZL will feature a synchronous reset if the parameter *grlib_async_reset_enable* is not set in the GRLIB configuration package. Otherwise, it will implement an asynchronous reset if *grlib_async_reset_enable* is set.

91.7.2 Endianness

The core automatically changes its endianness behavior depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for both big-endian and little-endian systems.

91.7.3 Clocking scheme

As explained throughout this section, there are up to 4 clock domains in the IP. These paragraphs describe their use and how they shall be generated and connected.

- AMBA clock (clk): the DMA and AMBA layers of the IP operate in this clock domain. It shall be connected to the AMBA clock of the system where the IP is instantiated.
- WizardLink clock (wizl_clk): used by the core logic, including the WizardLink codec and the interface between the DMA FIFOs and the codec. Some AHB registers are also implemented in this domain. This clock is normally generated from the SerDes output transmitter clock by using a Clock Conditioning Circuit. The target frequency shall be the HSSL line rate divided by 40 (i.e. it assumes a 40-bit SerDes interface). This is dependent on the specific SerDes device used. Example: if the link operates at 2.5 Gbps, the target frequency shall always be 62.5 MHz. If the SerDes transmitter clock is 125 MHz, then the multiplication factor of the CCC shall be 0.5.
- WizardLink transmitter clock (wizl_txclk): this clock is used by the transmitter logic within the codec if using a narrow SerDes interface (16/20 bits; both generics *use_sep_txclk* and *sel_16_20_bit_mode* set to 1). The target frequency shall be the HSSL line rate divided by 20 (i.e. a 20-bit SerDes interface, twice the frequency of the core clock). Is is normally generated from the SerDes output transmitter clock. In the example above, the frequency of this clock shall be 125 MHz and could be generated from the SerDes TX clock with a multiplication factor of 1.
- WizardLink RX recovered clock (wizli.se_rx_clk): if the data received from the SerDes is not synchronous to the core clock, the SerDes recovered RX clock shall be connected to this input. If the data is already synchronous, for instance because the SerDes comprises its own elastic buffer, this clock shall be connected to the core clock directly.

Any exchange of signals between clock domains, especially between the AMBA and core clocks, is protected by clock domain crossing techniques in order to avoid metastability problems in the IP.

Please note that the core clock and the AMBA clock have separate reset signals; the user shall ensure that these resets are properly synchronized with their respective clocks if a synchronous reset scheme is chosen.

91.7.4 Relation between GRWIZL and GRHSSL

GRHSSL is a High Speed Serial Link controller providing support for both SpaceFibre and WizardLink. The IP can be configured to implement either or both codecs by means of VHDL generics. When both protocols are implemented, the user shall select which codec is active at a time via AHB registers. The selected core will be in charge of driving the SerDes and bus master interfaces.

GRWIZL internally uses the GRHSSL IP and configures it so that only WizardLink is instantiated. Generics relevant to WizardLink are exposed in the top-level entity, whereas those related to SpaceFibre are tied to predefined values in order to guarantee that the SpaceFibre layer is not present in the design.

91.8 Registers

The core is programmed through registers mapped into AHB address space. Only 32-bit single accesses to the registers are supported. Registers within the range 0x00-0x7F are implemented in the WizardLink clock domain, whereas the range 0x80-0xFF is related to the AMBA/DMA layer.

Table 1787. GRWIZL registers

AHB address offset	Register
0x00	Codec Capabilities Register
0x04	Codec Control Register
0x08	Codec Status Register
0x0C	Control Characters Configuration Register
0x10	Idle Control Word Configuration Register
0x14	Idle Counter Register
0x18	Address Register
0x1C	RX Maximum Length Register
0x20	Programmable Commands TX Register
0x24	Programmable Commands RX Register
0x28	Programmable Commands RX Enable Register
0x2C	Programmable Commands IRQ Enable Register
0x30	Programmable Commands K-flags Register
0x40, 0x44, 0x48... 0x5C	Programmable Command Registers
0x80	DMA Capability Register
0x84	DMA Control Register
0x88	DMA Status Register
0x8C	WizardLink Enable Register
0x90	Interrupt Control Register
0xA0	TX Descriptor Table Address Register
0xA4	RX Descriptor Table Address Register
0xA8	Descriptor Control Register
0xAC	Statistics Register

91.8.1 Codec Capabilities Register

Table 1788. 0x00 - CCAP - Codec Capabilities Register

31	16
Reserved	
0x0000	
r	

15	14	13	9	8	4	3	2	1	0
Reserved		TXB		RXB		EFC	BM	IED	SCK
00		-		-		-	-	-	-
r		r		r		r	r	r	r

13: 9 Depth of the codec TX buffer (TXB). Encoded as depth (address bits) - 1. Value determined by the VHDL generic *depth_tx_buf*.

8: 4 Depth of the codec RX buffer (RXB). Encoded as depth (address bits) - 1. Value determined by the VHDL generic *depth_rx_buf*.

3 Number of External WizardLink FIFO channels (EFC). Value determined by the VHDL generic *numextwc*.

2 16/20 Bit Mode Selector (BM). If set, the SerDes interface is 16+2 bit (without 8b10b) or 20 bit (with 8b10b) wide instead of 36/40 bit. Determined by the VHDL generic *sel_16_20_bit_mode*.

1 Internal 8b10b encoder/decoder (IED). If set, internal 8B10B encoding and decoding is activated. Determined by the VHDL generic *use_8b10b*.

0 Use separate clock for TX (SCK). If set, the SerDes transmission clock is decoupled from the WizardLink core clock. An additional transmit buffer is instantiated for this reason. Determined by the VHDL generic *use_sep_txclk*.

91.8.2 Codec Control Register

Table 1789. 0x04 - CCTRL - Codec Control Register

31	16
Reserved	
0x0000	
r	

15	4	3	2	1	0
Reserved		BD	LB	TE	RE
0x000		0	0	0	0
r		rw	rw	rw	rw

3 Bypass DMA engine (BD). When this bit is set, WizardLink data are directly exposed via the external FIFO ports of the IP (EXTWCI, EXTWCO), thus bypassing the DMA engine. If not set, WizardLink data are handled by the DMA engine by means of hardware descriptors. This bit is only available if the generic *numextwc* is set to 1.

2 Loop-back mode (LB). The WizardLink codec has an internal loop-back feature that can be enabled / disabled with this bit.

1 SerDes Transmit Enable (TE). If set, the codec reads the words passed by the DMA layer and transmits them over the high-speed serial link. This control bit is also visible as an output of the IP and shall be connected to the Transmission Enable port of the SerDes, if it exists.

0 SerDes Receive Enable (RE). If set, the codec parses the words received over the high-speed serial link to extract IDLE and RX error words from the data stream, while the rest is passed on to the DMA layer. This control bit is also visible as an output of the IP and shall be connected to the Reception Enable port of the SerDes, if it exists.

91.8.3 Codec Status Register

Table 1790. 0x08 - CSTS - Codec Status Register

31	16
Reserved	
0x0000	
r	

15	10	9	8	7	0
Reserved		OR	REO	REC	
0x00		0	0	0x00	
r		wc	wc	r	

9 RX Overrun (OR). This bit is set when the codec receives a valid word but the FIFO is full, so the new data is lost. Cleared when written to 1b.

8 RX Error Count Overflow (REO). Set when the RXERR word counter has overflown. Cleared when written to 1b.

7: 0 RX Error Word Counter (REC). 8-bit wide counter for received RXERR words. These words are extracted by the codec and not passed to the DMA engine.

91.8.4 Control Characters Configuration Register

Table 1791. 0x0C - CCHAR - Control Characters Configuration Register

31	28	27	24	23	16
IDLE		Reserved		EEP	
0x0		0x0		0x00	
rw		r		rw	

15	8	7	0
EOP		COM	
0x00		0x00	
rw		rw	

31: 28 K-flags pattern for IDLE Control Words (IDLE). Together with the data pattern defined in the Idle Control Word Configuration Register, this field configures the Idle Control Words handled by the codec. The MSB bit is associated with the first byte transmitted.

23: 16 Error End of Packet marker (EEP). The WizardLink codec detects an EEP marker when any data byte matches this pattern and the corresponding K-flag is asserted. If set to the same value as the EOP, the EEP detection is effectively disabled.

15: 8 End of Packet marker (EOP). The WizardLink codec detects an EOP marker when any data byte matches this pattern and the corresponding K-flag is asserted.

7: 0 Synchronization comma (COM). The WizardLink codec automatically detects the synchronization comma and realigns the received word so that the comma appears on the least significant byte. Note that the 8b10b encoding scheme only allows certain values for the comma.

91.8.5 Idle Control Word Configuration Register

Table 1792. 0x10 - IDLE - Idle Control Word Configuration Register

31	16
DATA	
0x0000	
rw	

15	0
DATA	
0x0000	
rw	

31: 0 Data pattern for Idle Control Words (DATA). It defines the IDLE control words in conjunction with the K-flags pattern defined in the K-flags Configuration Register. In transmission, Idle Control Words are inserted to keep the link active if the codec does not have any other data to transmit. In reception, these control words are simply discarded, i.e. not passed to the DMA engine. The most significant byte is transmitted first.

91.8.6 Idle Counter Configuration Register

Table 1793. 0x14 - IDLECNT - Idle Counter Configuration Register

31	16
Reserved	
0x0000	
r	

15	0
CNT	
0x0000	
rw	

15: 0 IDLE Counter (CNT). Number of IDLE Control Words inserted between two consecutive data words. Used to throttle the bandwidth in transmission. Disabled when set to 0.

91.8.7 Address Register

Table 1794. 0x018 - ADDR - Address Register

31	16
Reserved	
0x0000	
r	

15 8	7 0
MASK	CODE
0x00	0x00
rw	rw

15: 8 Address Mask (MASK). Mask used in the address filter. When a bit of the mask is set, the corresponding bit of the WizardLink packet address is not compared with the Address Code.

7: 0 Address Code (CODE). Address compared with the masked WizardLink address byte. The address filter is applied to the first byte of the packet received whose K-flag is deasserted (i.e. not a control character).

91.8.8 RX Max Length Register

Table 1795. 0x1C - RXMAXLEN - RX Max Length Register

31 25	24
Reserved	MAXLGTH
0x00	0x000000
r	rw

2 1 0	
MAXLGTH	Reserved
0x000000	00
rw	r

24: 2 RX Maximum Length (MAXLGTH). Maximum number of words accepted per WizardLink packet. When the length of the packet exceeds this value, the rest of the packet is spilled and the Truncated bit is set when updating the RX descriptor. Note that control words internally processed by the codec such as IDLE and RX Error do not count towards this limit.

91.8.9 Programmable Commands TX Register

Table 1796. 0x20 - PROGTX - Programmable Commands TX Register

31 16
Reserved
0x0000
r

15 8	7 0
Reserved	REQ
0x00	0x00
r	rw

7: 0 Programmable command transmission request (REQ). When a bit in this field is set, the corresponding programmable command is transmitted by the core as soon as the transmitter is idle. The transmission of several commands can be requested simultaneously; if so, commands associated with lower indices are transmitted first. Once a command is transmitted, the corresponding bit in this field is automatically cleared.

91.8.10 Programmable Commands RX Register

Table 1797. 0x24 - PROGRX - Programmable Commands RX Register

31 16
Reserved
0x0000
r

15 8	7 0
Reserved	STS
0x00	0x00
r	wc

7: 0 Programmable command reception status (STS). The bits in this register are set once the corresponding programmable command has been received by the codec and removed from the data stream. The core only compares commands whose RX enable bit is set in the Programmable Commands RX Enable Register.

91.8.11 Programmable Commands RX Enable Register

Table 1798. 0x28 - PROGEN - Programmable Commands RX Enable Register

31	16
Reserved	
0x0000	
r	

15	8	7	0
Reserved		EN	
0x00		0x00	
r		rw	

7: 0 Programmable command reception enable (EN). When set, each word received by the core is compared with the associated programmable command. If a match is found, the corresponding bit in the Programmable Command RX Register is set and the word is filtered away.

91.8.12 Programmable Commands IRQ Enable Register

Table 1799. 0x2C - PROGIRQ - Programmable Commands IRQ Enable Register

31	16
Reserved	
0x0000	
r	

15	8	7	0
Reserved		IRQ	
0x00		0x00	
r		rw	

7: 0 Programmable command IRQ enable (EN). When a bit in this register is set, the core will request the generation of an interrupt to the DMA layer when the corresponding programmable command has been transmitted or received by the core. TX and RX interrupts can be masked separately in the Interrupt Control Register.

91.8.13 Programmable Commands K-flags Register

Table 1800. 0x30 - PROGKF - Programmable Commands K-flags Register

31	28	27	24	23	20	19	16
KF7		KF6		KF5		KF4	
0x0		0x0		0x0		0x0	
rw		rw		rw		rw	

15	12	11	8	7	4	3	0
KF3		KF2		KF1		KF0	
0x0		0x0		0x0		0x0	
rw		rw		rw		rw	

31: 28 K-flags pattern for the programmable command 7 (KF7). It defines the command together with the data pattern specified in the Programmable Command 7 Register. The MSB bit corresponds to the byte transmitted first.

27: 24 K-flags pattern for the programmable command 6 (KF6). It defines the command together with the data pattern specified in the Programmable Command 6 Register. The MSB bit corresponds to the byte transmitted first.

23: 20 K-flags pattern for the programmable command 5 (KF5). It defines the command together with the data pattern specified in the Programmable Command 5 Register. The MSB bit corresponds to the byte transmitted first.

19: 16 K-flags pattern for the programmable command 4 (KF4). It defines the command together with the data pattern specified in the Programmable Command 4 Register. The MSB bit corresponds to the byte transmitted first.

15: 12 K-flags pattern for the programmable command 3 (KF3). It defines the command together with the data pattern specified in the Programmable Command 3 Register. The MSB bit corresponds to the byte transmitted first.

11: 8 K-flags pattern for the programmable command 2 (KF2). It defines the command together with the data pattern specified in the Programmable Command 2 Register. The MSB bit corresponds to the byte transmitted first.

7: 4 K-flags pattern for the programmable command 1 (KF1). It defines the command together with the data pattern specified in the Programmable Command 1 Register. The MSB bit corresponds to the byte transmitted first.

3: 0 K-flags pattern for the programmable command 0 (KF0). It defines the command together with the data pattern specified in the Programmable Command 0 Register. The MSB bit corresponds to the byte transmitted first.

91.8.14 Programmable Command Registers

Table 1801. 0x40, 0x44... 0x5C - PROGCMD - Programmable Command Registers

31	16
CMD	
0x0000	
rw	

15	0
CMD	
0x0000	
rw	

31: 0 Data pattern for the programmable command (CMD). This register configures a programmable command in conjunction with the corresponding field in the Programmable Command K-flag Register. Commands 0 - 7 are associated with AHB offsets 0x40 - 0x5C, respectively. The most significant byte is transmitted first.

91.8.15 DMA Capabilities Register

Table 1802. 0x80 - DMACAP - DMA Capabilities Register

31	30 16
WIZ	Reserved
1	0x0000
r	r

15 4	3 2	1 0
Reserved	NTXD	NRXD
0x000	-	-
r	r	r

31 WizardLink protocol implemented (WIZ). Read-only bit to indicate that GRWIZL is present in the design.

3: 2 Number of TX descriptors (NTXD). It shows the size of the TX descriptor table. Value determined by the VHDL generic *num_txdesc*. Encoding:
- 00b: 64 descriptors
- 01b: 128 descriptors
- 10b: 256 descriptors
- 11b: 512 descriptors

1: 0 Number of RX descriptors (NRXD). It shows the size of the RX descriptor table. Value determined by the VHDL generic *num_rxdesc*. Encoding:
- 00b: 128 descriptors
- 01b: 256 descriptors
- 10b: 512 descriptors
- 11b: 1024 descriptors

91.8.16 DMA Control Register

Table 1803. 0x84 - DMACTRL - DMA Control Register

31	16
Reserved	
0x0000	
r	

15	4	3	2	1	0
Reserved		TS	RS	TE	RE
0x000		0	0	0	0
r		w	w	w	w

3 Transmitter Stop (TS). If set to 1b, GRWIZL will disable the transmitter as soon as the Transmit FSM enters IDLE state. Any pending TX enable request is cleared.

2 Receiver Stop (RS). If set to 1b, GRWIZL will disable the receiver as soon as the Receive FSM enters IDLE state. Any pending RX enable request is cleared.

1 Transmitter Enable (TE). If set to 1b, GRWIZL will enable the transmitter as soon as the Transmit FSM enters IDLE state. Any pending TX stop request is cleared.

0 Receiver Enable (RE). If set to 1b, GRWIZL will enable the receiver as soon as the Receive FSM enters IDLE state. Any pending RX stop request is cleared.

91.8.17 DMA Status Register

Table 1804. 0x88 - DMASTS - DMA Status Register

31	24	23	22	21	20	19	18	17
Reserved		TXC	RXC	LE	TO	Reserved		BEDESC
0x00		0	0	0	0	00		0x000
r		wc	wc	wc	wc	r		wc

8	7	4	3	2	1	0
BEDESC	BECODE		TX	RX	TA	RA
0x000	0x0		0	0	0	0
wc	wc		wc	wc	r	r

23 Programmable command TX status (TXC). If set, the codec has flagged the transmission of a programmable command whose individual IRQ bit is also set. The DMA layer generates an interrupt if enabled in the Interrupt Control Register.

22 Programmable command RX status (RXC). If set, the codec has flagged the reception of a programmable command whose individual IRQ bit is also set. The DMA layer generates an interrupt if enabled in the Interrupt Control Register.

21 Link event (LE). If set, the codec has detected an event in the WizardLink communication, such as RX overrun or error counter overflow. The DMA layer generates an interrupt if enabled in the Interrupt Control Register.

20 AHB Register Timeout (TO). Set when the DMA engine has reached the internal timeout (256 AMBA clock cycles) while waiting for a reply when accessing the AHB registers implemented in the codec clock domain.

17: 8 Bus Master Error - Descriptor (BEDESC). This field identifies the descriptor selected when an error occurred during an access through the bus master interface. Cleared when the BECODE field is written to a value other than 0x0.

7: 4 Bus Master Error - Code (BECODE). This field specifies what type of error occurred when accessing the bus. Writing a value other than 0x0 to this field clears the content of all the bus master error-related fields: BEDESC and BECODE. Encoding:

- 0x0: No errors
- 0x1: Fetching RX descriptor
- 0x2: Storing data bytes of a WizardLink packet (RX)
- 0x3: Storing K-flags of a WizardLink packet (RX)
- 0x4: Updating RX descriptor
- 0x5: Fetching TX descriptor
- 0x6: Fetching K-flags of a WizardLink packet (TX)

0x7: Fetching data bytes of a WizardLink packet (TX)
0x8: Updating TX descriptor

3 Packet transmitted (TX). If set to 1b, the DMA engine has fetched and passed a complete WizardLink packet on to the codec, and the associated descriptor has been updated. Cleared to 0b when written.

2 Packet received (RX). If set to 1b, the DMA engine has read a complete WizardLink packet from the codec and written it to external memory, and the associated descriptor has been updated. Cleared to 0b when written.

1 Transmitter Active (TA). If set to 1b, the transmitter FSM is active. This may imply that the Transmitter Enable bit in the DMA Control Register is set, or that the DMA engine is still completing an ongoing operation before being disabled.

0 Receiver Active (RA). If set to 1b, the receiver FSM is active. This may imply that the Receiver Enable bit in the DMA Control Register is set, or that the DMA engine is still completing an ongoing operation before being disabled.

91.8.18 WizardLink Enable Register

Table 1805. 0x8C - WIZLEN - WizardLink Enable Register

31	16
Reserved	
0x0000	
r	

15	1	0
Reserved		EN
0x0000		-
r		rw

0 WizardLink Enable (EN). This bit is used to enable the WizardLink mode of operation if GRWIZL is implemented together with other protocols as part of the GRHSSL IP. If WizardLink is the only protocol implemented, this bit will read as 1b and writing will have no effect.

91.8.19 Interrupt Control Register

Table 1806. 0x90 - IRQ - Interrupt Control Register

31	16
Reserved	
0x0000	
r	

15	6	5	4	3	2	1	0
Reserved		TXC	RXC	LE	BME	TX	RX
0x000		0	0	0	0	0	0
r		rw	rw	rw	rw	rw	rw

5 Programmable Command Transmission Interrupt Enable (TXC). If set, GRWIZL will generate an AMBA interrupt upon transmitting a programmable command with its individual IRQ enable bit set.

4 Programmable Command Reception Interrupt Enable (RXC). If set, GRWIZL will generate an AMBA interrupt upon receiving a word matching an enabled programmable command with its individual IRQ enable bit set.

3 Link Event Interrupt Enable (LE). If set to 1b, GRWIZL will generate an AMBA interrupt when the core detects an event in the WizardLink communication, such as RX overrun or error counter overflow.

2 Bus Master Error Interrupt Enable (BEI). If set to 1b, GRWIZL will generate an AMBA interrupt when an error is detected during an access over the bus master interface.

1 Transmit Interrupt Enable (TI). If set to 1b, GRWIZL will generate AMBA interrupts after transmitting a packet and updating the corresponding descriptor. Note that the Interrupt Enable bit in the descriptor associated with the packet shall also be set to 1b.

0 Receive Interrupt Enable (RI). If set to 1b, GRWIZL will generate AMBA interrupts after storing a packet and updating the corresponding descriptor. Note that the Interrupt Enable bit in the descriptor associated with the packet shall also be set to 1b.

91.8.20 TX Descriptor Table Address Register

Table 1807. 0xA0 - TXDADDR - TX Descriptor Table Address Register

31	x+1
TXADDR	
All 0b	
rw	

x	4	3	0
TXDESCSEL		Reserved	
All 0b		0x0	
rw		r	

31: x+1 TX Descriptor Table Address (TXADDR). Sets the base address of the descriptor table. The number of bits of this field depends on the size of the DMA transmit descriptor table, determined by the generic *num_txdesc*. The value of x is calculated as follows: $x = log_2(num_txdesc) + 3$.

x: 4 TX Descriptor Selector (TXDESCSEL). Offset within the descriptor table. Shows which descriptor is currently used by the core. For each new descriptor read, the selector will increase by 16 and eventually wrap to zero again. The number of bits of this field depends on the size of the DMA transmit descriptor table, determined by the generic *num_txdesc*. The value of x is calculated as follows: $x = log_2(num_txdesc) + 3$.

91.8.21 RX Descriptor Table Address Register

Table 1808. 0xA4 - RXDADDR - RX Descriptor Table Address Register

31	x+1
RXADDR	
All 0b	
rw	

x	4	3	0
RXDESCSEL		Reserved	
All 0b		0x0	
rw		r	

31: x+1 RX Descriptor Table Address (RXADDR). Sets the base address of the descriptor table. The number of bits of this field depends on the size of the DMA receive descriptor table, determined by the generic *num_rxdesc*. The value of x is calculated as follows: $x = log_2(num_rxdesc) + 3$.

x: 4 RX Descriptor Selector (RXDESCSEL). Offset within the descriptor table. Shows which descriptor is currently used by the core. For each new descriptor read, the selector will increase by 16 and eventually wrap to zero again. The number of bits of this field depends on the size of the DMA receive descriptor table, determined by the generic *num_rxdesc*. The value of x is calculated as follows: $x = log_2(num_rxdesc) + 3$.

91.8.22 Descriptor Control Register

Table 1809. 0xA8 - DCTRL - Descriptor Control Register

31	30	29	16
TDA	RDA	Reserved	
0	0	0x0000	
r	r	r	

15	4	3	2	1	0
Reserved		CTD	CRD	NTD	NRD
0x000		0	0	0	0
r		w	w	w	w

31 Transmit Descriptor Available (TDA). Read-only bit indicating whether the TX descriptor table contains descriptors yet to be fetched and processed. Cleared when the core finds a disabled descriptor (Descriptor Enable bit set to 0b inside the descriptor) or when a bus master error occurs during transmission.

30 Receive Descriptor Available (RDA). Read-only bit indicating whether the RX descriptor table contains descriptors yet to be fetched and processed. Cleared when the core finds a disabled descriptor (Descriptor Enable bit set to 0b inside the descriptor) or when a bus master error occurs during reception.

3 Clear Transmit Descriptor Table (CTD). Write-only bit that, when set, clears the content of the TX Descriptor Table. It takes effect only when the DMA channel is in IDLE state.

2 Clear Receive Descriptor Table (CRD). Write-only bit that, when set, clears the content of the RX Descriptor Table. It takes effect only when the DMA channel is in IDLE state.

1 New Transmit Descriptor Available (NTD). Write-only bit to indicate that new descriptors have been added to the TX Descriptor table. It takes effect only when the DMA channel is in IDLE state.

0 New Receive Descriptor Available (NRD). Write-only bit to indicate that new descriptors have been added to the RX Descriptor table. It takes effect only when the DMA channel is in IDLE state.

91.8.23 Statistics Register

Table 1810. 0xAC - STAT - Statistics Register

31	16
TX	
0x0000	
wc	

15	0
RX	
0x0000	
wc	

31: 16 Transmit counter (TX). Number of packets transmitted. The counter is reset when writing to this field a value other than 0.

15: 0 Receive counter (RX). Number of packets received. The counter is reset when writing to this field a value other than 0.

91.9 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x0C7 (GRWIZL). For description of vendor and device identifiers see GRLIB IP Library User's Manual.

91.10 Configuration options

Table 1811 shows the configuration options of the core (VHDL generics).

*Table 1811.*Configuration options

Generic name	Function	Allowed range	Default
tech	Technology for the internal memories.	0 - NTECH	inferred (0)
hmindex	AHB master index. Only present in the AHB wrapper.	0 - NAHBMST-1	0
hsindex	AHB slave index.	0 - NAHBSLV-1	0
haddr	Addr field of the AHB bar.	0 - 16#FFF#	0
hmask	Mask field of the AHB bar.	0 - 16#FFF#	16#FFF#
hirq	Interrupt line used by GRWIZL.	0 - NAHBIRQ-1	0
use_8b10b	If set, internal 8B10B encoding and decoding is activated.	0 - 1	1
use_sep_txclk	If set, the SerDes transmission clock is decoupled from the WizardLink core clock. An additional transmit buffer is instantiated for this reason.	0 - 1	0
sel_16_20_bit_mode	If set, the SerDes interface is 16+2 bit (without 8B10B) or 20 bit (with 8B10B) wide instead of 36/40 bit. If set, *use_sep_txclk* must also be set.	0 - 1	0
num_txdesc	Size of the TX descriptor table in number of descriptors.	64, 128, 256, 512	64
num_rxdesc	Size of the RX descriptor table in number of descriptors.	128, 256, 512, 1024	128
depth_dma_fifo	Depth of the DMA FIFOs. Sets the number of entries in the 144-bit transmitter and receiver FIFOs.	16 - 256	32
depth_tx_buf	Log(Depth) of the codec internal TX buffer, expressed as number of address bits.	1 - 32	10
depth_rx_buf	Log(Depth) of the codec internal RX buffer, expressed as number of address bits.	1 - 32	10
use_async_rxrst	Force asynchronous reset in the recovered clock domain, regardless of the general GRLIB settings.	0 - 1	0
ft_core_buf	Enable fault-tolerance against SEU errors for the codec internal TX and RX buffers. Possible options are byte parity protection (ft = 1), TMR registers (ft = 2), SEC-DED BCH (ft = 4) or technology specific protection (ft = 5). If set to 0, no protection is implemented.	0 - 5	0
ft_core_if	Enable fault-tolerance against SEU errors for the clock-domain-crossing FIFOs in the Interface Layer of the codec. The core supports the same configurations as above.	0 - 5	0
ft_dma	Enable fault-tolerance against SEU errors for the TX/RX FIFOs in the DMA layer. The core supports the same configurations as above.	0 - 5	0
scantest	Enable Scan Test support.	0 - 1	0
ahbbits	Width of AHB read/write data buses and maximum access size. Only present in the AHB wrapper.	32, 64, 128	AHBDW
lendian	Select endianness of the system: 0 for big-endian, 1 for little-endian. Only present in the AXI wrapper.	0 - 1	1
numextwc	If set, implement an external FIFO interface that can be used to bypass the DMA engine when transmitting and receiving WizardLink data.	0 - 1	0

91.11 Signal descriptions

Table 1812 shows the interface signals of the core when using the wrapper for AMBA AHB 2.0 (VHDL ports).

Table 1812. Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	System reset	Low
CLK	N/A	Input	AMBA clock	-
WIZL_RSTN	N/A	Input	WizardLink reset	Low
WIZL_CLK	N/A	Input	WizardLink clock	-
WIZL_TXCLK	N/A	Input	Optional WizardLink transmit clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-
WIZLI	RX_CLK	Input	SerDes receive clock	-
	RX_DATA	Input	Receive Data	-
	RX_KFLAGS	Input	Receive K-Flags	-
	RX_SERROR	Input	Receive Error Flags	High
	NO_SIGNAL	Input	SerDes No Signal Flag	High
WIZLO	TX_DATA	Output	Transmit Data	-
	TX_KFLAGS	Output	Transmit K-Flags	-
	TX_EN	Output	SerDes Transmitter Enable Flag	High
	RX_EN	Output	SerDes Receiver Enable Flag	High
	INV_POL	Output	SerDes Invert Polarity Flag	High
	TX_DATA_DBG	Output	Unencoded 32-bit transmit data	-
	TX_KFLAGS_DBG	Output	4-bit transmit K-Flags	-
	RX_DATA_DBG	Output	Unencoded 32-bit receive data	-
	RX_KFLAGS_DBG	Output	4-bit receive K-Flags	-
EXTWCI	TX_DATA	Input	WizardLink transmit data when using the external FIFO interface.	-
	TX_KFLAGS	Input	WizardLink transmit K-flags when using the external FIFO interface.	-
	TX_WEN	Input	WizardLink transmit write enable when using the external FIFO interface.	High
	RX_REN	Input	WizardLink receive read enable when using the external FIFO interface.	High
EXTWCO	TX_FULL	Output	WizardLink transmit full flag when using the external FIFO interface.	High
	RX_DATA	Output	WizardLink receive data when using the external FIFO interface.	-
	RX_KFLAGS	Output	WizardLink receive K-flags when using the external FIFO interface.	-
	RX_VALID	Output	WizardLink receive valid flag when using the external FIFO interface.	High

Table 1812. Signal descriptions

Signal name	Field	Type	Function	Active
MTESTI**	WIZL	Input	Memory BIST input signals to the FIFOs in the WizardLink codec. It consists of the following subarrays: - host_tx: Host TX FIFO - host_rx: Host RX FIFO - if_tx: Interface Layer async TX FIFO - if_rx: Interface Layer async RX FIFO	-
	DMA_TX	Input	Memory BIST input signal to the DMA TX FIFO	-
	DMA_RX	Input	Memory BIST input signal to the DMA RX FIFO	-
MTESTO**	WIZL	Output	Memory BIST output signals from the FIFOs in the WizardLink codec. It consists of the following subarrays: - host_tx: Host TX FIFO - host_rx: Host RX FIFO - if_tx: Interface Layer async TX FIFO - if_rx: Interface Layer async RX FIFO	-
	DMA_TX	Output	Memory BIST output signal from the DMA TX FIFO	-
	DMA_RX	Output	Memory BIST output signal from the DMA RX FIFO	-
MTESTCLK**	N/A	Input	Memory BIST clock	-

* see GRLIB IP Library User's Manual

** not available in FPGA releases

91.12 Library dependencies

Table 1813 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1813.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GRLIB	GENERIC BM PKG	Signals, components	Package including the generic bus master interface and the bridges for converting to AHB and AXI
GAISLER	HSSL	Signals, component	GRWIZL component and signal declarations

91.13 Instantiation

This example shows how the core can be instantiated using the AMBA 2.0 AHB wrapper.

```
library  ieee;
use      ieee.std_logic_1164.all;
library  gaisler;
use      gaisler.hssl.all;

entity example is
   generic (
      tech:       in    integer := 0);
   port (
   );

-- Signal declarations
signal clk       : std_ulogic;
signal rstn      : std_ulogic;
signal wizl_clk  : std_ulogic;
signal wizl_rstn : std_ulogic;

signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
signal ahbmi : ahb_mst_in_type;
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);

signal wizli : grhssl_in_type;
signal wizlo : grhssl_out_type;

------------------------------------------------------------------------------
-- Component instantiation
------------------------------------------------------------------------------

wizl_dut : grwizl_ahb
      generic map (
        tech               => 0,
        hmindex            => 1,
        hsindex            => 7,
        haddr              => 16#A10#,
        hmask              => 16#FFF#,
        hirq               => 1,
        use_8b10b          => 1,
        use_sep_txclk      => 0,
        sel_16_20_bit_mode => 0,
        num_txdesc         => 256,
        num_rxdesc         => 512,
        depth_dma_fifo     => 32,
        depth_tx_buf       => 10,
        depth_rx_buf       => 10,
        use_async_rxrst    => 1)
      port map (
        clk        => clk,
        rstn       => rstn,
        wizl_clk   => hssl_clk,
        wizl_rstn  => hssl_rstn,
        wizl_txclk => '0',                -- Not using separate TX clock
        -- AHB interface
        ahbmi      => ahbmi,
        ahbmo      => ahbmo(1),
        ahbsi      => ahbsi,
        ahbso      => ahbso(7),
        -- Serdes interface
        wizli      => hssli,
        wizlo      => hsslo
        );
```

92 I2C2AHB - I^{2}C to AHB bridge

92.1 Overview

The I^{2}C slave to AHB bridge is an I^{2}C slave that provides a link between the I^{2}C bus and AMBA AHB. The core is compatible with the Philips I^{2}C standard and external pull-up resistors must be supplied for both bus lines.

On the I^{2}C bus the slave acts as an I^{2}C memory device where accesses to the slave are translated to AMBA accesses. The core can translate I^{2}C accesses to AMBA byte, halfword or word accesses. The core makes use of I^{2}C clock stretching but can also be configured to use a special mode without clock stretching in order to support systems where master or physical layer limitations prevent stretching of the I^{2}C clock period.

GRLIB also contains another I^{2}C slave core, without an AHB interface, where the transfer of each individual byte is controlled by software via an APB interface, see the I2CSLV core documentation for more information.

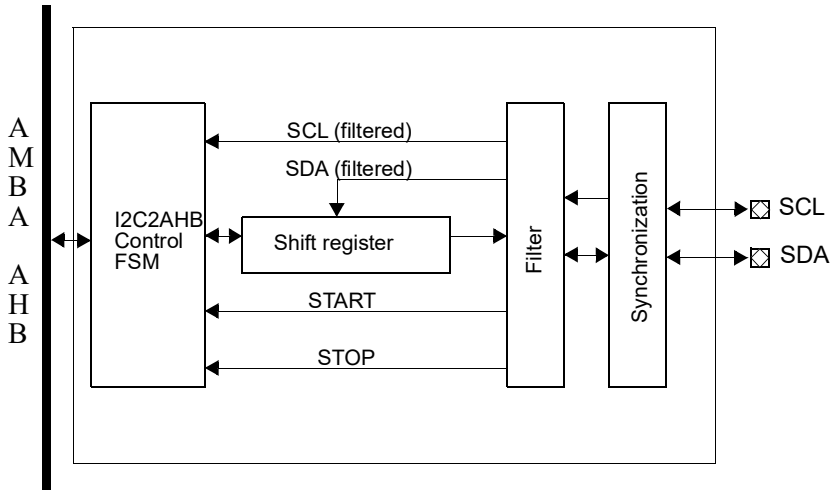


Figure 245. Block diagram, optional APB interface not shown

92.2 Operation

92.2.1 Transmission protocol

The I^{2}C-bus is a simple 2-wire serial multi-master bus with collision detection and arbitration. The bus consists of a serial data line (SDA) and a serial clock line (SCL). The I^{2}C standard defines three transmission speeds; Standard (100 kb/s), Fast (400 kb/s) and High speed (3.4 Mb/s).

A transfer on the I^{2}C-bus begins with a START condition. A START condition is defined as a high to low transition of the SDA line while SCL is high. Transfers end with a STOP condition, defined as a low to high transition of the SDA line while SCL is high. These conditions are always generated by a master. The bus is considered to be busy after the START condition and is free after a certain amount of time following a STOP condition. The bus free time required between a STOP and a START condition is defined in the I^{2}C-bus specification and is dependent on the bus bit rate.

Figure 246 shows a data transfer taking place over the I^{2}C-bus. The master first generates a START condition and then transmits the 7-bit slave address. The bit following the slave address is the R/$\overline{W}$ bit which determines the direction of the data transfer. In this case the R/$\overline{W}$ bit is zero indicating a write operation. After the master has transmitted the address and the R/$\overline{W}$ bit it releases the SDA line. The receiver pulls the SDA line low to acknowledge the transfer. If the receiver does not acknowledge the

transfer, the master may generate a STOP condition to abort the transfer or start a new transfer by generating a repeated START condition.

After the address has been acknowledged the master transmits the data byte. If the R/$\overline{W}$ bit had been set to '1' the master would have acted as a receiver during this phase of the transfer. After the data byte has been transferred the receiver acknowledges the byte and the master generates a STOP condition to complete the transfer.

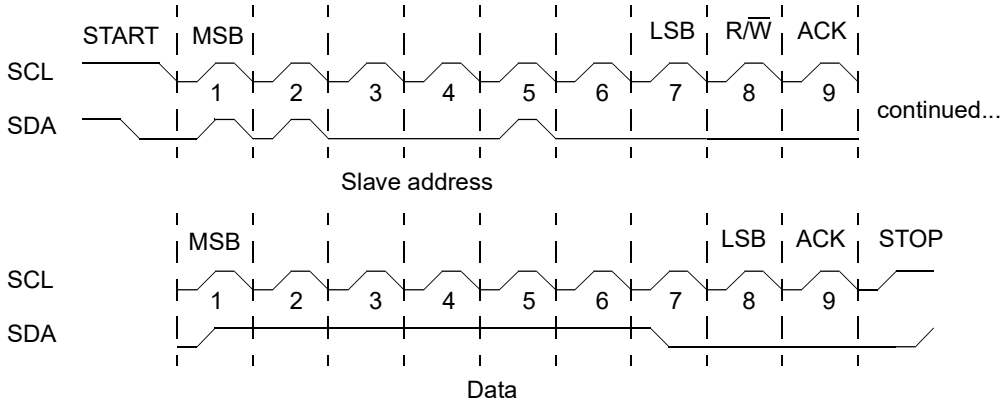


Figure 246. Complete I^{2}C data transfer

If the data bit rate is too high for a slave device or if the slave needs time to process data, it may stretch the clock period by keeping SCL low after the master has driven SCL low. Clock stretching is a configurable parameter of the core (see sections 92.2.4 and 92.2.6).

92.2.2 Slave addressing

The core's I^{2}C addresses are set with VHDL generics at implementation time. If the core has been implemented with the optional APB interface, then the I^{2}C addresses can be changed via registers available via APB.

The core responds to two addresses on the I^{2}C bus. Accesses to the I2C memory address are translated to AMBA AHB accesses and accesses to the I^{2}C configuration address access the core's configuration register.

92.2.3 System clock requirements and sampling

The core samples the incoming I^{2}C SCL clock and does not introduce any additional clock domains into the system. Both the SCL and SDA lines first pass through two stage synchronizers and are then filtered with a low pass filter consisting of four registers.

START and STOP conditions are detected if the SDA line, while SCL is high, is at one value for two system clock cycles, toggles and keeps the new level for two system clock cycles.

The synchronizers and filters constrain the minimum system frequency. The core requires the SCL signal to be stable for at least four system clock cycles before the core accepts the SCL value as the new clock value. The core's reaction to transitions will be additionally delayed since both lines are taken through two-stage synchronizers before they are filtered. Therefore it takes the core over eight system clock cycles to discover a transition on SCL.

92.2.4 Configuration register access

The I^{2}C configuration register is accessed via a separate I^{2}C address (I^{2}C configuration address). The configuration register has the layout shown in table 1814.

*Table 1814.*I2C2AHB configuration register

7	6	5	4	3	2	1	0
Reserved		PROT	MEXC	DMAACT	NACK	HSIZE	

7:6	Reserved, always zero (read only)
5	Memory protection triggered (PROT) - '1' if last AHB access was outside the allowed memory area. Updated after each AMBA access (read only)
4	Memory exception (MEXC) - '1' if core receives AMBA ERROR response. Updated after each AMBA access (read only)
3	DMA active (DMAACT) - '1' if core is currently performing a DMA operation.
2	NACK (NACK) - Use NACK instead of clock stretching. See documentation in section 92.2.6.
1:0	AMBA access size (HSIZE) - Controls the access size that the core will use for AMBA accesses. 0: byte, 1: halfword, 2: word. HSIZE = "11" is illegal.

Reset value: 0x02

Reads from the I^{2}C configuration address will return the current value of the configuration register. Writes to the I^{2}C configuration address will affect the writable bits in the configuration register.

92.2.5 AHB accesses

All AMBA accesses are done in big endian format. The first byte sent to or from the slave is the most significant byte.

To write a word on the AHB bus the following I2C bus sequence should be performed:

1. Generate START condition
2. Send I2C memory address with the R/$\overline{W}$ bit set to '0'.
3. Send four byte AMBA address, the most significant byte is transferred first
4. Send four bytes to write to the specified address
5. If more than four consecutive bytes should be written, continue to send additional bytes, otherwise go to 6.
6. Generate STOP condition

To perform a read access on the AHB bus, the following I2C bus sequence should be performed:

1. Generate START condition
2. Send I2C memory address with the R/$\overline{W}$ bit set to '0'.
3. Send four byte AMBA address, the most significant byte is transferred first
4. Generate (repeated) START condition
5. Send I2C memory address with the R/$\overline{W}$ bit set to '1'.
6. Read the required number of bytes and NACK the last byte
7. Generate stop condition

During consecutive read or write operations, the core will automatically increment the address. The access size (byte, halfword or word) used on AHB is set via the HSIZE field in the I2C2AHB configuration register.

The core always respects the access size specified via the HSIZE field. If a write operation writes fewer bytes than what is required to do an access of the specified HSIZE then the write data will be dropped, no access will be made on AHB. If a read operation reads fewer bytes than what is specified by HSIZE then the remaining read data will be dropped at a START or STOP condition. This means,

for instance, that if HSIZE is “10” (word) the core will perform two word accesses if a master reads one byte, generates a repeated start condition, and reads one more byte. Between these two accesses the address will have been automatically increased, so the fist access will be to address *n* and the second to address *n+4*.

The automatic address increment means that it is possible to write data and then immediately read the data located at the next memory position. As an example, the following sequence will write a word to address 0 and then read a word from address 4:

1. Generate START condition
2. Send I2C memory address with the R/$\overline{W}$ bit set to ‘0’.
3. Send four byte AMBA address, all zero.
4. Send four bytes to write to the specified address
5. Generate (repeated) START condition
6. Send I2C memory address with the R/$\overline{W}$ bit set to ‘1’.
7. Read the required number of bytes and lack the last byte
8. Generate stop condition

The core will not mask any address bits. Therefore it is important that the I^2C master respects AMBA rules when performing halfword and word accesses. A halfword access must be aligned on a two byte address boundary (least significant bit of address must be zero) and a word access must be aligned on a four byte boundary (two least significant address bits must be zero).

The core can be configured to generate interrupt requests when an AHB access is performed if the core is implemented with the APB register interface, see the APB register documentation for details.

92.2.6 Clock stretching or NACK mode

The core has two main modes of operation for AMBA accesses. In one mode the core will use clock stretching while performing an AHB operation and in the other mode the core will not acknowledge bytes (abort the I^2C access) when the core is busy. Clock stretching is the preferred mode of operation. The NACK mode can be used in scenarios where the I^2C master or physical layer does not support clock stretching. The mode to use is selected via the NACK field in the I^2C configuration register.

When clock stretching is enabled (NACK field is ‘0’) the core will stretch the clock when the slave is accessed (via the I2C memory address) and the slave is busy processing a transfer. Clock stretching is also used when a data byte has been transmitted, or received, to keep SCL low until a DMA operation has completed. In the transmit (AMBA read) case SCL is kept low before the rising edge of the first byte. In the receive case (AMBA write) the ACK cycle for the previous byte is stretched.

When clock stretching is disabled (NACK field is ‘1’) the core will never stretch the SCL line. If the core is busy performing DMA when it is addressed, the address will not be acknowledged. If the core performs consecutive writes and the first write operation has not finished the core will now acknowledge the written byte. If the core performs a read operation and the read DMA operation has not finished when the core is supposed to deliver data then the core will go to its idle state and not respond to more accesses until a START condition is generated on the bus. This last part means that the NACK mode is practically unusable in systems where the AMBA access can take longer than one I^2C clock period. This can be compensated by using a very slow I^2C clock.

92.2.7 Memory protection

The core is configured at implementation time to only allow accesses to a specified AHB address range (which can be the full 4 GiB AMBA address range). If the core has been implemented with the optional APB register interface then the address range is soft configurable and the reset value is specified with VHDL generics.

The VHDL generics *ahbaddrh* and *ahbaddrl* define the base address for the allowed area. The VHDL generics *ahbmaskh* and *ahbmaskl* define the size of the area. The generics are used to assign the memory protection area's address and mask in the following way:

Protection address, bits 31:16 (*protaddr[31:16]*): ahbaddrh
Protection address, bits 15:0 (*protaddr[15:0]*): ahbaddrl
Protection mask, bits 31:16 (*protmask[31:16]*): ahbmaskh
Protection mask, bits 15:0 (*protmask[15:0]*): ahbmaskl

Before the core performs an AMBA access it will perform the check:

(((incoming address) xor (protaddr)) and protmask) /= 0x00000000

If the above expression is true (one or several bits in the incoming address differ from the protection address, and the corresponding mask bits are set to '1') then the access is inhibited. As an example, assume that *protaddr* is 0xA0000000 and *protmask* is 0xF0000000. Since *protmask* only has ones in the most significant nibble, the check above can only be triggered for these bits. The address range of allowed accessed will thus be 0xA0000000 - 0xAFFFFFFF.

The memory protection check is performed at the time when the core is to perform the AHB access. It is possible to start a write operation and transmit an illegal address to the core without any errors. If additional bytes are transmitted (so that a HSIZE access can be made) the core will NACK the byte that triggers the AHB access.

For a read operation the core will NACK the I^2C memory address of the first AHB access of the read in case the access would be to restricted memory. If consecutive bytes are read from the core and one of the later accesses lead to restricted memory being accessed, then the core will abort all operations and enter its idle state. In this case junk data will be returned and there is no way for the core to alert the master that memory protection has been triggered.

The core will set the configuration register bit PROT if an access is attempted outside the allowed address range. This bit is updated on each AHB access and will be cleared by an access inside the allowed range. Note that the (optional) APB status register has a PROT field with a slightly different behavior.

92.3 Registers

The core can optionally be implemented with an APB interface that provides registers mapped into APB address space.

Table 1815. I^2C slave registers

APB address offset	Register
0x00	Control register
0x04	Status register
0x08	Protection address register
0x0C	Protection mask register
0x10	I2C slave memory address register
0x14	I2C slave configuration address register

92.3.1 Control Register

*Table 1816.*0x0 - CTRL - Control register

31	2	1	0
RESERVED		IRQEN	EN
0		0	*
r		rw	rw

31 : 2 RESERVED

1 Interrupt enable (IRQEN) - When this bit is set to '1' the core will generate an interrupt each time the DMA field in the status register transitions from '0' to '1'.

0 Core enable (EN) - When this bit is set to '1' the core is enabled and will respond to I2C accesses. Otherwise the core will not react to I2C traffic.

Reset value: Implementation dependent

92.3.2 Status Register

*Table 1817.*0x04 - STAT - Status register

31	3	2	1	0
RESERVED		PROT	WR	DMA

31 : 3 RESERVED

2 Protection triggered (PROT) - Set to '1' if an access has triggered the memory protection. This bit will remain set until cleared by writing '1' to this position. Note that the other fields in this register will be updated on each AHB access while the PROT bit will remain at '1' once set.

1 Write access (WR) - Last AHB access performed was a write access. This bit is read only.

0 Direct Memory Access (DMA) - This bit gets set to '1' each time the core attempts to perform an AHB access. By setting the IRQEN field in the control register this condition can generate an interrupt. This bit can be cleared by software by writing '1' to this position.

92.3.3 Protection Address Register

*Table 1818.*0x08 - PADDR - Protection address register

31	0
PROTADDR	
*	
rw	

31 : 0 Protection address (PROTADDR) - Defines the base address for the memory area where the core is allowed to make accesses.

92.3.4 Protection Mask Register

*Table 1819.*0x0C - PMASK - Protection mask register

31	0
PROTMASK	
*	
rw	

31 : 0 Protection mask (PROTMASK) - Selects which bits in the Protection address register that are used to define the protected memory area.

92.3.5 I2C Slave Memory Address Register

Table 1820. 0x10 - SLVADDR - I2C slave memory address register

31 7	6 0
RESERVED	I2CSLVADDR
0	*
r	rw

31 : 7 RESERVED

6 : 0 I2C slave memory address (I2CSLVADDR) - Address that slave responds to for AHB memory accesses

92.3.6 I2C Slave Configuration Address Register

*Table 1821.*0x14 - SLVCFG - I2C slave configuration address register

31 7	6 0
RESERVED	I2CCFGADDR
0	*
r	rw

31 : 7 RESERVED

6 : 0 I2C slave configuration address (I2CCFGADDR) - Address that slave responds to for configuration register accesses.

92.4 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x00B. For a description of vendor and device identifiers see the GRLIB IP Library User's Manual.

92.5 Implementation

92.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

92.5.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

92.6 Configuration options

Table 1822 shows the configuration options of the core (VHDL generics). Two different top level entites for the core is available. One with the optional APB interface (i2c2ahb_apb) and one without the APB interface (i2c2ahb). The entity without the APB interface has fewer generics as indicated in the table below.

*Table 1822.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST	0
ahbaddrh	Defines bits 31:16 of the address used for the memory protection area	0 - 16#FFFF#	0

*Table 1822.*Configuration options

Generic name	Function	Allowed range	Default
ahbaddrl	Defines bits 15:0 of the address used for the memory protection area	0 - 16#FFFF#	0
ahbmaskh	Defines bits 31:16 of the mask used for the memory protection area	0 - 16#FFFF#	0
ahbmaskl	Defines bits 15:0 of the mask used for the memory protection area	0 - 16#FFFF#	0
resen	Reset value for core enable bit (only available on the i2c2ahb_apb entity).	0 - 1	0
pindex	APB slave index (only available on the i2c2ahb_apb entity).	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR (only available on the i2c2ahb_apb entity).	0 - 16#FFF#	0
pmask	MASK field of the APB BAR (only available on the i2c2ahb_apb entity).	0 - 16#FFF#	16#FFF#
pirq	Interrupt line driven by APB interface (only available on the i2c2ahb_apb entity).	0 - NAHBIRQ-1	0
i2caddr	The slave's (initial) I^2C address. i2caddr specified the core's I2C memory address and (i2caddr+1) will be the cores I2C configuration address.	0 - 126	0
oepol	Output enable polarity	0 - 1	0
filter	Low-pass filter length. This generic should specify, in number of system clock cycles plus one, the time of the shortest pulse on the I2C bus to be registered as a valid value. For instance, to disregard any pulse that is 50 ns or shorter in a system with a system frequency of 54 MHz this generic should be set to: ((pulse time) / (clock period)) + 1 = (50 ns) / ((1/(54 MHz)) + 1 = 3.7 The value from this calculation should always be rounded up. In other words an appropriate filter length for a 54 MHz system is 4.	2 - 512	2

92.7 Signal descriptions

Table 1823 shows the interface signals of the core (VHDL ports).

*Table 1823.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBI	*	Input	AHB master input signals	-
AHBO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
I2CI	SCL	Input	I^2C clock line input	-
	SDA	Input	I^2C data line input	-
I2CO	SCL	Output	I^2C clock line output	-
	SCLOEN	Output	I^2C clock line output enable	Low**
	SDA	Output	I^2C data line output	-
	SDAOEN	Output	I^2C data line output enable	Low**
	ENABLE	Output	High when core is enabled, low otherwise.	High

* see GRLIB IP Library User's Manual
** depends on value of OEPOL VHDL generic.

92.8 Signal definitions and reset values

The signals and their reset values are described in table 1824.

*Table 1824.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
scl	InputOutput	I^2C clock line	-	Hi-Z
sda	InputOutput	I^2C data line	-	Hi-Z

92.9 Library dependencies

Table 1825 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1825.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	I2C	Component, signals	Component declaration, I2C signal definitions

92.10 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;
```

```
library gaisler;
use gaisler.misc.all;

entity i2c2ahb_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- I2C signals
    iic_scl : inout std_ulogic;
    iic_sda : inout std_ulogic
    );
end;

architecture rtl of i2c2ahb_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector;
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector;

  -- I2C signals
  signal i2ci : i2c_in_type;
  signal i2co : i2c_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- NOTE: There are also wrappers for the top-level entities that do not make use of VHDL
  --       records. These wrappers are called i2c2ahb_apb_gen and i2c2ahb_gen.

  -- I2C-slave, with APB interface
  i2c2ahb0 : i2c2ahb_apb
    generic map (
      hindex   => 1,
      ahbaddrh => ahbaddrh,
      ahbaddrl => ahbaddrl,
      ahbmaskh => ahbmaskh,
      ahbmaskl => ahbmaskl,
      resen    => 1,
      pindex   => 1,
      paddr    => 1,
      pmask    => 16#fff#,
      i2caddr  => i2caddr,
      oepol    => 0,
      filter   => I2C_FILTER)
    port map (rstn, clk, ahbmi, ahbmo(1), apbi, apbo(1),
              i2ci, i2co);
  scl_pad : iopad generic map (tech => padtech)
    port map (iic_scl, i2co.scl, i2co.scloen, i2ci.scl);
  sda_pad : iopad generic map (tech => padtech)
    port map (iic_sda, i2co.sda, i2co.sdaoen, i2ci.sda);
end;
```

93 I2CMST - I²C-master

93.1 Overview

The I²C-master core is a modified version of the OpenCores I²C-Master with an AMBA APB interface. The core is compatible with Philips I²C standard and supports 7- and 10-bit addressing. Standard-mode (100 kb/s) and Fast-mode (400 kb/s) operation are supported directly. External pull-up resistors must be supplied for both bus lines.

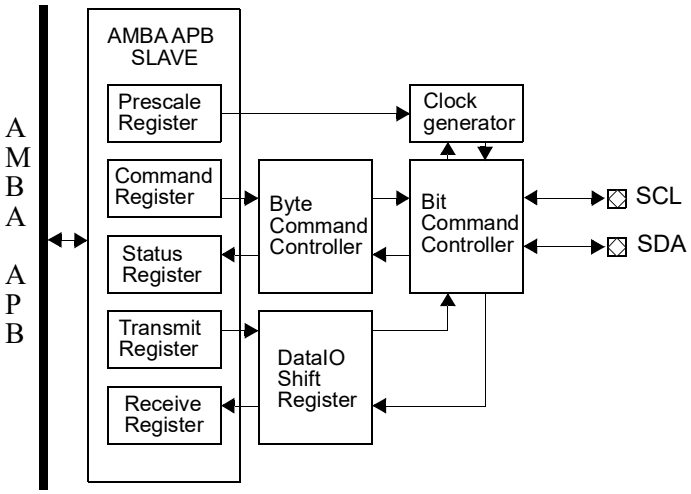


Figure 247. Block diagram

93.2 Operation

93.2.1 Transmission protocol

The I²C-bus is a simple 2-wire serial multi-master bus with collision detection and arbitration. The bus consists of a serial data line (SDA) and a serial clock line (SCL). The I²C standard defines three transmission speeds; Standard (100 kb/s), Fast (400 kb/s) and High speed (3.4 Mb/s).

A transfer on the I²C-bus begins with a START condition. A START condition is defined as a high to low transition of the SDA line while SCL is high. Transfers end with a STOP condition, defined as a low to high transition of the SDA line while SCL is high. These conditions are always generated by a master. The bus is considered to be busy after the START condition and is free after a certain amount of time following a STOP condition. The bus free time required between a STOP and a START condition is defined in the I²C-bus specification and is dependent on the bus bit rate.

Figure 248 shows a data transfer taking place over the I²C-bus. The master first generates a START condition and then transmits the 7-bit slave address. The bit following the slave address is the R/$\overline{W}$ bit which determines the direction of the data transfer. In this case the R/$\overline{W}$ bit is zero indicating a write operation. After the master has transmitted the address and the R/$\overline{W}$ bit it releases the SDA line. The receiver pulls the SDA line low to acknowledge the transfer. If the receiver does not acknowledge the transfer, the master may generate a STOP condition to abort the transfer or start a new transfer by generating a repeated START condition.

After the first byte has been acknowledged the master transmits the data byte. If the R/$\overline{W}$ bit had been set to ‘1’ the master would have acted as a receiver during this phase of the transfer. After the data byte has been transferred the receiver acknowledges the byte and the master generates a STOP condition to complete the transfer. Section 93.2.3 contains three more example transfers from the perspective of a software driver.

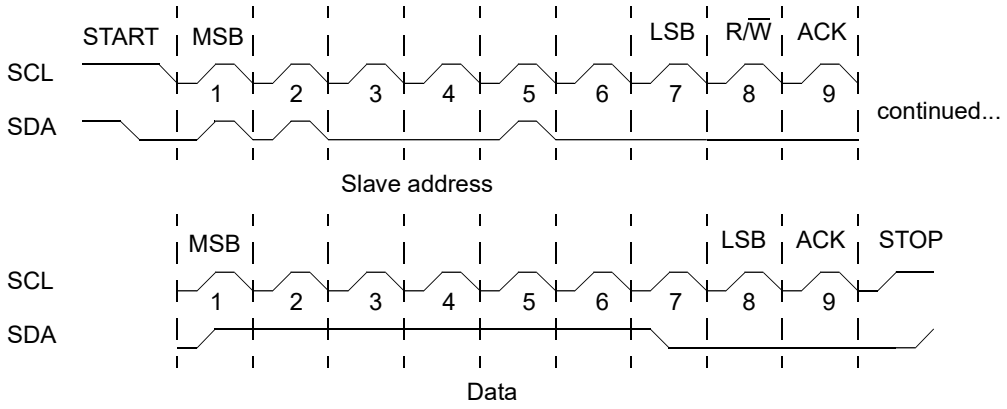


Figure 248. Complete I^{2}C data transfer

If the data bitrate is too high for a slave device, it may stretch the clock period by keeping SCL low after the master has driven SCL low.

93.2.2 Clock generation

The core uses the prescale register to determine the frequency of the SCL clock line and of the 5*SCL clock that the core uses internally. To calculate the prescale value use the formula:

$$Prescale = \frac{AMBAclockfrequency}{5 \cdot SCLfrequency} - 1$$

The *SCLfrequency* is 100 kHz for Standard-mode operation (100 kb/s) and 400 kHz for Fast mode operation. To use the core in Standard-mode in a system with a 60 MHz clock driving the AMBA bus the required prescale value is:

$$Prescale = \frac{60Mhz}{5 \cdot 100kHz} - 1 = 119 = 0x77$$

Note that the prescale register should only be changed when the core is disabled. The minimum recommended prescale value is 3 due to synchronization issues. This limits the minimum system frequency to 2 MHz for operation in Standard-mode (to be able to generate a 100 kHz SCL clock). However, a system frequency of 2 MHz will not allow the implementation fulfill the 100 ns minimum requirement for data setup time (required for Fast-mode operation). For compatibility with the I^{2}C Specification, in terms of minimum required data setup time, the minimum allowed system frequency is 20 MHz due to synchronization issues. If the core is run at lower system frequencies, care should be taken so that data from devices is stable on the bus one system clock period before the rising edge of SCL.

93.2.3 Software operational model

The core is initialized by writing an appropriate value to the clock prescale register and then setting the enable (EN) bit in the control register. Interrupts are enabled via the interrupt enable (IEN) bit in the control register.

To write a byte to a slave the I^{2}C-master must generate a START condition and send the slave address with the R/$\overline{W}$ bit set to ‘0’. After the slave has acknowledged the address, the master transmits the

data, waits for an acknowledge and generates a STOP condition. The sequence below instructs the core to perform a write:

1. Left-shift the I^{2}C-device address one position and write the result to the transmit register. The least significant bit of the transmit register (R/$\overline{W}$) is set to '0'.

2. Generate START condition and send contents of transmit register by setting the STA and WR bits in the command register.

3. Wait for interrupt, or for TIP bit in the status register to go low.

4. Read RxACK bit in status register. If RxACK is low the slave has acknowledged the transfer, proceed to step 5. If RxACK is set the device did not acknowledge the transfer, go to step 1.

5. Write the slave-data to the transmit register.

6. Send the data to the slave and generate a stop condition by setting STO and WR in the command register.

7. Wait for interrupt, or for TIP bit in the status register to go low.

8. Verify that the slave has acknowledged the data by reading the RxACK bit in the status register. RxACK should not be set.

To read a byte from an I^{2}C-connected memory much of the sequence above is repeated. The data written in this case is the memory location on the I^{2}C slave. After the address has been written the master generates a repeated START condition and reads the data from the slave. The sequence that software should perform to read from a memory device:

1. Left-shift the I^{2}C-device address one position and write the result to the transmit register. The least significant bit of the transmit register (R/W) is set to '0'.

2. Generate START condition and send contents of transmit register by setting the STA and WR bits in the command register.

3. Wait for interrupt or for TIP bit in the status register to go low.

4. Read RxACK bit in status register. If RxACK is low the slave has acknowledged the transfer, proceed to step 5. If RxACK is set the device did not acknowledge the transfer, go to step 1.

5. Write the memory location to be read from the slave to the transmit register.

6. Set the WR bit in the command register. Note that a STOP condition is not generated here.

7. Wait for interrupt, or for TIP bit in the status register to go low.

8. Read RxACK bit in the status register. RxACK should be low.

9. Address the I^{2}C-slave again by writing its left-shifted address into the transmit register. Set the least significant bit of the transmit register (R/W) to '1' to read from the slave.

10. Set the STA and WR bits in the command register to generate a repeated START condition.

11. Wait for interrupt, or for TIP bit in the status register to go low.

12. Read RxACK bit in the status register. The slave should acknowledge the transfer.

13. Prepare to receive the data read from the I^{2}C-connected memory. Set bits RD, ACK and STO on the command register. Setting the ACK bit NAKs the received data and signifies the end of the transfer.

14. Wait for interrupt, or for TIP in the status register to go low.

15. The received data can now be read from the receive register.

To perform sequential reads the master can iterate over steps 13 - 15 by not setting the ACK and STO bits in step 13. To end the sequential reads the ACK and STO bits are set. Consult the documentation of the I^{2}C-slave to see if sequential reads are supported.

The final sequence illustrates how to write one byte to an I^{2}C-slave which requires addressing. First the slave is addressed and the memory location on the slave is transmitted. After the slave has acknowledged the memory location the data to be written is transmitted without a generating a new START condition:

1. Left-shift the I^{2}C-device address one position and write the result to the transmit register. The least significant bit of the transmit register (R/W) is set to '0'.
2. Generate START condition and send contents of transmit register by setting the STA and WR bits in the command register.
3. Wait for interrupt or for TIP bit in the status register to go low.
4. Read RxACK bit in status register. If RxACK is low the slave has acknowledged the transfer, proceed to step 5. If RxACK is set the device did not acknowledge the transfer, go to step 1.
5. Write the memory location to be written from the slave to the transmit register.
6. Set the WR bit in the command register.
7. Wait for interrupt, or for TIP bit in the status register to go low.
8. Read RxACK bit in the status register. RxACK should be low.
9. Write the data byte to the transmit register.
10. Set WR and STO in the command register to send the data byte and then generate a STOP condition.
11. Wait for interrupt, or for TIP bit in the status register to go low.
12. Check RxACK bit in the status register. If the write succeeded the slave should acknowledge the data byte transfer.

The example sequences presented here can be generally applied to I^{2}C-slaves. However, some devices may deviate from the protocol above, please consult the documentation of the I^{2}C-slave in question. Note that a software driver should also monitor the arbitration lost (AL) bit in the status register.

93.2.4 Signal filters

The core is configured at implementation to use one of two possible filter strategies: a static filter or a dynamic filter, the selection between the two options is made with the *dynfilt* VHDL generic.

With a static filter (*dynfilt* = 0) the core will implement low-pass filters using simple shift registers. The number of shift registers is determined by the VHDL generic *filter*. When all bits in a shift register are equal, the core will consider the state of the input signal (SCL or SDA) to have changed. An appropriate value for the filter generic is calculated via:

$$filter = \frac{\overline{pulsetime}}{systemclockperiod} + 1$$

To disregard any pulse that is 50 ns or shorter in a system with a system frequency of 54 MHz the *filter* generic should be set to: (50 ns) / ((1/(54 MHz)) + 1 = 3.7. The value from this calculation should always be rounded up. In other words an appropriate filter length for a 54 MHz system is 4.

With a dynamic filter (*dynfilt* = 1) the VHDL generic *filter* determines the number of bits implemented in a counter that controls the sample window. The reload value for the counter can then be specified by software by writing to the core's dynamic filter register available via the APB interface. The number of bits required for the dynamic counter is calculated using (where system clock period is the shortest system clock period that the design will use):

$$filter = \log 2\left(\frac{\overline{pulsetime}}{systemclockperiod} + 1\right)$$

When using dynamic filtering, the core will ignore all pulses shorter than the system clock period multiplied with the value of the FILT field in the core's Dynamic Filter register and may also ignore pulses that are shorter than 2 * FILT * (system clock period) - 1.

93.3 Registers

The core is programmed through registers mapped into APB address space.

Table 1826. I^2C-master registers

APB address offset	Register
0x00	Clock prescale register
0x04	Control register
0x08	Transmit register*
0x08	Receive register**
0x0C	Command register*
0x0C	Status register**
0x10	Dynamic filter register***

* Write only

** Read only

*** Only available on some implementations

93.3.1 I²C-Master Clock Prescale Register

Table 1827. 0x00 - PRESCALE - I²C-master Clock prescale register

31 16	15 0
RESERVED	Clock prescale
0	0xFFFF
r	rw

31 : 16 RESERVED

15:0 Clock prescale - Value is used to prescale the SCL clock line. Do not change the value of this register unless the EN field of the control register is set to '0'. The minimum recommended value of this register is 0x0003. Lower values may cause the master to violate I²C timing requirements due to synchronization issues.

93.3.2 I²C-Master Control Register

Table 1828. 0x04 - CTRL - I²C-master control register

31 8	7	6	5 0
RESERVED	EN	IEN	RESERVED
0	0	0	0
r	rw	rw	r

31 : 8 RESERVED

7 Enable (EN) - Enable I²C core. The core is enabled when this bit is set to '1'.

6 Interrupt enable (IEN) - When this bit is set to '1' the core will generate interrupts upon transfer completion.

5:0 RESERVED

93.3.3 I²C-Master Transmit Register

Table 1829. 0x08 - TX - I²C-master transmit register

31 8	7 1	0
RESERVED	TDATA	RW
0	0	0
-	w	w

31 : 8 RESERVED

7:1 Transmit data (TDATA) - Most significant bits of next byte to transmit via I²C

0 Read/Write (RW) - In a data transfer this is the data's least significant bit. In a slave address transfer this is the RW bit. '1' reads from the slave and '0' writes to the slave.

93.3.4 I²C-Master Receive Register

Table 1830. 0x08 - RX - I²C-master receive register

31 8	7 0
RESERVED	RDATA
	0
	r

31 : 8 RESERVED

7:0 Receive data (RDATA) - Last byte received over I²C-bus.

93.3.5 I²C-Master Command Register

Table 1831. 0x0C -CMD - I²C-master command register

31 … 8	7	6	5	4	3	2 … 1	0
RESERVED	STA	STO	RD	WR	ACK	RESERVED	IACK
0	0	0	0	0	0	0	0
r	w*	w*	w*	w*	w*	r	-

31 : 8 RESERVED

7 Start (STA) - Generate START condition on I²C-bus. This bit is also used to generate repeated START conditions.

6 Stop (STO) - Generate STOP condition

5 Read (RD) - Read from slave

4 Write (WR) - Write to slave

3 Acknowledge (ACK) - Used when acting as a receiver. ‘0’ sends an ACK, ‘1’ sends a NACK.

2:1 RESERVED

0 Interrupt acknowledge (IACK) - Clears interrupt flag (IF) in status register.

93.3.6 I²C-Master Status Register

Table 1832. 0x0C - STAT - I²C-master status register

31 … 8	7	6	5	4 … 2	1	0
RESERVED	RxACK	BUSY	AL	RESERVED	TIP	IF
0	0	0	0	0		
r	r	r	r	r	r	wc

31 : 8 RESERVED

7 Receive acknowledge (RxACK) - Received acknowledge from slave. ‘1’ when no acknowledge is received, ‘0’ when slave has acked the transfer.

6 I²C-bus busy (BUSY) - This bit is set to ‘1’ when a start signal is detected and reset to ‘0’ when a stop signal is detected.

5 Arbitration lost (AL) - Set to ‘1’ when the core has lost arbitration. This happens when a stop signal is detected but not requested or when the master drives SDA high but SDA is low.

4:2 RESERVED

1 Transfer in progress (TIP) - ‘1’ when transferring data and ‘0’ when the transfer is complete. This bit is also set when the core will generate a STOP condition.

0 Interrupt flag (IF) - This bit is set when a byte transfer has been completed and when arbitration is lost. If IEN in the control register is set an interrupt will be generated. New interrupts will ge generated even if this bit has not been cleared.

93.3.7 I²C-Master Dynamic Filter Register

*Table 1833.*0x10 - FILT - I²C-master dynamic filter register

31 x	x-1 0
RESERVED	FILT
0	all 1
r	rw*

31 : x RESERVED

x-1 : 0 Dynamic filter reload value (FILT) - This field sets the reload value for the dynamic filter counter. The core will ignore all pulses on the bus shorter than FILT * (system clock period) and may also ignore pulses shorter than 2 * FILT * (system clock period) - 1. The reset value of this register is all '1'.

This register is not available in all implementations, and only for core revisions higher than two (the core's version number can be read from the plug'n'play area). When implemented, the number of bits in the FILT field is implementation dependent. Software can probe the precense of this register by writing 0x1 to the register location and reading back the value. If the read value is non-zero then the core has been implemented with a dynamic filter.

93.4 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x028. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

93.5 Implementation

93.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

93.6 Configuration options

Table 1834 shows the configuration options of the core (VHDL generics).

*Table 1834.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by I^2C-master	0 - NAHBIRQ-1	0
oepol	Output enable polarity	0 - 1	0
filter	Low-pass filter length. This generic should specify, in number of system clock cycles plus one, the time of the shortest pulse on the I2C bus to be registered as a valid value. For instance, to disregard any pulse that is 50 ns or shorter in a system with a system frequency of 54 MHz this generic should be set to: ((pulse time) / (clock period)) + 1 = (50 ns) / ((1/(54 MHz)) + 1 = 3.7 The value from this calculation should always be rounded up. In other words an appropriate filter length for a 54 MHz system is 4. Note that the value of this generic changes meaning if the dynfilt generic described below is non-zero. See description below.	2 - 512	2
dynfilt	Dynamic low-pass filter length. If this generic is non-zero the core will be implemented with a configurable filter. If dynfilt is non-zero the filter generic, described above, specifies how many bits that will be implemented for the dynamic filter counter.	0 - 1	0

93.7 Signal descriptions

Table 1835 shows the interface signals of the core (VHDL ports).

*Table 1835.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
I2CI	SCL	Input	I^2C clock line input	-
	SDA	Input	I^2C data line input	-
I2CO	SCL	Output	I^2C clock line output	-
	SCLOEN	Output	I^2C clock line output enable	Low
	SDA	Output	I^2C data line output	-
	SDAOEN	Output	I^2C data line output enable	Low

* see GRLIB IP Library User's Manual

93.8 Signal definitions and reset values

The signals and their reset values are described in table 1836.

*Table 1836.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
scl	InputOutput	I^2C clock line		Tri-state
sda	InputOutput	I^2C data line	-	Tri-state

93.9 Timing

The timing waveforms and timing parameters are shown in figure 249 and are defined in table 1837.

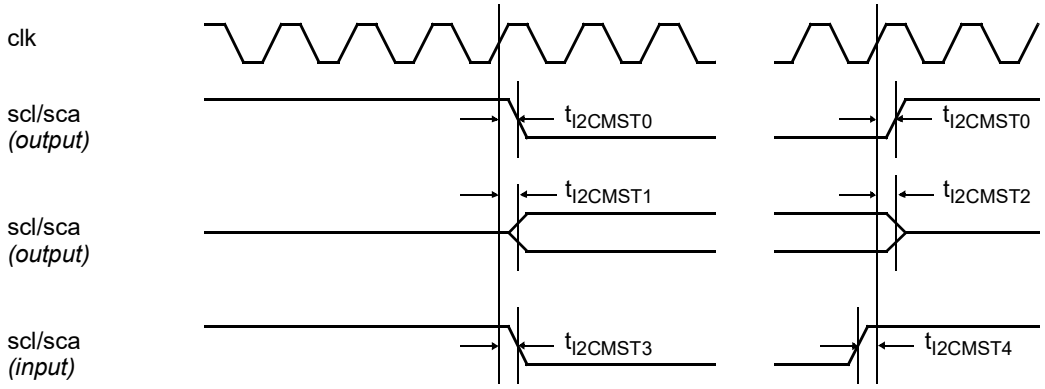


Figure 249. Timing waveforms

*Table 1837.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{I2CMST0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns
$t_{I2CMST1}$	clock to non-tri-state delay	rising *clk* edge	TBD	TBD	ns
$t_{I2CMST2}$	clock to tri-state delay	rising *clk* edge	TBD	TBD	ns
$t_{I2CMST3}$	input to clock hold[x]	rising *clk* edge	x	x	ns
$t_{I2CMST4}$	input to clock setup[x]	rising *clk* edge	x	x	ns
[x] The SCL and SDA inputs are re-synchronized internally and do not need to meet any setup or hold requirements relative to the core's AMBA APB clock.					

The core's I^2C bus functional timing depends on the core's scaler value. When the scaler is set for the core to operate in Fast- or Standard-Mode, the timing characteristics specified in the I^2C-bus Specification apply.

93.10 Library dependencies

Table 1838 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1838.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	I2C	Component, signals	Component declaration, I2C signal definitions

93.11 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.misc.all;

entity i2c_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- I2C signals
    iic_scl : inout std_ulogic;
    iic_sda : inout std_ulogic
    );
end;

architecture rtl of i2c_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- I2C signals
  signal i2ci : i2c_in_type;
  signal i2co : i2c_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- I2C-master
  i2c0 : i2cmst
    generic map (pindex => 12, paddr => 12, pmask => 16#FFF#,
                 pirq => 8, filter => (BUS_FREQ_in_kHz*5+50000)/100000+1)
    port map (rstn, clkm, apbi, apbo(12), i2ci, i2co);

  -- Using bi-directional pads:
  i2c_scl_pad : iopad generic map (tech => padtech)
    port map (iic_scl, i2co.scl, i2co.scloen, i2ci.scl);
  i2c_sda_pad : iopad generic map (tech => padtech)
    port map (iic_sda, i2co.sda, i2co.sdaoen, i2ci.sda);
  -- Note: Some designs may want to use a uni-directional pad for the clock. In this case the
  -- the clock should have a on-chip feedback like: i2ci.scl <= i2co.scloen (for OEPOL = 0)
  -- This feedback connection should have the same delay as i2co.sdaoen to i2ci.sda
end;
```

94 I2CSLV - I²C slave

94.1 Overview

The I²C slave core is a simple I²C slave that provides a link between the I²C bus and the AMBA APB. The core is compatible with Philips I²C standard and supports 7- and 10-bit addressing with an optionally software programmable address. Standard-mode (100 kb/s) and Fast-mode (400 kb/s) operation are supported directly. External pull-up resistors must be supplied for both bus lines.

GRLIB also contains another I²C slave core that has DMA capabilities, see the I2C2AHB core documentation for details.

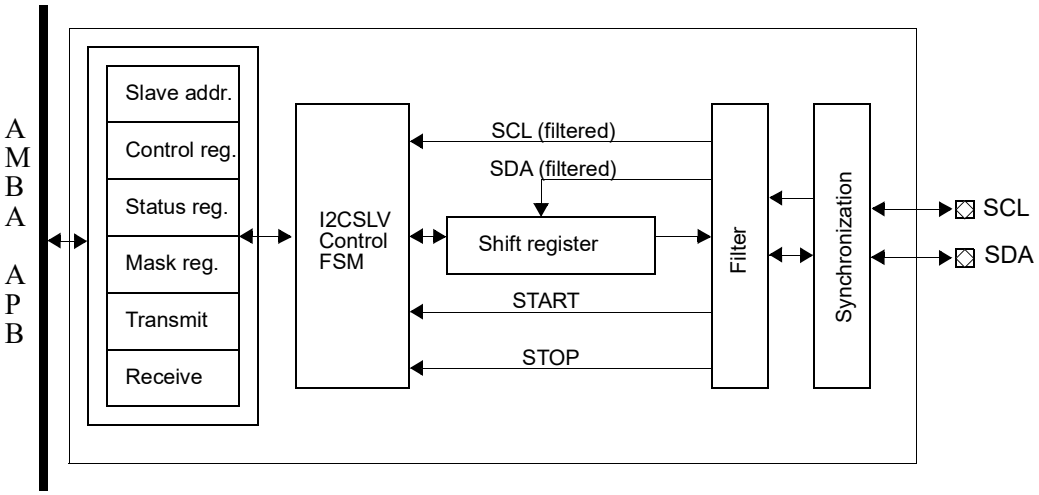


Figure 250. Block diagram

94.2 Operation

94.2.1 Transmission protocol

The I²C-bus is a simple 2-wire serial multi-master bus with collision detection and arbitration. The bus consists of a serial data line (SDA) and a serial clock line (SCL). The I²C standard defines three transmission speeds; Standard (100 kb/s), Fast (400 kb/s) and High speed (3.4 Mb/s).

A transfer on the I²C-bus begins with a START condition. A START condition is defined as a high to low transition of the SDA line while SCL is high. Transfers end with a STOP condition, defined as a low to high transition of the SDA line while SCL is high. These conditions are always generated by a master. The bus is considered to be busy after the START condition and is free after a certain amount of time following a STOP condition. The bus free time required between a STOP and a START condition is defined in the I²C-bus specification and is dependent on the bus bit rate.

Figure 251 shows a data transfer taking place over the I²C-bus. The master first generates a START condition and then transmits the 7-bit slave address. I²C also supports 10-bit addresses, which are discussed briefly below. The bit following the slave address is the R/$\overline{W}$ bit which determines the direction of the data transfer. In this case the R/$\overline{W}$ bit is zero indicating a write operation. After the master has transmitted the address and the R/$\overline{W}$ bit it releases the SDA line. The receiver pulls the SDA line low to acknowledge the transfer. If the receiver does not acknowledge the transfer, the master may generate a STOP condition to abort the transfer or start a new transfer by generating a repeated START condition.

After the address has been acknowledged the master transmits the data byte. If the R/$\overline{W}$ bit had been set to ‘1’ the master would have acted as a receiver during this phase of the transfer. After the data

byte has been transferred the receiver acknowledges the byte and the master generates a STOP condition to complete the transfer.

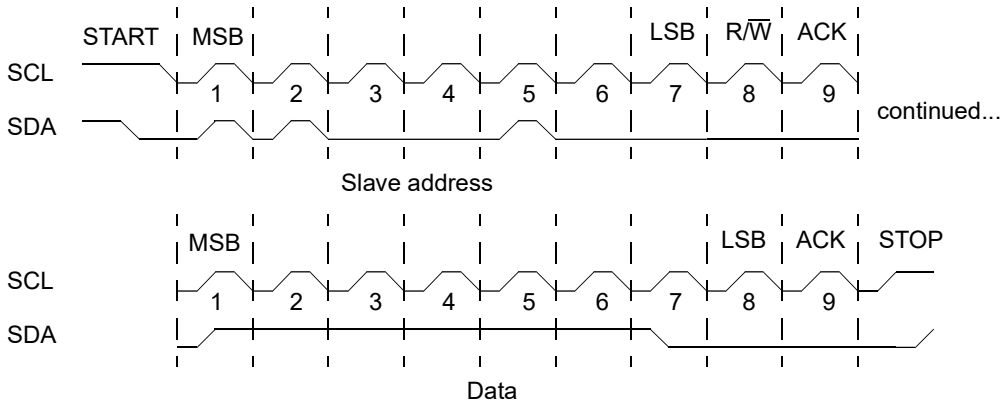


Figure 251. Complete I^{2}C data transfer

An I^{2}C slave may also support 10-bit addressing. In this case the master first transmits a pattern of five reserved bits followed by the two first bits of the 10-bit address and the R/$\overline{W}$ bit set to '0'. The next byte contains the remaining bits of the 10-bit address. If the transfer is a write operation the master then transmits data to the slave. To perform a read operation the master generates a repeated START condition and repeats the first part of the 10-bit address phase with the R/$\overline{W}$ bit set to '1'.

If the data bitrate is too high for a slave device or if the slave needs time to process data, it may stretch the clock period by keeping SCL low after the master has driven SCL low.

94.2.2 Slave addressing

The core's addressing support is implementation dependent. The core may have a programmable address and may support 10-bit addresses. If the core has support for 10-bit addressing, the TBA bit of the Slave address register will be set to '1' after reset. If the core's address is programmable this bit is writable and is used by the core to determine if it should listen to a 7- or 10-bit address.

Software can determine the addressing characteristics of the core by writing and reading the Slave address register. The core supports 10-bit addresses if the TBA bit is, or can be set, to '1'. The core has a software programmable address if the SLVADDR field in the same register can be changed.

94.2.3 System clock requirements and sampling

The core samples the incoming I^{2}C SCL clock and does not introduce any additional clock domains into the system. Both the SCL and SDA lines first pass through two stage synchronizers and are then filtered with a low pass filter consisting of four registers.

START and STOP conditions are detected if the SDA line, while SCL is high, is at one value for two system clock cycles, toggles and keeps the new level for two system clock cycles.

The synchronizers and filters constrain the minimum system frequency. The core requires the SCL signal to be stable for at least four system clock cycles before the core accepts the SCL value as the new clock value. The core's reaction to transitions will be additionally delayed since both lines are taken through two-stage synchronizers before they are filtered. Therefore it takes the core over eight system clock cycles to discover a transition on SCL. To use the slave in Standard-mode operation at 100 kHz the recommended minimum system frequency is 2 MHz. For Fast-mode operation at 400 kHz the recommended minimum system frequency is 6 MHz.

94.2.4 Operational model

The core has four main modes of operation and is configured to use one of these modes via the Control register bits Receive Mode (RMOD) and Transmit Mode (TMOD). The mode setting controls the core's behavior after a byte has been received or transmitted.

The core will always NAK a received byte if the receive register is full when the whole byte is received. If the receive register is free the value of RMOD determines if the core should continue to listen to the bus for the master's next action or if the core should drive SCL low to force the master into a wait state. If the value of the RMOD field is '0' the core will listen for the master's next action. If the value of the RMOD field is '1' the core will drive SCL low until the Receive register has been read and the Status register bit Byte Received (REC) has been cleared. Note that the core has not accepted a byte if it does not acknowledge the byte.

When the core receives a read request it evaluates the Transmit Valid (TV) bit in the Control register. If the Transmit Valid bit is set the core will acknowledge the address and proceed to transmit the data held in the Transmit register. After a byte has been transmitted the core assigns the value of the Control register bit Transmit Always Valid (TAV) to the Transmit Valid (TV) bit. This mechanism allows the same byte to be sent on all read requests without software intervention. The value of the Transmit Mode (TMOD) bit determines how the core acts after a byte has been transmitted and the master has acknowledged the byte, if the master NAKs the transmitted byte the transfer has ended and the core goes into an idle state. If TMOD is set to '0' when the master acknowledges a byte the core will continue to listen to the bus and wait for the master's next action. If the master continues with a sequential read operation the core will respond to all subsequent requests with the byte located in the Transmit Register. If TMOD is '1' the core will drive SCL low after a master has acknowledged the transmitted byte. SCL will be driven low until the Transmit Valid bit in the control register is set to '1'. Note that if the Transmit Always Valid (TAV) bit is set to '1' the Transmit Valid bit will immediately be set and the core will have show the same behavior for both Transmit modes.

When operating in Receive or Transmit Mode '1', the bus will be blocked by the core until software has acknowledged the transmitted or received byte. This may have a negative impact on bus performance and it also affects single byte transfers since the master is prevented to generate STOP or repeated START conditions when SCL is driven low by the core.

The core reports three types of events via the Status register. When the core NAKs a received byte, or its address in a read transfer, the NAK bit in the Status register will be set. When a byte is successfully received the core asserts the Byte Received (REC) bit. After transmission of a byte, the Byte Transmitted (TRA) bit is asserted. These three bits can be used as interrupt sources by setting the corresponding bits in the Mask register.

94.3 Registers

The core is programmed through registers mapped into APB address space.

Table 1839. I^2C slave registers

APB address offset	Register
0x00	Slave address register
0x04	Control register
0x08	Status register
0x0C	Mask register
0x10	Receive register
0x14	Transmit register

94.3.1 Slave Address Register

*Table 1840.*0x00 - SLVADDR - Slave address register

31	30 ... ALEN	ALEN-1 ... 0
TBA	RESERVED	SLVADDR
*	0	*
rw	r	rw*

31 Ten-bit Address (TBA) - When this bit is set the core will interpret the value in the SLVADDR field as a 10-bit address. If the core has 10-bit address support this bit will have the reset value '1'.

30 : ALEN RESERVED

(ALEN-1):0 Slave address (SLVADDR) - Contains the slave I2C address. The width of the slave address field, ALEN, is 7 bits (6:0) if the core only has support for 7-bit addresses. If the core has support for 10-bit addressing the width of SLVADDR is 10 bits. Depending on the hardware configuration this register may be read only. The core checks the length of the programmed address and will function with 7-bit addresses even if it has support for 10-bit addresses.

I^2C addresses can be allocated by NXP, please see the link in the core's overview section.

94.3.2 Control Register

*Table 1841.*0x04 - CTRL - Control register

31 ... 5	4	3	2	1	0
RESERVED	RMOD	TMOD	TV	TAV	EN
0	NR	NR	NR	NR	NR
r	rw	rw	rw	rw	rw

31 : 5 RESERVED

4 Receive Mode (RMOD) - Selects how the core handles writes:

'0': The slave accepts one byte and NAKs all other transfers until software has acknowledged the received byte by reading the Receive register.

'1': The slave accepts one byte and keeps SCL low until software has acknowledged the received byte by reading the Receive register.

3 Transmit Mode (TMOD) - Selects how the core handles reads:

'0': The slave transmits the same byte to all if the master requests more than one byte in the transfer. The slave then NAKs all read requests as long as the Transmit Valid (TV) bit is unset.

'1': The slave transmits one byte and then keeps SCL low until software has acknowledged that the byte has been transmitted by setting the Transmit Valid (TV) bit.

2 Transmit Valid (TV) - Software sets this bit to indicate that the data in the transmit register is valid. The core automatically resets this bit when the byte has been transmitted. When this bit is '0' the core will either NAK or insert wait states on incoming read requests, depending on the Transmit Mode (TMOD).

1 Transmit Always Valid (TAV) - When this bit is set, the core will not clear the Transmit Valid (TV) bit when a byte has been transmitted.

0 Enable core (EN) - Enables core. When this bit is set to '1' the core will react to requests to the address set in the Slave address register. If this bit is '0' the core will keep both SCL and SDA inputs in Hi-Z state.

94.3.3 Status Register

Table 1842. 0x08 - STAT - Status register

31 … 3	2	1	0
RESERVED	REC	TRA	NAK
0	0	0	0
r	*	wc	wc

31 : 3 RESERVED

2 Byte Received (REC) - This bit is set to '1' when the core accepts a byte and is automatically cleared when the Receive register has been read.

1 Byte Transmitted (TRA) - This bit is set to '1' when the core has transmitted a byte and is cleared by writing '1' to this position. Writes of '0' have no effect.

0 NAK Response (NAK) - This bit is set to '1' when the core has responded with NAK to a read or write request. This bit does not get set to '1' when the core responds with a NAK to an address that does not match the cores address. This bit is cleared by writing '1' to this position, writes of '0' have no effect.

94.3.4 Mask Register

Table 1843. 0x0C - MASK - Mask register

31 … 3	2	1	0
RESERVED	RECE	TRAE	NAKE
0	0	0	0
r	rw	rw	rw

31 : 3 RESERVED

2 Byte Received Enable (RECE) - When this bit is set the core will generate an interrupt when bit 2 in the Status register gets set.

1 Byte Transmitted Enable (TRAE) - When this bit is set the core will generate an interrupt when bit 1 in the Status register gets set.

0 NAK Response Enable (NAKE) - When this bit is set the core will generate an interrupt when bit 0 in the Status register gets set.

94.3.5 Receive Register

Table 1844. 0x10 - RX - Receive register

31 … 8	7 … 0
RESERVED	RECBYTE
0	NR
r	r

31 : 8 RESERVED

7:0 Received Byte (RECBYTE) - Last byte received from master. This field only contains valid data if the Byte received (REC) bit in the status register has been set.

94.3.6 Transmit Register

*Table 1845.*0x14 - TX - Transmit register

31 8	8 7 0
RESERVED	TRABYTE
0	NR
r	rw

31 : 8 RESERVED

7:0 Transmit Byte (TRABYTE) - Byte to transmit on the next master read request.

Reset value: Undefined

94.4 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x03E. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

94.5 Implementation

94.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

94.6 Configuration options

Table 1846 shows the configuration options of the core (VHDL generics).

*Table 1846.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by I^2C slave	0 - NAHBIRQ-1	0
hardaddr	If this generic is set to 1 the core uses the value of generic i2caddr as the hard coded address. If hardaddr is set to 0 the core's address can be changed via the Slave address register.	0 - 1	0
tenbit	If this generic is set to 1 the core will support 10-bit addresses. Note that the core can still be configured to use a 7-bit address.	0 - 1	0
i2caddr	The slave's (initial) I^2C address.	0 - 1023	0
oepol	Output enable polarity	0 - 1	0
filter	Low-pass filter length. This generic should specify, in number of system clock cycles plus one, the time of the shortest pulse on the I2C bus to be registered as a valid value. For instance, to disregard any pulse that is 50 ns or shorter in a system with a system frequency of 54 MHz this generic should be set to: ((pulse time) / (clock period)) + 1 = (50 ns) / ((1/(54 MHz)) + 1 = 3.7 The value from this calculation should always be rounded up. In other words an appropriate filter length for a 54 MHz system is 4.	2 - 512	2

94.7 Signal descriptions

Table 1847 shows the interface signals of the core (VHDL ports).

*Table 1847.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
I2CI	SCL	Input	I^2C clock line input	-
	SDA	Input	I^2C data line input	-
I2CO	SCL	Output	I^2C clock line output	-
	SCLOEN	Output	I^2C clock line output enable	Low**
	SDA	Output	I^2C data line output	-
	SDAOEN	Output	I^2C data line output enable	Low**
	ENABLE	Output	High when core is enabled, low otherwise	High

* see GRLIB IP Library User's Manual
** Depends on OEPOL VHDL generic

94.8 Signal definitions and reset values

The signals and their reset values are described in table 1848.

*Table 1848.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
scl	InputOutput	I^2C clock line	-	Hi-Z
sda	InputOutput	I^2C data line	-	Hi-Z

94.9 Library dependencies

Table 1849 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1849.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	I2C	Component, signals	Component declaration, I2C signal definitions

94.10 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.misc.all;
```

```
entity i2cslv_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- I2C signals
    iic_scl : inout std_ulogic;
    iic_sda : inout std_ulogic
    );
end;

architecture rtl of i2cslv_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- I2C signals
  signal i2ci : i2c_in_type;
  signal i2co : i2c_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- I2C-slave
  i2cslv0 : i2cslv
    generic map (pindex => 1, paddr => 1, pmask => 16#FFF#, pirq => 1,
                 hardaddr => 0, tenbit => 1, i2caddr => 16#50#)
    port map (rstn, clk, apbi, apbo(1), i2ci, i2co);
  i2cslv0_scl_pad : iopad generic map (tech => padtech)
    port map (iic_scl, i2co.scl, i2co.scloen, i2ci.scl);
  i2cslv0_sda_pad : iopad generic map (tech => padtech)
    port map (iic_sda, i2co.sda, i2co.sdaoen, i2ci.sda);
end;
```

95 IRQMP - Multiprocessor Interrupt Controller

95.1 Overview

The AMBA system in GRLIB provides an interrupt scheme where interrupt lines are routed together with the remaining AHB/APB bus signals, forming an interrupt bus. Interrupts from AHB and APB units are routed through the bus, combined together, and propagated back to all units. The multiprocessor interrupt controller core is attached to the AMBA bus as an APB slave, and monitors the combined interrupt signals.

The interrupts generated on the interrupt bus are all forwarded to the interrupt controller. The interrupt controller prioritizes, masks and propagates the interrupt with the highest priority to the processor. In multiprocessor systems, the interrupts are propagated to all processors.

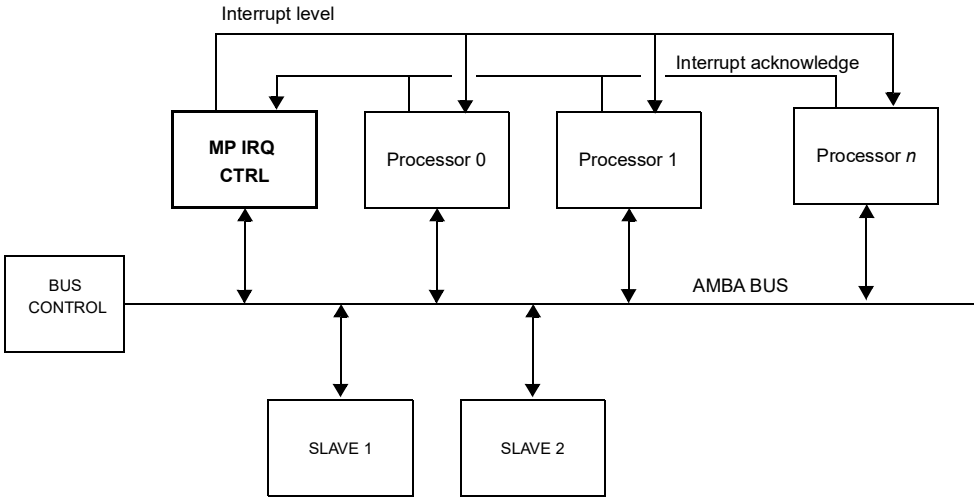


Figure 252. LEON multiprocessor system with Multiprocessor Interrupt controller

95.2 Operation

95.2.1 Interrupt prioritization

The interrupt controller monitors interrupt 1 - 15 of the interrupt bus (APBI.PIRQ[15:1]). When any of these lines are asserted high, the corresponding bit in the interrupt pending register is set. The pending bits will stay set even if the PIRQ line is de-asserted, until cleared by software or by an interrupt acknowledge from the processor. The default behaviour for peripherals is to use pulsed interrupts (an interrupt line is asserted for one clock cycle to signal an interrupt).

Each interrupt can be assigned to one of two priority levels (0 or 1) as programmed in the interrupt level register. Level 1 has higher priority than level 0. The interrupts are prioritized within each level, with interrupt 15 having the highest priority and interrupt 1 the lowest. The highest interrupt from level 1 will be forwarded to the processor. If no unmasked pending interrupt exists on level 1, then the highest unmasked interrupt from level 0 will be forwarded. PIRQ[31:16] are not used by the IRQMP core unless extended interrupts are enabled (see section 95.2.2). PIRQ[0] is not used by the IRQMP core unless interrupt remapping is enabled (see section 95.2.6).

Interrupts are prioritized at system level, while masking and forwarding of interrupts in done for each processor separately. Each processor in a multiprocessor system has separate interrupt mask and force registers. When an interrupt is signaled on the interrupt bus, the interrupt controller will prioritize

interrupts, perform interrupt masking for each processor according to the mask in the corresponding mask register and forward the interrupts to the processors.

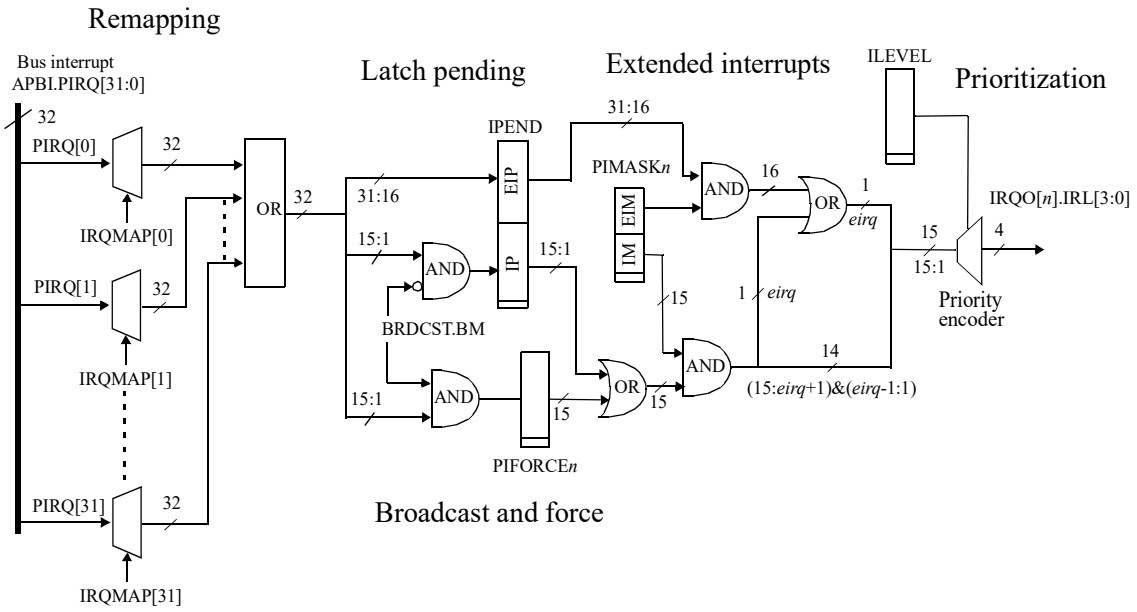


Figure 253. Interrupt controller block diagram. Interrupt acknowledgement is not shown. Logic is only shown for one processor. When multiple processors are implemented (*ncpu*>1), the interrupt forcing, masking, and priority encoding is performed separately for each processor. Each processor has its own PIFORCE*n* and PIMASK*n* register. The registers IRQMAP, IPEND, BRDCST, and ILEVEL are common to all processors. Interrupt remapping and extended interrupts are optional features that can be disabled at implementation time to reduce complexity.

When a processor acknowledges the interrupt (which happens automatically in the process of taking an interrupt trap), the corresponding pending bit will automatically be cleared. Note that in a multi-processor system, the bit in the pending register will be cleared as soon as one of the processors acknowledges the interrupt and interrupt broadcast functionality should be used for interrupts that need to be propagated to all processors.

Interrupts can also be forced by setting a bit in the interrupt force register. In this case, the processor acknowledgment will clear the force bit rather than the pending bit. After reset, the interrupt mask register is set to all zeros while the remaining control registers are undefined. Note that interrupt 15 cannot be masked by the LEON processor and should be used with care - most operating systems do not safely handle this interrupt.

Most GRLIB peripherals use pulsed interrupts, but some (notably GRGPIO and APBUART) can also generate level interrupts. A level interrupt line remains set until the condition that triggered the interrupt has cleared. Hence the interrupt line will typically remain asserted when the processor acknowledges the interrupt (as this occurs immediately when the interrupt trap is taken) and therefore the same interrupt will be latched into the pending interrupt register again on the next clock cycle. Therefore, the interrupt trap will usually be taken a second time as soon as software returns from the interrupt handler. It is possible to prevent such duplicate interrupts by explicitly clearing the pending interrupt from the interrupt handler, but if the interrupt line is shared with pulsed interrupts then care must be taken to prevent interrupts from being missed.

95.2.2 Extended interrupts

The AHB/APB interrupt bus consist of 32 signals ([31:0]), while the IRQMP only uses lines 1 - 15 in the nominal mode. To use the additional 16 interrupt lines (16-31), extended interrupt handling can be enabled by setting the VHDL generic *eirq* to a value between 1 - 15. The interrupt lines 16 - 31 will then also be handled by the interrupt controller, and the interrupt pending and mask registers will be

extended to 31 bits. Since the processor only has 15 interrupt levels (1 - 15), the extended interrupts will generate one of the regular interrupts, indicated by the value of the *eirq* generic. When the interrupt is taken and acknowledged by the processor, the regular interrupt (*eirq*) and the extended interrupt pending bits are automatically cleared. The extended interrupt acknowledge register will identify which extended interrupt that was most recently acknowledged. This register can be used by software to invoke the appropriate interrupt handler for the extended interrupts.

95.2.3 Processor status monitoring

The processor status can be monitored through the Multiprocessor Status Register. The STATUS field in this register indicates if a processor is in power-down ('1') or running ('0'). A processor can be made running by writing a '1' to its status field. After reset, all processors except processor 0 are halted (can be changed using the functionality associated with the *extrun* VHDL generic). When the system is properly initialized, processor 0 can start the remaining processors by writing to their STATUS bits. See also the extended boot support in section 95.2.4.

95.2.4 Extended boot support

When the bootreg generic is set, registers are added to allow starting a halted CPU from an arbitrary 8 byte aligned entry point. The CPU can be started with the same register write as when the entry point is written, or the CPU can be started later using the regular multiprocessor status register bit.

An error register is also added to allow monitoring CPUs for error mode, and to allow forcing a specific CPU into error mode. This can be used to monitor and re-boot CPUs without resetting the system.

A read-only bit in the multiprocessor status register can be read to see if the bootreg functionality has been implemented.

95.2.5 Interrupt broadcasting

The Broadcast Register is activated when the generic *ncpu* is > 1. An incoming irq that has its bit set in the Broadcast Register is propagated to the force register of *all* CPUs instead of to the Pending Register. This can be used to implement a timer that fires to all CPUs with that same irq.

95.2.6 Interrupt (re)map functionality

The interrupt controller can optionally be implemented (as an alternative to the two-level interrupt scheme) with functionality to allow dynamic remapping between bus interrupt lines and interrupt controller interrupt lines. If the design includes this functionality then switch-logic will be placed on the incoming interrupt vector from the AMBA bus before the IRQ pending register. The Interrupt map registers will be available starting at offset 0x300 from the interrupt controller's base address.

The interrupt map registers contain one field for each bus interrupt line in the system. The value within this field determines to which interrupt controller line the bus interrupt line is connected. In case several bus interrupt lines are mapped to the same controller interrupt line (several fields in the Interrupt map registers have the same value) then the bus interrupt lines will be OR:ed together.

Note that if bus interrupt line X is remapped to controller interrupt line Y then bit Y of the pending register will be set when a peripheral asserts interrupt X. Remapping interrupt lines via the Interrupt map registers has the same effect as changing the interrupt assignments in the RTL code.

95.3 Registers

The core is controlled through registers mapped into APB address space. The number of implemented registers depends on the number of processor in the multiprocessor system.

To make the boot registers available in the lower 256-byte space (the default APB bar size, with *pmask*=16#fff#), the registers are mapped into space that becomes unused when the number of processors (*ncpu*) is 8 or lower. If the number of CPUs is higher than 8, then the registers are only available in the higher space and a larger APB address mapping must be used to make them available. For example by setting *pmask*=16#ffc# to set the APB bar size to 1024 bytes.

Table 1850. Interrupt Controller registers

APB address offset	Register	Acronym
0x00	Interrupt level register	ILEVEL
0x04	Interrupt pending register	IPEND
0x08	Interrupt force register (*ncpu* = 0)	IFORCE
0x0C	Interrupt clear register	ICLEAR
0x10	Multiprocessor status register	MPSTAT
0x14	Broadcast register	BRDCST
0x18	Error mode status register	ERRSTAT
0x40 + 4 * *n*	Processor *n* interrupt mask register	PIMASK*n*
0x60 + 4 * n	Alias for processor *n* boot address register (only if *bootreg*=1 and *ncpu*<9)	BADDR*n*
0x80 + 4 * *n*	Processor *n* interrupt force register	PIFORCE*n*
0xC0 + 4 * *n*	Processor *n* extended interrupt acknowledge register	PEXTACK*n*
0x200 + 0x4 * *n*	Processor n boot address register (if *bootreg* = 1)	BADDR*n*
0x300 + 0x4 * *n*	Interrupt map register *n* (if *irqmap* > 0)	IRQMAP*n*

95.3.1 Interrupt Level Register

*Table 1851.*0x000 - ILEVEL - Interrupt Level Register

31 16	15 1	0
RESERVED	IL[15:1]	R
0	NR	0
r	rw	r

31:16 Reserved

15:1 Interrupt Level *n* (IL[*n*]) - Interrupt priority level for interrupt *n*.

0 Reserved

95.3.2 Interrupt Pending Register

*Table 1852.*0x004 - IPEND - Interrupt Pending Register

31 16	15 1	0
EIP[31:16]	IP[15:1]	R
0	0	0
rw	rw	r

31:16 Extended Interrupt Pending *n* (EIP[*n*]) - Interrupt pending for extended interrupt *n*.

15:1 Interrupt Pending *n* (IP[*n*]) - Interrupt pending for interrupt *n*.

0 Reserved

95.3.3 Interrupt Force Register (*ncpu* = 1)

*Table 1853.*0x008 - IFORCE - Interrupt Force Register (*ncpu* = 1)

31 16	15 1	0
RESERVED	IF[15:1]	R
0	0	0
r	rw	r

31:16 Reserved

15:1 Interrupt Force *n* (IF[*n*]) - Force interrupt *n* for processor 0.

0 Reserved

95.3.4 Interrupt Clear Register

*Table 1854.*0x00C - ICLEAR - Interrupt Clear Register

31 16	15 1	0
EIC[31:16]	IC[15:1]	R
0	0	0
w	w	r

31:16 Extended Interrupt Clear *n* (EIC[*n*]) - Writing '1' to EIC[*n*] will clear extended interrupt *n* (IPEND.EIP[*n*] will be set to '0'). Writing '0' has no effect.

15:1 Interrupt Clear *n* (IC[*n*]) - Writing '1' to IC[*n*] will clear interrupt *n* (IPEND.IP[n] will be set to '0'). Writing '0' has no effect.

0 Reserved

95.3.5 Multiprocessor Status Register

*Table 1855.*0x010 - MPSTAT - Multiprocessor Status Register

31 28	27	26	25 20	19 16	15 0
NCPU	BA	ER	RESERVED	EIRQ	STATUS[15:0]
*	*	*	0	*	*
r	r	r	r	r	rw

31:28 Number of CPUs (NCPU) - Number of CPUs in the system minus 1. Computed from *ncpu*-generic.

27 Broadcast Available (BA) - Set to '1' if MPSTAT.NCPU>0.

26 Extended boot registers available (ER). Set to '1' if *bootreg* generic is 1.

25:20 Reserved

19:16 Extended IRQ (EIRQ) - Interrupt number (1 - 15) used for extended interrupts. Equal to the value of the *eirq* generic. Fixed to 0 if extended interrupts are disabled.

15:0 Power-down status of processor *n* (STATUS[*n*]) - '1' = power-down, '0' = running. Write STATUS[*n*] with '1' to start processor *n*.

95.3.6 Broadcast Register (*ncpu* > 1)

*Table 1856.*0x014 - BRDCST - Broadcast Register (*ncpu* > 1)

31 16	15 1	0
RESERVED	BM15:1]	R
0	0	0
r	rw	r

31:16 Reserved

15:1 Broadcast Mask *n* (BM[*n*]) - If BM[*n*] = '1' then interrupt *n* is broadcast (written to the Force Register of all CPUs), otherwise standard semantics apply (interrupt written to the IPEND register).

0 Reserved

95.3.7 Error Mode Status Register

*Table 1857.*0x018 - ERRSTAT - Error Mode Status Register

31 28	27 26	20 19	16	15 0
RESERVED				ERRMODE[15:0]
0				*
r				rw

31:16 Reserved

15:0 Read: Error mode status of CPU *n* (ERRMODE[*n*]) - '1' = error mode, '0' = other (debug/run/power-down).

Write: Force CPU *n* into error mode

Register is read-only if *bootreg* generic is 0.

95.3.8 Processor N Interrupt Mask Register

*Table 1858.*0x040 + 4**n* - PIMASK*n* - Processor *n* Interrupt Mask Register

31 16	15 1	0
EIM[31:16]	IM15:1]	R
0	0	0
rw	rw	r

31:16 Extended Interrupt Mask *n* (EIM[*n*]) - Interrupt mask for extended interrupt *n*.

15:1 Interrupt Mask *n* (IM[n]) - If IM[*n*] = '0' then interrupt *n* is masked, otherwise it is enabled.

0 Reserved

95.3.9 Processor N Interrupt Force Register (*ncpu* > 1)

Table 1859. 0x080 + 4*n* - PIFORCE*n* - Processor *n* Interrupt Force Register (*ncpu* > 1)

31 17	16	15 1	0
IFC[15:1]	R	IF15:1]	R
0	0	0	0
wc	r	rw*	r

31:17 Interrupt Force Clear *n* (IFC[*n*]) - Interrupt force clear for interrupt *n*.

16 Reserved

15:1 Interrupt Force *n* (IF[*n*]) - Force interrupt *n*.

0 Reserved

95.3.10 Processor N Extended Interrupt Acknowledge Register

Table 1860. 0x0C0 + 4*n* - PEXTACK*n* - Processor *n* Extended Interrupt Acknowledge Register

31 5	4 0
RESERVED	EID[4:0]
0	0
r	r

31:5 Reserved

4:0 Extended interrupt ID (EID) - ID (16-31) of the most recently acknowledged extended interrupt.

If this field is 0, and support for extended interrupts exist, the last assertion of interrupt *eirq* was not the result of an extended interrupt being asserted. If interrupt *eirq* is forced, or asserted, this field will be cleared unless one, or more, of the interrupts 31 - 16 are enabled and set in the pending register.

95.3.11 Processor N Boot Address Register (*bootreg* = 1)

Table 1861. 0x200 + 0x4*n* - BADDR*n* - Processor *n* Boot Address register (*bootreg* = 1)

31 28	27 26	20 19	16 15	3	2 1	0
BOOTADDR[31:3]					RES	AS
-					-	-
w					-	w

31:3 Entry point for booting up processor *n*, 8-byte aligned

2:1 Reserved (write 0)

0 Start processor immediately after setting address

95.3.12 Interrupt Map Register N (*irqmap* > 0)

Table 1862. 0x300+4*n* - IRQMAP*n* - Interrupt map register *n*

31 24	23 16	15 8	7 0
IRQMAP[*n**4]	IRQMAP[*n**4+1]	IRQMAP[*n**4+2]	IRQMAP[*n**4+3]
*n**4	*n**4+1	*n**4+2	*n**4+3
rw	rw	rw	rw

b+7 : b Interrupt map (IRQMAP) - If the core has been implemented to support interrupt mapping then the Interrupt map register at offset 0x300 + 4*n* specifies the mapping for interrupt lines 4*n* to 4*n*+3. The bus interrupt line 4*n*+*x* will be mapped to the interrupt controller interrupt line specified by the value of IRQMAP[*n**4+*x*].

95.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x00D. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

95.5 Implementation

95.5.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

95.6 Configuration options

Table 1863 shows the configuration options of the core (VHDL generics).

*Table 1863.*Configuration options

Generic	Function	Allowed range	Defau1lt
pindex	Selects which APB select signal (PSEL) that will be used to access the interrupt controller	0 to NAPBMAX-1	0
paddr	The 12-bit MSB APB address	16#000# to 16#fff#	0
pmask	The APB address mask	16#000# to 16#fff#	16#fff#
ncpu	Number of processors in multiprocessor system	1 to 16	1
eirq	Enable extended interrupts	0 - 15	0
irqmap	Enable interrupt (re-)map registers. If irqmap is set to 0 then interrupt map functionality is disabled. If irqmap is nonzero then (irqmap + eirq) = 1 includes map registers for bus interrupt lines 0 - 15. If (irqmap + eirq) > 1 then interrupt map registers are available for bus interrupt lines 0 - 31.	0 - 2	0
bootreg	Enable boot address register and error mode register.	0 - 1	1
extrun	Use external run vector. If this generic is set to 1 the start of processors after reset will be controlled via the input signal cpurun. If this generic is set to 0, CPU 0 will be started after reset and the other CPUs will be put in power-down mode. This requires that the *smp* VHDL generic on the LEON entity is nonzero.	0 - 1	0

95.7 Signal descriptions

Table 1864 shows the interface signals of the core (VHDL ports).

*Table 1864.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
IRQI[*n*]	INTACK	Input	Processor *n* Interrupt acknowledge	High
	IRL[3:0]		Processor *n* interrupt level	High
	PWD		Processor *n* in power down mode	High
	FPEN		Unused	-
	ERR		Processor *n* in error mode	High
IRQO[*n*]	IRL[3:0]	Output	Processor *n* Input interrupt level	High
	RESUME		Reset power-down and error mode of processor *n*	High
	RSTRUN		Start processor *n* after reset (SMP systems only)	High
	RSTVEC[31:12]		Always zero	-
	INDEX[3:0]		CPU index	-
	PWDSETADDR		In power-down/error mode, shift PC to nPC and set PWDNEWADDR to PC.	High
	PWDNEWADDR [31:2]		New PC value used with PWDSETADDR	-
	FORCEERR		Force CPU into error mode	High
CPURUN[]	N/A	Input	If position n in this vector is set to '1', processor n will be started after reset. Otherwise processor n will go into power-down. This signal is only used if VHDL generic extrun is /= 0.	High

* see GRLIB IP Library User's Manual

95.8 Library dependencies

Table 1865 shows libraries that should be used when instantiating the core.

*Table 1865.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	LEON3	Signals, component	Signals and component declaration

95.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.leon3.all;
```

```
entity irqmp_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    ...  -- other signals
    );
end;

architecture rtl of irqmp_ex is
  constant NCPU : integer := 4;

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
  signal ahbsi : ahb_slv_in_type;

  -- GP Timer Unit input signals
  signal irqi   : irq_in_vector(0 to NCPU-1);
  signal irqo   : irq_out_vector(0 to NCPU-1);

  -- LEON3 signals
  signal leon3i : l3_in_vector(0 to NCPU-1);
  signal leon3o : l3_out_vector(0 to NCPU-1);

begin

  -- 4 LEON3 processors are instantiated here
  cpu : for i in 0 to NCPU-1 generate
    u0 : leon3s generic map (hindex => i)
    port map (clk, rstn, ahbmi, ahbmo(i), ahbsi,
irqi(i), irqo(i), dbgi(i), dbgo(i));
  end generate;

  -- MP IRQ controller
  irqctrl0 : irqmp
  generic map (pindex => 2, paddr => 2, ncpu => NCPU)
  port map (rstn, clk, apbi, apbo(2), irqi, irqo);
end
```

96 IRQ(A)MP - Multiprocessor Interrupt Controller with extended ASMP support

96.1 Overview

The AMBA system in GRLIB provides an interrupt scheme where interrupt lines are routed together with the remaining AHB/APB bus signals, forming an interrupt bus. Interrupts from AHB and APB units are routed through the bus, combined together, and propagated back to all units. The multiprocessor interrupt controller core is attached to the AMBA bus as an APB slave, and monitors the combined interrupt signals.

The interrupts generated on the interrupt bus are all forwarded to the interrupt controller. The interrupt controller prioritizes, masks and propagates the interrupt with the highest priority. The interrupt controller is configured at instantiation to implement one or several internal interrupt controllers. Each processor in a system can then be dynamically routed to one of the internal controllers. This allows safe Asymmetric Multiprocessing (ASMP) operation. For Symmetric Multiprocessor (SMP) operation, several processors can be routed to the same internal interrupt controller.

The IRQ(A)MP core is an extended version of the traditional multiprocessor interrupt controller. If a design does not need to have extended support for Asymmetric Multiprocessing, nor support for interrupt timestamping, it is recommended to use the IRQMP core instead.

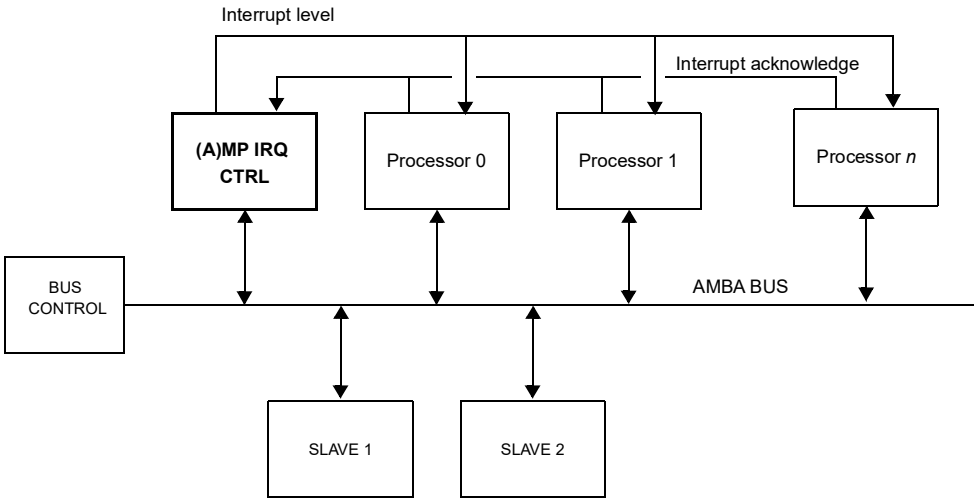


Figure 254. LEON multiprocessor system with Multiprocessor Interrupt controller

96.2 Operation

96.2.1 Support for Asymmetric Multiprocessing

Extended support for Asymmetric Multiprocessing (ASMP) is activated when the VHDL generic *nctrl* is > 1. Asymmetric Multiprocessing support means that parts of the interrupt controller are duplicated in order to provide safe ASMP operation. If the VHDL generic nctrl = 1 the core will have the same behavior as the normal IRQMP Multiprocessor interrupt controller core. If nctrl > 1, the core's register set will be duplicated on 4 KiB address boundaries. The core's register interface will also enable the use of three new registers, one Asymmetric Multiprocessing Control Register and two Interrupt Controller Select Registers.

Software can detect if the controller has been implemented with support for ASMP by reading the Asymmetric Multiprocessing Control register. If the field NCTRL is 0, the core was not implemented

with ASMP extensions. If the value of NCTRL is non-zero, the core has NCTRL+1 sets of registers with additional underlying functionality. From a software view this is equivalent to having NCTRL+1 interrupt controllers available and software can configure to which interrupt controller a processor should connect.

After system reset, all processors are connected to the first interrupt controller accessible at the core's base address. Software can then use the Interrupt Controller Select Registers to assign processors to other (internal) interrupt controllers. After assignments have been made, it is recommended to freeze the contents of the select registers by writing '1' to the lock bit in the Asymmetric Multiprocessing Control Register. The lock bit can be cleared by software by writing '0' to the bit.

When a software driver for the interrupt controller is loaded, the driver should check the Asymmetric Multiprocessing Control Register and Interrupt Controller Select Registers to determine to which controller the current processor is connected. After software has determined that it has been assigned to controller n, software should only access the controller with registers at offset 0x1000 * n. Note that the controllers are enumerated with the first controller being n = 0.

The processor specific registers (mask, force, interrupt acknowledge) can be read from all interrupt controllers. However the processor specific mask and interrupt acknowledge registers can only be written from the interrupt controller to which the processor is assigned. This also applies to individual bits in the Multiprocessor Status Register. Interrupt Force bits in a processor's Interrupt Force Register can only be cleared through the controller to which the processor is assigned. If the ICF field in the Asymmetric Multiprocessing Control Register is set to '1', all bits in all Interrupt Force Registers can be set, but not cleared, from all controllers. If the ICF field is '0' the bits in a processor's Interrupt Force register can only be set from the controller to which the processor is assigned.

96.2.2 Interrupt prioritization

The interrupt controller monitors interrupt 1 - 15 of the interrupt bus (APBI.PIRQ[15:1]). When any of these lines are asserted high, the corresponding bit in the interrupt pending register is set. The pending bits will stay set even if the PIRQ line is de-asserted, until cleared by software or by an interrupt acknowledge from the processor. The default behaviour for peripherals is to use pulsed interrupts (an interrupt line is asserted for one clock cycle to signal an interrupt).

Each interrupt can be assigned to one of two levels (0 or 1) as programmed in the interrupt level register. Level 1 has higher priority than level 0. The interrupts are prioritised within each level, with interrupt 15 having the highest priority and interrupt 1 the lowest. The highest interrupt from level 1 will be forwarded to the processor. If no unmasked pending interrupt exists on level 1, then the highest unmasked interrupt from level 0 will be forwarded. PIRQ[31:16] are not used by the IRQMP core.

Interrupts are prioritised at system level, while masking and forwarding of interrupts in done for each processor separately. Each processor in an multiprocessor system has separate interrupt mask and force registers. When an interrupt is signalled on the interrupt bus, the interrupt controller will prioritize interrupts, perform interrupt masking for each processor according to the mask in the corresponding mask register and forward the interrupts to the processors.

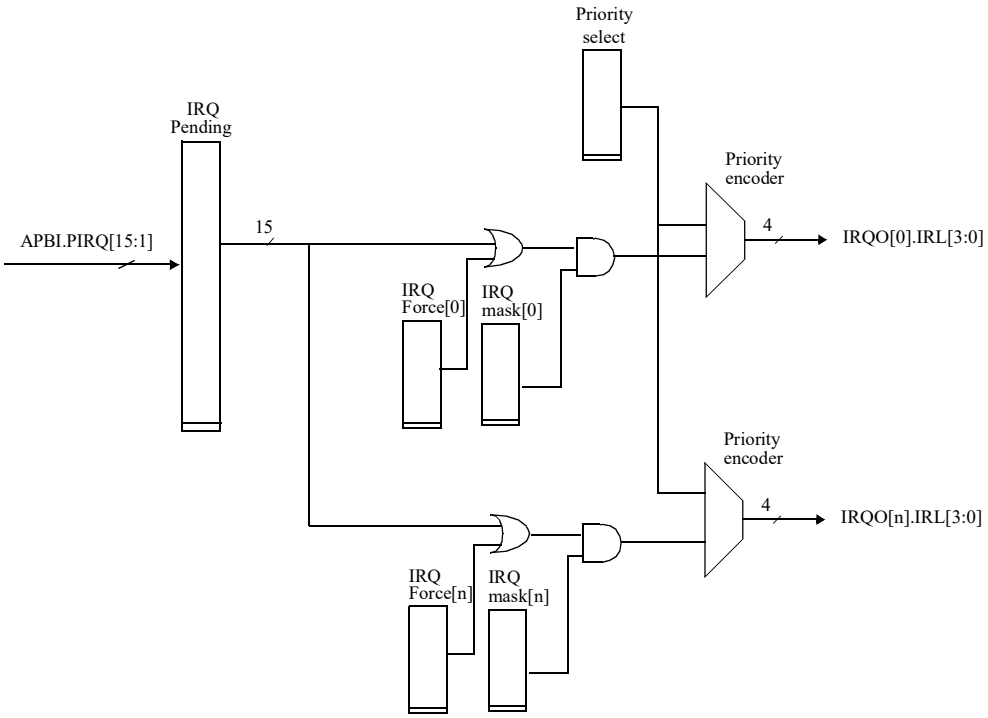


Figure 255. Interrupt controller block diagram

When a processor acknowledges the interrupt, the corresponding pending bit will automatically be cleared. Note that in a multiprocessor system, the bit in the pending register will be cleared as soon as one of the processors acknowledges the interrupt and interrupt broadcast functionality should be used for interrupts that need to be propagated to all processors. Interrupt can also be forced by setting a bit in the interrupt force register. In this case, the processor acknowledgement will clear the force bit rather than the pending bit. After reset, the interrupt mask register is set to all zeros while the remaining control registers are undefined. Note that interrupt 15 cannot be maskable by the LEON processor and should be used with care - most operating systems do not safely handle this interrupt.

96.2.3 Extended interrupts

The AHB/APB interrupt consist of 32 signals ([31:0]), while the IRQMP only uses lines 1 - 15 in the nominal mode. To use the additional 16 interrupt lines (16-31), extended interrupt handling can be enabled by setting the VHDL generic *eirq* to a value between 1 - 15. The interrupt lines 16 - 31 will then also be handled by the interrupt controller, and the interrupt pending and mask registers will be extended to 32 bits. Since the processor only has 15 interrupt levels (1 - 15), the extended interrupts will generate one of the regular interrupts, indicated by the value of the *eirq* generic. When the interrupt is taken and acknowledged by the processor, the regular interrupt (*eirq*) and the extended interrupt pending bits are automatically cleared. The extended interrupt acknowledge register will identify which extended interrupt that was most recently acknowledged. This register can be used by software to invoke the appropriate interrupt handler for the extended interrupts. When VHDL generic *irqmap* has a value of 3, support for 64 AHB/APB interrupts lines are enabled. Interrupt 32 - 63 are not handled by the interrupt controller and therefor needs to be remapped to one of the interrupt 1 - 31.

96.2.4 Processor status monitoring

The processor status can be monitored through the Multiprocessor Status Register. The STATUS field in this register indicates if a processor is in power-down (‘1’) or running (‘0’). A processor can be made running by writing a ‘1’ to its status field. After reset, all processors except processor 0 are in power-down (can be changed using the functionality associated with the *extrun* VHDL generic).

When the system is properly initialized, processor 0 can start the remaining processors by writing to their STATUS bits.

The core can be implemented with support for specifying the processor reset start address dynamically. Please see section 96.2.10 for further information.

96.2.5 Interrupt broadcasting

The Broadcast Register is activated when the generic *ncpu* is > 1. An incoming irq that has its bit set in the Broadcast Register is propagated to the force register of *all* CPUs instead of to the Pending Register. This can be used to implement a timer that fires to all CPUs with that same irq.

96.2.6 Interrupt timestamping description

Support for interrupt timestamping is implemented when the VHDL generic *tstamp* is > 0.

Interrupt timestamping is controlled via the Interrupt Timestamp Control register(s). Each Interrupt Timestamp Control register contains a field (TSTAMP) that contains the number of timestamp registers sets that the core implements. A timestamp register sets consist of one Interrupt Timestamp Counter register, one Interrupt Timestamp Control register, one Interrupt Assertion Timestamp register and one Interrupt Acknowledge Timestamp register.

Software enables timestamping for a specific interrupt via a Interrupt Timestamp Control Register. When the selected interrupt line is asserted, software will save the current value of the interrupt timestamp counter into the Interrupt Assertion Timestamp register and set the S1 field in the Interrupt Timestamp Control Register. When the processor acknowledges the interrupt, the S2 field of the Interrupt Timestamp Control register will be set and the current value of the timestamp counter will be saved in the Interrupt Acknowledge Timestamp Register. The difference between the Interrupt Assertion timestamp and the Interrupt Acknowledge timestamp is the number of system clock cycles that was required for the processor to react to the interrupt and divert execution to the trap handler.

The core can be configured to stamp only the first occurrence of an interrupt or to continuously stamp interrupts. The behavior is controlled via the Keep Stamp (KS) field in the Interrupt Timestamp Control Register. If KS is set, only the first assertion and acknowledge of an interrupt is stamped. Software must then clear the S1 and S2 fields for a new timestamp to be taken. If Keep Stamp is disabled (KS field not set), the controller will update the Interrupt Assertion Timestamp Register every time the selected interrupt line is asserted. In this case the controller will also automatically clear the S2 field and also update the Interrupt Acknowledge Timestamp register with the current value when the interrupt is acknowledged.

For controllers with extended ASMP support, each internal controller has a dedicated set of Interrupt timestamp registers. This means that the Interrupt Acknowledge Timestamp Register(s) on a specific controller will only be updated if and when the processor connected to the controller acknowledges the selected interrupt. The Interrupt Timestamp Counter is shared by all controllers and will be incremented when an Interrupt Timestamp Control register has the ITSEL field set to a non-zero value.

96.2.7 Interrupt timestamping usage guidelines

Note that KS = ‘0’ and a high interrupt rate may cause the Interrupt Assertion Timestamp register to be updated (and the S2 field reset) before the processor has acknowledged the first occurrence of the interrupt. When the processor then acknowledges the first occurrence, the Interrupt Acknowledge Timestamp register will be updated and the difference between the two Timestamp registers will not show how long it took the processor to react to the first interrupt request. If the interrupt frequency is expected to be high it is recommended to keep the first stamp (KS field set to ‘1’) in order to get reliable measurements. KS = ‘0’ should not be used in systems that include cores that use level interrupts, the timestamp logic will register each cycle that the interrupt line is asserted as an interrupt.

In order to measure the full interrupt handling latency in a system, software should also read the current value of the Interrupt Timestamp Counter when entering the interrupt handler. In the typical case,

a software driver's interrupt handler reads a status register and then determines the action to take. Adding a read of the timestamp counter before this status register read can give an accurate view of the latency during interrupt handling.

The core listens to the system interrupt vector when reacting to interrupt line assertions. This means that the Interrupt Assertion Timestamp Register(s) will not be updated if software writes directly to the pending or force registers. To measure the time required to serve a forced interrupt, read the value of the Interrupt Timestamp counter before forcing the interrupt and then read the Interrupt Acknowledge Timestamp and Interrupt Timestamp counter when the processor has reacted to the interrupt.

96.2.8 Watchdog

Support for watchdog inputs is implemented when the VHDL generic *wdogen* > 0, the number of watchdog input is determined by the VHDL generic *nwdog*.

The core can be implemented with support for asserting a bit in the controller's Interrupt Pending Register when an external watchdog signal is asserted. This functionality can be used to implement a sort of soft watchdog for one or several processor cores. The controller's Watchdog Control Register contains a field that shows the number of external watchdog inputs supported and fields for configuring which watchdog inputs that should be able to assert a bit in the Interrupt Pending Register. The pending register will be assigned in each cycle that a selected watchdog input is high. Therefore it is recommended that the watchdog inputs are connected to sources which send a one clock cycle long pulse when a watchdog expires. Otherwise software should make sure that the watchdog signal is deasserted before re-enabling interrupts during interrupt handling.

For controllers with extended ASMP support, each internal controller has a dedicated Watchdog Control register. Assertion of a watchdog input will only affect the pending register on the internal interrupt controllers that have enabled the watchdog input in their Watchdog Control Register.

96.2.9 Interrupt (re)map functionality

The interrupt controller can optionally be implemented (as an alternative to the two-interrupt levels scheme) with functionality to allow dynamic remapping between bus interrupt lines and interrupt controller interrupt lines. If the design includes this functionality then switch-logic will be placed on the incoming interrupt vector from the AMBA bus before the IRQ pending register. The Interrupt map registers will be available starting at offset 0x300 from the interrupt controller's base address.

The interrupt map registers contain one field for each bus interrupt line in the system. The value within this field determines to which interrupt controller line the bus interrupt line is connected. In case several bus interrupt lines are mapped to the same controller interrupt line (several fields in the Interrupt map registers have the same value) then the bus interrupt lines will be OR:ed together.

Note that if bus interrupt line X is remapped to controller interrupt line Y then bit Y of the pending register will be set when a peripheral asserts interrupt X. Remapping interrupt lines via the Interrupt map registers has the same effect as changing the interrupt assignments in the RTL code.

96.2.10 Dynamic processor reset start address

When the bootreg generic is set, registers are added to allow starting a halted CPU from an arbitrary 8 byte aligned entry point. The CPU can be started with the same register write as when the entry point is written, or the CPU can be started later using the regular multiprocessor status register bit.

An error register is also added to allow monitoring CPUs for error mode, and to allow forcing a specific CPU into error mode. This can be used to monitor and re-boot CPUs without resetting the system.

A read-only bit in the multiprocessor status register can be read to see if the bootreg functionality has been configured in.

96.3 Registers

The core is controlled through registers mapped into APB address space. The number of implemented registers depends on the number of processors in the multiprocessor system. The number of accessible register sets depend on the value of the NCTRL field in the Asymmetric Multiprocessing Control Register. The register set for controller n is accessed at offset 0x1000*n.

In order to make the boot registers available in the lower 256-byte space that is in legacy mappings, the registers are mapped into space that becomes unused when the number of processors is 8 or lower. If the number of CPUs is higher than 8, then the registers are only available in the higher space and a larger APB address mapping must be used to make them available.

Table 1866. Interrupt Controller registers

APB address offset	Register
0x000	Interrupt level register
0x004	Interrupt pending register
0x008	Interrupt force register (NCPU = 0)
0x00C	Interrupt clear register
0x010	Multiprocessor status register
0x014	Broadcast register
0x018	Error mode register
0x01C	Watchdog control register
0x020	Asymmetric multiprocessing control register
0x024	Interrupt controller select register for processor 0 - 7
0x028	Interrupt controller select register for processor 8 - 15
0x02C - 0x03C	Reserved
0x040	Processor interrupt mask register
0x044	Processor 1 interrupt mask register
0x040 + 0x4 * *n*	Processor *n* interrupt mask register
0x060 + 0x4 * *n*	Alias for processor *n* boot address register if NCPU < 9
0x080	Processor interrupt force register
0x084	Processor 1 interrupt force register
0x080 + 0x4 * *n*	Processor *n* interrupt force register
0x0C0	Processor extended interrupt acknowledge register
0x0C4	Processor 1 extended interrupt acknowledge register
0x0C0 + 0x4 * *n*	Processor *n* extended interrupt acknowledge register
0x100	Interrupt timestamp counter register
0x104	Interrupt timestamp 0 control register
0x108	Interrupt assertion timestamp 0 register
0x10C	Interrupt acknowledge timestamp 0 register
0x100 + 0x10 * *n*	Interrupt timestamp counter register (mirrored in each set)
0x104 + 0x10 * *n*	Interrupt timestamp *n* control register
0x108 + 0x10 * *n*	Interrupt assertion timestamp *n* register
0x10C + 0x10 * *n*	Interrupt acknowledge timestamp *n* register
0x200 + 0x4* *n*	Processor n boot address register
0x300 + 0x4 * *n*	Interrupt map register *n*

96.3.1 Interrupt Level Register

Table 1867. 0x000 - ILEVEL - Interrupt Level Register

31 16	15 1	0
RESERVED	IL[15:1]	R
0	NR	0
r	rw	r

31:16 Reserved

15:1 Interrupt Level n (IL[n]) - Interrupt level for interrupt n

0 Reserved

96.3.2 Interrupt Pending Register

Table 1868. 0x004 - IPEND - Interrupt Pending Register

31 16	15 1	0
EIP[31:16]	IP[15:1]	R
0	0	0
rw	rw	r

31:16 Extended Interrupt Pending n (EIP[n])

15:1 Interrupt Pending n (IP[n]) - Interrupt pending for interrupt n

0 Reserved

96.3.3 Interrupt Force Register (NCPU = 0)

Table 1869. 0x008 - IFORCE0 - Interrupt Force Register (NCPU = 0)

31 16	15 1	0
RESERVED	IF[15:1]	R
0	0	0
r	rw	r

31:16 Reserved

15:1 Interrupt Force n (IF[n]) - Force interrupt nr n.

0 Reserved

96.3.4 Interrupt Clear Register

Table 1870. 0x00C - ICLEAR - Interrupt Clear Register

31 16	15 1	0
EIC[31:16]	IC[15:1]	R
0	0	0
w	w	r

31:16 Extended Interrupt Clear n (EIC[n])

15:1 Interrupt Clear n (IC[n]) - Writing ‘1’ to IC[n] will clear interrupt n

0 Reserved

96.3.5 Multiprocessor Status Register

*Table 1871.*0x010 - MPSTAT - Multiprocessor Status Register

31 28	27	26	25 20	19 16	15 0
NCPU	BA	ER	RESERVED	EIRQ	STATUS[15:0]
*	*		0	*	*
r	r		r	r	rw

31:28 Number of CPUs (NCPU) - Number of CPUs in the system - 1

27 Broadcast Available (BA) - Set to '1' if NCPU > 0.

26 Extended boot registers available (ER). Set to '1' if bootreg generic is 1.

25:20 Reserved

19:16 Extended IRQ (EIRQ) - Interrupt number (1 - 15) used for extended interrupts. Fixed to 0 if extended interrupts are disabled.

15:0 Power-down status of CPU[n] (STATUS[n]) - '1' = power-down, '0' = running. Write STATUS[n] with '1' to start processor n.

96.3.6 Broadcast Register (NCPU > 0)

*Table 1872.*0x014 - BRDCST - Broadcast Register (NCPU > 0)

31 16	15 1	0
RESERVED	BM15:1]	R
0	0	0
r	rw	r

31:16 Reserved

15:1 Broadcast Mask n (BM[n]) - If BM[n] = '1' then interrupt n is broadcasted (written to the Force Register of all CPUs), otherwise standard semantic applies (Pending register)

0 Reserved

96.3.7 Error Mode Status Register

*Table 1873.*0x018 - ERRSTAT - Error Mode Status Register

31 28	27 26	20 19	16 15	0
RESERVED			ERRMODE[15:0]	
0			*	
r			rw	

31:16 Reserved

15:0 Read: Error mode status of CPU[n] (STATUS[n]) - '1' = error mode, '0' = other (debug/run/power-down).

Write: Force CPU[n] into error mode

Register is read-only if bootreg generic is 0..

96.3.8 Watchdog Control Register (NCPU > 0)

Table 1874. 0x01C - WDOGCTRL - Watchdog Control Register (NCPU > 0)

31 27	26 20	19 16	15 0
NWDOG	Reserved	WDOGIRQ	WDOGMSK
*	0	NR	0
r	r	rw	rw

31:27 Number of watchdog inputs (NWDOG) - Number of watchdog inputs that the core supports.

26:20 Reserved

19:16 Watchdog interrupt (WDOGIRQ) - Selects the bit in the pending register to set when any line watchdog line selected by the WDOGMSK field is asserted.

15:0 Watchdog Mask n (WDOGMSK[n]) - If WDOGMSK[n] = '1' then the assertion of watchdog input n will lead to the bit selected by the WDOGIRQ field being set in the controller's Interrupt Pending Register.

96.3.9 Asymmetric Multiprocessing Control Register

Table 1875. 0x020 - ASMPCTRL - Asymmetric Multiprocessing Control Register

31 28	27 2	1	0
NCTRL	RESERVED	ICF	L
*	0	0	0
r	r	rw	rw

31:28 Number of internal controllers (NCTRL) - NCTRL + 1 is the number of internal interrupt controllers available.

27:2 Reserved

1 Inter-controller Force (ICF) - If this bit is set to '1' all Interrupt Force Registers can be set from any internal controller. If this bit is '0', a processor's Interrupt Force Register can only be set from the controller to which the processor is connected. Bits in an Interrupt Force Register can only be cleared by the controller or by writing the Interrupt Force Clear field on the controller to which the processor is connected.

0 Lock (L) - If this bit is written to '1', the contents of the Interrupt Controller Select registers is frozen. This bit can only be set if NCTRL > 0.

96.3.10 Interrupt Controller Select Register for Processors 0 - 7 (NCTRL > 0)

Table 1876. 0x024 - ICLSELR0 - Interrupt Controller Select Register for Processors 0 -7 (NCTRL > 0)

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
ICSEL0	ICSEL1	ICSEL2	ICSEL3	ICSEL4	ICSEL5	ICSEL6	ICSEL7
0	0	0	0	0	0	0	0
rw	rw	rw	rw	rw	rw	rw	rw

31:0 Interrupt controller select for processor n (ICSEL[n]) - The nibble ICSEL[n] selects the (internal) interrupt controller to connect to processor n.

96.3.11 Interrupt Controller Select Register for Processors 8 - 15 (NCTRL > 0)

Table 1877. 0x028 - ICSELR1 - Interrupt Controller Select Register for Processors 8 - 15 (NCTRL > 0)

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
ICSEL8	ICSEL9	ICSEL10	ICSEL11	ICSEL12	ICSEL13	ICSEL14	ICSEL15
0	0	0	0	0	0	0	0
rw	rw	rw	rw	rw	rw	rw	rw

31:0 Interrupt controller select for processor n (ICSEL[n]) - The nibble ICSEL[n] selects the (internal) interrupt controller to connect to processor n.

96.3.12 Processor Interrupt Mask Register

*Table 1878.*0x040,... - PIMASK - Processor Interrupt Mask Register

31 16	15 1	0
EIM[31:16]	IM15:1]	R
0	0	0
rw	rw	r

31:16	Extended Interrupt Mask n (EIC[n]) - Interrupt mask for extended interrupts
15:1	Interrupt Mask n (IM[n]) - If IM[n] = ‘0’ then interrupt n is masked, otherwise it is enabled.
0	Reserved

96.3.13 Processor Interrupt Force Register (NCPU > 0)

*Table 1879.*0x080 - PCFORCE - Processor Interrupt Force Register (NCPU > 0)

31 17	16	15 1	0
IFC[15:1]	R	IF15:1]	R
0	0	0	0
wc	r	rw*	r

31:17	Interrupt Force Clear n (IFC[n]) - Interrupt force clear for interrupt n
16	Reserved
15:1	Interrupt Force n (IF[n]) - Force interrupt nr n
0	Reserved

96.3.14 Extended Interrupt Acknowledge Register

*Table 1880.*0x0C0,... PEXTACK - Extended Interrupt Acknowledge Register

31 5	4 0
RESERVED	EID[4:0]
0	0
r	r

31:5 Reserved

4:0 Extended interrupt ID (EID) - ID (16-31) of the most recent acknowledged extended interrupt

If this field is 0, and support for extended interrupts exist, the last assertion of interrupt *eirq* was not the result of an extended interrupt being asserted. If interrupt *eirq* is forced, or asserted, this field will be cleared unless one, or more, of the interrupts 31 - 16 are enabled and set in the pending register.

96.3.15 Interrupt Timestamp Counter Register

*Table 1881.*0x100,110 - TCNT - Interrupt Timestamp Counter register

31 0
TCNT
0
r

31:0 Timestamp Counter (TCNT) - Current value of timestamp counter. The counter increments whenever a TSISEL field in a Timestamp Control Register is non-zero. The counter will wrap to zero upon overflow and is read only.

96.3.16 Timestamp N Control Register

*Table 1882.*0x1n4 - ITSTMPCn - Timestamp n Control Register

31 27	26	25	24 6	5	4 0
TSTAMP	S1	S2	RESERVED	KS	TSISEL
*	0	0	0	0	0
r	wc	wc	r	rw	rw

31:27 Number of timestamp register sets (TSTAMP) - The number of available timestamp register sets.

26 Assertion Stamped (S1) - Set to ‘1’ when the assertion of the selected line has received a timestamp. This bit is cleared by writing ‘1’ to its position. Writes of ‘0’ have no effect.

25 Acknowledge Stamped (S2) - Set to ‘1’ when the processor acknowledge of the selected interrupt has received a timestamp. This bit can be cleared by writing ‘1’ to this position, writes of ‘0’ have no effect. This bit can also be cleared automatically by the core, see description of the KS field below.

24:6 RESERVED

5 Keep Stamp (KS) - If this bit is set to ‘1’ the core will keep the first stamp value for the first interrupt until the S1 and S2 fields are cleared by software. If this bit is set to ‘0’ the core will time stamp the most recent interrupt. This also has the effect that the core will automatically clear the S2 field whenever the selected interrupt line is asserted and thereby also stamp the next acknowledge of the interrupt.

4:0 Timestamp Interrupt Select (TSISEL) - This field selects the interrupt line (0 - 31) to timestamp.

96.3.17 Interrupt Assertion Timestamp Register

*Table 1883.*0x1n8 - ITSTMPASn - Interrupt Assertion Timestamp register N

31	0
TASSERTION	
0	
r	

31:0 Timestamp of Assertion (TASSERTION) - The current Timestamp Counter value is saved in this register when timestamping is enabled and the interrupt line selected by TSISEL is asserted.

96.3.18 Interrupt Acknowledge Timestamp Register

*Table 1884.*0x1nC - ITSTMPASn - Interrupt Acknowledge Timestamp register N

31	0
TACKNOWLEDGE	
0	
r	

31:0 Timestamp of Acknowledge (TACKNOWLEDGE) - The current Timestamp Counter value is saved in this register when timestamping is enabled, the Acknowledge Stamped (S2) field is '0', and the interrupt selected by TSISEL is acknowledged by a processor connected to the interrupt controller.

96.3.19 Processor N Boot Address Register

*Table 1885.*0x200 + 0x4* n - BADDRn - Processor n Boot Address register

31 28	27 26	20 19	16 15	3	2 1	0
BOOTADDR[31:3]					RES	AS
-					-	-
w					-	w

31:3 Entry point for booting up processor N, 8-byte aligned

2:1 Reserved (write 0)

0 Start processor immediately after setting address

96.3.20 Interrupt Map Register N

*Table 1886.*0x300 + IRQMAPn - Interrupt map register n

31 24	23 16	15 8	7 0
IRQMAP[n*4]	IRQMAP[n*4+1]	IRQMAP[n*4+2]	IRQMAP[n*4+3]
n.n	n.4+1	n.4+2	n.4+3
rw	rw	rw	rw

b+7 : b Interrupt map (IRQMAP) - If the core has been implemented to support interrupt mapping then the Interrupt map register at offset 0x300 + 4*n specifies the mapping for interrupt lines 4*n to 4*n+3. The bus interrupt line 4*n+x will be mapped to the interrupt controller interrupt line specified by the value of IRQMAP[n*4+x].

96.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x00D (same as IRQMP core). For description of vendor and device identifiers see GRLIB IP Library User's Manual.

96.5 Implementation

96.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

96.6 Configuration options

Table 1887 shows the configuration options of the core (VHDL generics).

*Table 1887.*Configuration options

Generic	Function	Allowed range	Default
pindex	Selects which APB select signal (PSEL) that will be used to access the interrupt controller	0 to NAPBSLV-1	0
paddr	The 12-bit MSB APB address	0 - 16#FFF#	0
pmask	The APB address mask. The mask determines the size of the memory area occupied by the core. The minimum required memory area based on the *nctrl* VHDL generic gives (nctrl : pmask) = 1 : 16#FFF#, 2: 16#FE0#, 3-4 : 16#FC0#, 5-8 : 16#F80#, 9-16 : 16#F00#. Note that even with nctrl = 1 the core may require a larger area than 256 bytes if the core has been implemented with support for timestamping and/or dynamic reset addresses.	0 - 16#FFF#	16#FFF#
ncpu	Number of processors in multiprocessor system	1 to 16	1
eirq	Enable extended interrupts	0 - 15	0
nctrl	Asymmetric multiprocessing system extension. This generic defines the number of internal interrupt controllers that will be implemented in the core.	1 - 16	1
tstamp	Interrupt timestamping. If this generic is non-zero the core will include a timestamp counter and *tstamp* set(s) of interrupt timestamp register(s).	0 - 16	0
wdogen	Enable watchdog inputs. If this generic is set to 1 the core will include logic to assert a selected interrupt when a watchdog input is asserted.	0 - 1	0
nwdog	Number of watchdog inputs	1 - 16	1
dynrstaddr	Deprecated feature, must be set to 0	0 - 0	0
rstaddr	Deprecated feature, must be set to 0	0 - (2^{20}-1)	0
extrun	Use external run vector. If this generic is set to 1 the start of processors after reset will be controlled via the input signal cpurun. If this generic is set to 0, CPU 0 will be started after reset and the other CPUs will be put in power-down mode. This requires that the SMP VHDL generic on the LEON entity is nonzero.	0 - 1	0
irqmap	Enable interrupt (re-)map registers. If irqmap is set to 0 then interrupt map functionality is disabled. If irqmap is nonzero then (irqmap + eirq) = 1 includes map registers for bus interrupt lines 0 - 15. If (irqmap + eirq) > 1 then interrupt map registers are available for bus interrupt lines 0 - 31. When irqmap = 3 then support for 64 interrupt sources is enabled and interrupt map registers are available for bus interrupt lines 0 - 63.	0 - 3	0

*Table 1887.*Configuration options

Generic	Function	Allowed range	Default
exttimer	Use external timer input *timer*. The external timer replaces the internal time stamp counter. The counter is no longer started based on the value of the IRQ sel field. Users must ensure that the timer input is toggling when-they want to do timestamps	0 - 1	0
bootreg	Enable boot address register and error mode register.	0 - 1	1

96.7 Signal descriptions

Table 1888 shows the interface signals of the core (VHDL ports).

*Table 1888.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
IRQI[n]	INTACK	Input	Processor *n* Interrupt acknowledge	High
	IRL[3:0]		Processor *n* interrupt level	High
	PWD		Processor *n* in power down mode	High
	FPEN		Unused	-
	ERR		Processor *n* in error mode	High
IRQO[n]	IRL[3:0]	Output	Processor *n* Input interrupt level	High
	RESUME		Reset power-down and error mode of processor *n*	High
	RSTRUN		Start processor *n* after reset (SMP systems only)	High
	RSTVEC[31:12]		Always zero	-
	INDEX[3:0]		CPU index	High
	PWDSETADDR		In power-down/error mode, shift PC to nPC and set PWDNEWADDR to PC.	High
	PWDNEWADDR [31:2]		New PC value used with PWDSETADDR	-
	FORCEERR		Force CPU into error mode	High
WDOG[]	N/A	Input	Watchdog input signals	High
CPURUN[]	N/A	Input	If position n in this vector is set to ‘1’, processor n will be started after reset. Otherwise processor n will go into power-down. This signal is only used if VHDL generic extrun is /= 0.	High
TIMER[]	N/A	Input	Timer value, used then VHDL generic exttimer /=0	-
RSTMAP	N/A	Input	Reset value for IRQ mapping register (only used when 64 interrupt sources is supported). Bit[4:0] = mapping for IRQ0 Bit[9:5] = mapping for IRQ1 ...	-

* see GRLIB IP Library User’s Manual

96.8 Library dependencies

Table 1889 shows libraries that should be used when instantiating the core.

*Table 1889.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	LEON3	Signals, component	Signals and component declaration

96.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.leon3.all;

entity irqamp_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    ...  -- other signals
    );
end;

architecture rtl of irqamp_ex is
  constant NCPU : integer := 4;

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
  signal ahbsi : ahb_slv_in_type;

  -- GP Timer Unit input signals
  signal irqi   : irq_in_vector(0 to NCPU-1);
  signal irqo   : irq_out_vector(0 to NCPU-1);

  -- LEON3 signals
  signal leon3i : l3_in_vector(0 to NCPU-1);
  signal leon3o : l3_out_vector(0 to NCPU-1);

begin

  -- 4 LEON3 processors are instantiated here
  cpu : for i in 0 to NCPU-1 generate
    u0 : leon3s generic map (hindex => i)
    port map (clk, rstn, ahbmi, ahbmo(i), ahbsi,
irqi(i), irqo(i), dbgi(i), dbgo(i));
  end generate;

  -- MP IRQ controller
  irqctrl0 : irqamp
  generic map (pindex => 2, paddr => 2, ncpu => NCPU, nctrl => NCPU)
  port map (rstn, clk, apbi, apbo(2), irqi, irqo);
end
```

97 L2C - Level 2 Cache controller

97.1 Overview

The L2C implements a Level-2 cache for processors with AHB interfaces. The L2C works as an AHB to AHB/AXI bridge, caching data that is read or written via the bridge. The cache is a unified cache and in a system with LEON processors, data may exist in the Level-1 and Level-2 cache, or only in the Level-1 or Level-2 cache. A front-side AHB interface is connected to the processor bus, while a backend AHB/AXI interface is connected to the memory bus. Both front-side and backend buses can be individually configured to 32, 64 or 128 bits data width. The front-side bus and the backend bus must be clocked with the same clock. Figure 256 shows a system block diagram for the cache controller.

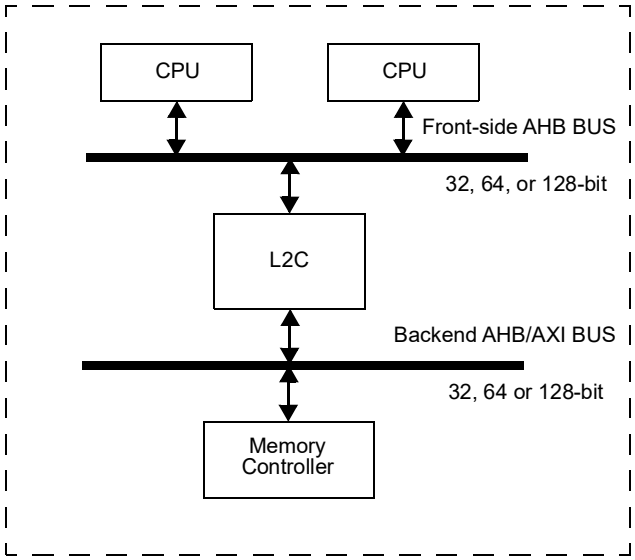


Figure 256. Block diagram

97.2 Configuration

The Level-2 cache can be configured as direct-mapped or multi-way with associativity 2, 3 or 4. The replacement policy for a multi-way configuration can be configured as: LRU (least-recently-used), pseudo-random or master-index (where the way to replace is determined by the master index). The way size is configurable between 1 - 512 KiB with a line size of 32/64 bytes.

97.2.1 Replacement policy

The core can implements three different replacement policies: LRU (least-recently-used), (pseudo-) random and master-index. The LRU replacement policy is configured as default. With master-index replacement policy, master 0 would replace way 1, master 1 would replace way 2, and so on. For master indexes corresponding to a way number larger than the number of implemented ways, there are two options to determine which way to replace. One option is to map all these master indexes to a specific way. This is done by specifying this way in the index-replace field in the control register and selecting this option in the replacement policy field, also located in the control register. It is not allowed to select a locked way in the index-replace field. The second option is to replace way = ((master index) modulus (number of ways)). This option can be selected in the replacement policy field, but is only allowed with multi-way associativity 2 or 4.

97.2.2 Write policy

The cache can be configured to operate as write-through or copy-back cache. Before changing the write policy to write-through, the cache has to be disabled and flushed (to write back dirty cache lines to memory). This can be done by setting the Cache disable bit when issuing a flush all command. The write policy is controlled via the cache control register. More fine-grained control can also be obtained by enabling the MTRR registers (see text below).

97.2.3 Memory type range registers

The memory type range registers (MTRR) are used to control the cache operation with respect to the address. Each MTRR can define an area in memory to be uncached, write-through or write-protected. The MTRR consist of a 14-bit address field, a 14-bit mask and two 2-bit control fields. The address field is compared to the 14 most significant bits of the cache address, masked by the mask field. If the unmasked bits are equal to the address, an MTRR hit is declared. The cache operation is then performed according to the control fields (see register descriptions). If no hit is declared or if the MTRR is disabled, cache operation takes place according to the cache control register. The number of MTRRs is configurable through the *mtrr* VHDL generic. When changing the value of any MTRR register, cache must be disabled and flushed (This can be done by setting the Cache disable bit when issue a flush all command).

Note that the write-protection provided via the MTRR registers is enforced even if the cache is disabled.

97.2.4 Cachability

The core uses a VHDL generic CACHED to determine which address range is cachable. Each bit in this 16-bit value defines the cachability of a 256 Mbyte address block on the AMBA AHB bus. A value of 16#00F3# will thus define cachable areas in 0 - 0x20000000 and 0x40000000 - 0x80000000. When the VHDL generic CACHED is 0, the cachable areas is defined by the plug&play information on the backend bus. When implemented with a AXI backend bus the cachability needs to be defined with the VHDL generic CACHED. The core can also be configured to use the HPROT signal to override the cachable area defined by VHDL generic CACHED. A access can only be redefined as non-cachable by the HPROT signal. See table 1890 for information on how HPROT can change the access cachability within a cachable address area. The AMBA AHB signal HPROT[3] defines the access cacheable when active high and the AMBA AHB signal HPROT[2] defines the access bufferable when active high.

Table 1890. Access cachability using HPROT.

HPROT:	non-cachable, non-bufferable	non-cachable, bufferable	cacheable
Read hit	Cache access*	Cache access	Cache access
Read miss	Memory access	Memory access	Cache allocation and Memory access
Write hit	Cache and Memory access	Cache access	Cache access
Write miss	Memory access	Memory access	Cache allocation

* When the HPROT-Read-Hit-Bypass bit is set in the cache control register this will generate a Memory access.

97.2.5 Cache tag entry

Table 1891 show the different fields of the cache tag entry for a cache with set size equal to 1 kbyte. The number of bits implemented is depending on the cache configuration.

Table 1891. L2C Cache tag entry

31	10	9	8	7	6	5	4	0
TAG		Valid		Dirty		RES	LRU	

31 : 10	Address Tag (TAG) - Contains the address of the data held in the cache line.
9 : 8	Valid bits. When set, the corresponding sub-block of the cache line contains valid data. Valid bit 0 corresponds to the lower 16 bytes sub-block (with offset 1) in the cache line and valid bit 1 corresponds to the upper 16 bytes sub-block (with offset 0) in the cache line.
7 : 6	Dirty bits When set, this sub-block contains modified data.
5	RESERVED
4 : 0	LRU bits

97.2.6 AHB address mapping

The AHB slave interface occupies three AHB address ranges. The first AHB memory bar is used for memory/cache data access. The address and size of this bar is configured via VHDL generics. The second AHB memory bar is used for access to configuration registers and the diagnostic interface. This bar has a configurable address via VHDL generic but always occupies 4 MiB in the AHB address space. The third AHB memory bar is used to map the ioarea of the backend AHB bus (to access the plug&play information on that bus, not supported when AXI backend is selected). The address and size of the this bar is configured via VHDL generics. The address is available in the user-defined register 1 of the configuration record.

97.2.7 Memory protection and Error handling

The ft VHDL generic enables the implementation of the Error Detection And Correction (EDAC) protection for the data and tag memory. One error can be corrected and two error can be detected with the use of a (39, 32, 7) BCH code. When implemented, the EDAC functionality can dynamically be enabled or disabled. Before being enabled the cache should be flushed. The dirty and valid bits fore each cache line is implemented with TMR. When EDAC error or backend AHB/AXI error or write-protection hit in a MTRR register is detected the error status register is updated to store the error type. The address which cause the error is also saved in the error address register. The error types is prioritised in the way that a uncorrected EDAC error will overwrite any other previously stored error in the error status register. In all other cases, the error status register has to be cleared before a new error can be stored. Each error type (correctable-, uncorrectable EDAC error, write-protection hit, backend AHB/AXI error) has a pending register bit. When set and this error is unmasked, a interrupt is generated. When uncorrectable error is detected in the read data the core will respond with a AHB error. AHB error response can also be enabled for a access that match a stored error in the error status register. Error detection is done per cache line. The core also provide a correctable error counter accessible via the error status register.

*Table 1892.*Cache action on detected EDAC error

Access/Error type	Cache-line not dirty	Cache-line dirty
Read, Correctable Tag error	Tag is corrected before read is handled, Error status is updated with a correctable error.	Tag is corrected before read is handled, Error status is updated with a correctable error.
Read, Uncorrectable Tag error	Cache-line invalidated before read is handled, Error status is updated with a correctable error.	Cache-line invalidated before read is handled, Error status is updated with a uncorrectable error. Cache data is lost.
Write, Correctable Tag error	Tag is corrected before write is handed, Error status is updated with a correctable error.	Tag is corrected before write is handled, Error status is updated with a correctable error.
Write, Uncorrectable Tag error	Cache-line invalidated before write is handled, Error status is updated with a correctable error.	Cache-line invalidated before write is handled, Error status is updated with a uncorrectable error. Cache data is lost.
Read, Correctable Data error	Cache-data is corrected and updated, Error status is updated with a correctable error. AHB access is not affected.	Cache-data is corrected and updated, Error status is updated with a correctable error. AHB access is not affected.
Read, Uncorrectable Data error	Cache-line is invalidated, Error status is updated with a correctable error. AHB access is terminated with retry.	By default the cache-line is NOT invalidated (this can be configured by bit[10] EDI in the Error Handling / Injection configuration register), Error status is updated with a uncorrectable error. AHB access is terminated with error.
Write (<32-bit), Correctable Data error	Cache-data is corrected and updated, Error status is updated with a correctable error. AHB access is not affected.	Cache-data is corrected and updated, Error status is updated with a correctable error. AHB access is not affected.
Write (<32-bit), Uncorrectable Data error	Cache-line is re-fetched from memory, Error status is updated with a correctable error. AHB access is not affected.	Cache-line is invalidated, Error status is updated with a uncorrectable error. AHB access write data and cache data is lost.

97.2.8 Scrubber

When EDAC protection is implemented a cache scrubber is enabled. The scrubber is controlled via two register in the cache configuration interface. To scrub one specific cache line the index and way of the line is set in the scrub control register. To issue the scrub operation, the pending bit is set to 1. The scrubber can also be configured to continuously loop through and scrub each cache line by setting the enabled bit to 1. In this mode, the delay between the scrub operation on each cache line is determine by the scrub delay register (in clock cycles).

97.2.9 Locked way

One or more ways can be configured to be locked (not replaced). The number of way that should be locked is configured by the locked-way field in the control register. The way to be locked is starting with the uppermost way (for a 4-way associative cache way 4 is the first locked way, way 3 the second, and so on). After a way is locked, this way has to be flushed with the “way flush” function to update the tag match the desired locked address. During this “way flush” operation, the data can also be fetched from memory.

97.2.10 Data priming

Data can be loaded from one or two address ranges. Before triggering the priming operation, the start and stop address need to be configured. To specify if one or both address ranges should be loaded the respective enable bit (PSTART0/1.EN) need to be set. To trigger the operation, the pending bit (PSTART0.P) needs to be set to '1'. If only one address range should be loaded, the first set of priming register (PSTART0, PSTOP0) should be used. The cache lines are loaded from the start address to the

stop address. When two address ranges should be loaded, one cache line from each area is loaded before moving to the next line in the address range. If the cache already contains the cache line specified to be loaded, the priming operation moves to the next line in the priming address range without fetching any data from memory but the LRU replacement information is updated.

97.3 Operation

97.3.1 Read

A cachable read access to the core results in a tag lookup to determine if the requested data is located in the cache memory. For a hit (requested data is in the cache) the data is read from the cache and no read access is issued to the memory. If the requested data is not in the cache (cache miss), the cache controller issue a read access to the memory controller to fetch the cache line including the requested data. The replacement policy determine which cache line in a multi-way configuration that should be replaced and its tag is updated. If the replaced cache line is modified (dirty) this data is stored in a write buffer and after the requested data is fetched from memory the replaced cache line is written to memory.

For a non-cachable read access to the cache, the cache controller can issue a single read access or a burst read access to fetch the data from memory. The access type is determine by how the cache is configured regarding hprot support and bypass line fetch in the access control register. The data is stored in a read buffer and the state of the cache is not modified in any way.

The cache will insert wait-states until the read access is determined to be a cache hit or miss. For a cache hit the data is then delivered. For a miss the cache can insert wait-states during the memory fetch or issue a AMBA SPLIT (depending on how the cache is configured). AMBA SPLIT response is only implemented in version 3 of the core. AMBA SPLIT is disabled for accesses to the L2C register interface, wait-states are inserted.

97.3.2 Write

A cachable write access to the core results in a tag lookup to determine if the cache line is located in the cache. For a hit the cache line is updated. No access is issued to the memory for a copy-back configuration. When the core is configured as a write-through cache, each write access is also issued towards the memory. For a miss, the replacement policy determines which cache line in a multi-way configuration that should be replaced and updates its tag. If the replaced cache line is dirty, this is stored in a write buffer to be written back to the memory. The new cache line is updated with the data from the write access and for a non 128-bit access the rest of the cache line is fetched from memory. Last (when copy-back policy is used and the replaced cache line was marked dirty) the replaced cache line is written to memory. When the core is configured as a write-through cache, no cache lines are marked as dirty and no cache line needs to be written back to memory. Instead the write access is issued towards the memory as well. A new cache line is allocated on a miss for a cacheable write access independent of write policy (copy-back or write-through).

For a non-cachable write access to the core, the data is stored in a write buffer and the cache controller issue single write accesses to write the data to memory. The state of the cache is unmodified during this access.

The cache can accept a non sub-word write hit access every clock cycle. When the cache is unable to accept a new write access the cache inserts wait-states or issue a AMBA SPLIT response depending on how the cache is configured. AMBA SPLIT response is only implemented in version 3 of the core. AMBA SPLIT is disabled for accesses to the L2C register interface, wait-states are inserted.

97.3.3 Cache flushing

The cache can be flushed by accessing a cache flush register. There is three flushing modes: invalidate (reset valid bits), write back (write back dirty cache lines to memory, but no invalidation of the cache content) and flush (write back dirty cache lines to memory and invalidate the cache line). The flush

command can be applied to the entire cache, one way or to only one cache line. The cache line to be flushed can be addresses in two ways: direct address (specify way and line address) and memory address (specify which memory address that should be flushed in the cache. The controller will make a cache lookup for the specified address and on a hit, flush that cache line). When the entire cache is flushed the Memory Address field should be set to zero. Invalidate a cache line takes 5 clock cycles. If the cache line needs to be written back to memory one additional clock cycle is needed plus the memory write latency. When the whole cache is flushed the invalidation of the first cache line takes 5 clock cycles, after this one line can be invalidate each clock cycle. When a cache line needs to be written back to memory this memory access will be stored in a access buffer. If the buffer is full the invalidation of the next cache line is stall until a slot in the buffer has opened up. If the cache also should be disabled after the flush is complete, it is recommended to set the cache disable bit together with the flush command instead of writing '0' to the cache enable bit in the cache control register.

Note that after a processor (or any other AHB master) has initiated a flush the processor is not blocked by the flush unless it writes or requests data from the Level-2 cache. The cache blocks all accesses (responds with AMBA SPLIT or wait-states depending on cache configuration) until the flush is complete. AMBA SPLIT response is only implemented in version 3 of the core.

97.3.4 Disabling Cache

To be able to safely disable the cache when it is being accessed, the cache need to be disabled and flushed at the same time. This is accomplished by setting the cache disable bit when issue the flush command.

97.3.5 Diagnostic cache access

The diagnostic interface can be used for RAM block testing and direct access to the cache tag, cache data content and EDAC check bits. The read-check-bits field in the error control register selects if data content or the EDAC check bits should be read out. On writes, the EDAC check bits can be selected from the data-check-bit or tag-check-bit register. These register can also be XOR:ed with the correct check bits on a write. See the error control register for how this is done.

97.3.6 Error injection

Except using the diagnostic interface, the EDAC check bits can also be manipulated on a regular cache access. By setting the xor-check-bit field in the error control register the data EDAC check bits will be XOR:ed with the data-check-bit register on the next write or the tag EDAC check bits will be XOR:ed with the tag-check-bit register on the next tag replacement. The tag check bit manipulation is only done if the tag-check-bit register is not zero. The xor-check-bit is reset on the next tag replacement or data write. Error can also be injected by writing a address (or way, index, offset) together with the inject bit to the "Error injection" register. This will XOR the check-bits (or tag/data) for the specified address (or cache-line) with the tag-/data-check-bit register depending on the Error injection configuration register. If the specified address in not cached, the cache content will be unchanged.

97.3.7 AHB slave interface

The slave interface is the core's connection to the CPU and the level 1 cache. The core can accept 8-bit(byte), 16-bit(half word), 32-bit(word), 64-bit, and 128-bit single accesses and also 32-bit, 64-bit, and 128-bit burst accesses. For an access during a flush operation, the core will respond with a AHB SPLIT response or with wait-states. For a uncorrectable error or a backend AHB error on a read access, the core will respond with a AMBA ERROR response. AMBA SPLIT response is only implemented in version 3 of the core.

97.3.8 AHB master interface

The master interface is the core's connection to the memory controller. During cache line fetch, the controller can issue either a 32-bit, 64-bit or 128-bit burst access. For a non cachable access and in

write-through mode the core can also issue a 8-bit(byte), 16-bit(half word), 32-bit(word), 64-bit, or 128-bit single write access. The bbuswidth VHDL generic controls the maximum bus access size on the master interface in the “wide-bus” address range. If set to 128 (default), the largest access will be 128-bit. If set to 64, the largest access will be 64-bit. If set to 32, the largest access will be 32-bit. The “wide-bus address range is defined by the wbmask VHDL generic. Each bit in this 16-bit value represents a 256 Mbyte address block on the AMBA AHB bus. The cache will only generate wide accesses (> 32-bit) to address ranges which wbmask bit is ‘1’. For address ranges which wbmask bit is ‘0’, wide accesses will be translated to 32-bit bursts.

The HBURST value during burst accesses will correspond to SINGLE, INCR, INCR4, INCR8 or INCR16, depending on burst type and AHB data bus width.

97.3.9 AXI master interface

AXI master interface supports data width of 32, 64, or 128-bit, configured with the GRLIB configuration option AXIDW. The interface supports both AXI3/AXI4 bus interfaces (no AXI4 specific operation is performed by the master interface). The value of the AXI AxCACHE can be configured via a configuration register (L2CACCC). When the cache need to fetch a cache line from memory and write a cache line to memory, both the read and write operation are started at the same time. The AXI master interface supports only one outstanding read and one outstanding write access. The burst length depends on the bus width and cache line size. Minimum burst size is 16 Bytes (one half 32 Bytes cache line) and the maximum burst size is 64 Bytes (one 64 Bytes cache line).

97.3.10 Cache status

The cache controller has a status register which provide information on the cache configuration (multi-way configuration and set size). The core also provides an access counter and a hit counter via AHB mapped registers. These register can be used to calculate hit rate. The counters increments for each data access to core (i.e. a burst access is only counted as one access). When writing 0 to the access counter, the internal access/hit counters is cleared and its value is loaded to the registers accessible via the AHB interface. In wrapping mode both counters will be cleared when the access counter is wrapping at 0xFFFFFFFF. In shifting mode both counters will be shifted down 16 bits when the access counter reach 0xFFFFFFFF. In this mode the accessible counter registers is updated automatically when the access counter’s 16 LSB reach the value of 0xFFFF.

The core can also implement a front-side bus usage counter. This counter records every clock cycle the bus is not in idle state. The registers accessible via the AHB interface is updated in the same way as for the hit counter registers. Writing 0 to the bus cycle counter register resets the bus usage counters. This counter also has a wrapping and shifting mode similar to the hit counter.

In addition to the counter registers, the core also provide output signals for: cache hit, cache miss, and cache access. These signals can be connected to external statistic counters.

97.3.11 Endianness

The core is designed for big-endian systems.

97.4 Registers

The core is configured via registers mapped into the AHB memory address space. Only 32-bit single-accesses to the registers are supported.

*Table 1893.*L2C: AHB registers

AHB address offset	Register
0x00	Control register
0x04	Status register
0x08	Flush (Memory address)
0x0C	Flush (set, index)
0x10	Access counter
0x14	Hit counter
0x18	Bus cycle counter
0x1C	Bus usage counter
0x20	Error status/control
0x24	Error address
0x28	TAG-check-bit (Tag-bit)
0x2C	Data-check-bit (Data-bit)
0x30	Scrub Control/Status
0x34	Scrub Delay

*Table 1893.*L2C: AHB registers

AHB address offset	Register
0x38	Error injection
0x3C	Access control (Only available in version 3 of the core)
0x40	Priming start (first area)
0x44	Priming stop (first area)
0x48	Priming start (second area)
0x4C	Priming stop (second area)
0x50	Error handling / injection configuration
0x80 - 0xFC	MTRR registers
0x80000 - 0xFFFFC	Diagnostic interface (Tag) 0x80000: Tag 1, way-1 0x80004: Tag 1, way-2 0x80008: Tag 1, way-3 0x8000C: Tag 1, way-4 0x80010: Tag check-bits way-0,1,2,3 (Read only) bit[31] = RESERVED bit[30:24] = check-bits for way-1. bit[23] = RESERVED bit[22:16] = check-bits for way-2. bit[15] = RESERVED bit[14:8] = check-bits for way-3. bit[7] = RESERVED bit[6:0] = check-bits for way-4. 0x80020: Tag 2, way-1 0x80024: ...
0x200000 - 0x3FFFFC	Diagnostic interface (Data) 0x200000 - 0x27FFFC: Data or check-bits way-1 0x280000 - 0x2FFFFF: Data or check-bits way-2 0x300000 - 0x27FFFC: Data or check-bits way-3 0x380000 - 0x3FFFFF: Data or check-bits way-4 When check-bits are read out: Only 32-word at offset 0x0, 0x10, 0x20,... are valid check-bits. bit[31:28] = RESERVED bit[27:21] = check-bits for data word at offset 0x0. bit[20:14] = check-bits for data word at offset 0x4. bit[13:7] = check-bits for data word at offset 0x8. bit[6:0] = check-bits for data word at offset 0xc.

97.4.1 Control Register

*Table 1894.*0x00 - L2CC - L2C Control register

31	29	28	27	19	18	16	15	12	11	8	7	6	5	4	3	2	1	0
EN	EDAC	REPL	RESERVED		BBS		INDEX-WAY		LOCK		RES		HPRHB	HPB	UC	HC	WP	HP
0	0	0	0		-		0		0		0		0	0	0	0	0	0
rw	rw	rw	r		rw		rw		rw		r		rw	rw	rw	rw	rw	rw

31 Cache enable (EN) - When set, the cache controller is enabled. When disabled, the cache is bypassed.

30 EDAC enable (EDAC)

29: 28 Replacement policy (REPL) -
00: LRU
01: (pseudo-) random
10: Master-index using index-replace field
11: Master-index using the modulus function

27: 19 RESERVED

18: 16 Backend bus size configuration (BBS) -
“100”: Configure backend bus size to 128-bit.
“011”: Configure backend bus size to 64-bit.
“010”: Configure backend bus size to 32-bit.
“000”: No configuration update is done.
Other values: not supported.

15: 12 Master-index replacement (INDEX-WAY) - Way to replace when Master-index replacement policy and master index is larger than number of ways in the cache.

11: 8 Locked ways (LOCK) - Number of locked ways.

7: 6 RESERVED

5 HPROT read hit bypass (HPRHB) - When set, a non-cacheable and non-bufferable read access will bypass the cache on a cache hit and return data from memory. Only used with HPROT support.

4 HPROT bufferable (HPB) - When HPROT is used to determine cachability and this bit is set, all accesses is marked bufferable.

3 Bus usage status mode (UC) - 0 = wrapping mode, 1 = shifting mode.

2 Hit rate status mode (HC) - 0 = wrapping mode, 1 = shifting mode.

1 Write policy (WP) - When set, the cache controller uses the write-through write policy. When not set, the write policy is copy-back.

0 HPROT enable (HP) - When set, use HPROT to determine cachability.

97.4.2 Status Register

*Table 1895.*0x04 - L2CS - L2C Status register

31	25	24	23	22	21	16	15	13	12	2	1	0
RESERVED	DP	LS	AT	MP	MTRR		BBUS-W		WAY-SIZE		WAY	
0	*	*	*	*	*		1		*		*	
r	r	r	r	r	r		r		r		r	

31: 26 RESERVED

25 Data priming (DP) - 1 = supported.

24 Cache line size (LS) - 1 = 64 bytes, 0 = 32 bytes.

23 Access time (AT) - Access timing is simulated as if memory protection is implemented

22 Memory protection (MP) - implemented

21: 16 Memory Type Range Registers (MTRR) - Number of MTRR registers implemented

15: 13 Backend bus width (BBUS-W) 1 = 128-bit, 2 = 64-bit, 4 = 32-bit

Table 1895. 0x04 - L2CS - L2C Status register

12: 2		Cache way size (WAY-SIZE) - Size in kBytes
1: 0		Multi-Way configuration (WAY)
	“00“: Direct mapped “01“: 2-way “10“: 3-way “11“: 4-way	

97.4.3 Flush (Memory Address) Register

*Table 1896.*0x08 - L2CFMA - L2C Flush (Memory address) register

31 5	4	3	2 0
Memory Address (ADDR)	R	DI	FMODE
NR	0	0	0
rw	r	w	rw

31: 5 Memory Address (ADDR) - (For flush all cache lines, this field should be set to zero)

4 RESERVED

3 Cache disable (DI) - Setting this bit to '1' is equal to setting the Cache enable bit to '0' in the Cache Control register

2: 0 Flush mode (FMODE) -
"001": Invalidate one line, "010": Write-back one line, "011": Invalidate & Write-back one line.
"101": Invalidate all lines, "110": Write-back all lines, "111": Invalidate & Write-back all lines.
Only dirty cache lines are written back to memory.

97.4.4 Flush (Set, Index) Register

*Table 1897.*0x0C - L2CFSI - L2C Flush (Set, Index) register

31 16	10	9	8	7	6	5 4	3	2	1 0
INDEX / TAG		FL	VB	DB	R	WAY	DI	WF	FMODE
NR		0	0	0	0	0	0	0	0
rw		rw	rw	rw	r	rw	w	rw	rw

31: 16 Cache line index (INDEX) - used when a specific cache line is flushed

31: 10 (TAG) - used when "way flush" is issued. If a specific cache line is flushed, bit[15:10] should be set to zero. When a way flush is issued, this field will be used as the TAG for the selected cache way.

9 Fetch Line (FL) - If set to '1' data is fetched form memory when a "way flush" is issued. If a specific cache line is flushed, this bit should be set to zero

8 Valid bit (VB) - used when "way flush" is issued. If a specific cache line is flushed, this bit should be set to zero.

7 Dirty bit (DB) - used when "way flush" is issued. If a specific cache line is flushed, this bit should be set to zero

6 RESERVED

5: 4 Cache way (WAY) -

3 Cache disable (DI) - Setting this bit to '1' is equal to setting the Cache enable bit to '0' in the Cache Control register.

2 Way-flush (WF) - When set one way is flushed, If a specific cache line should be flushed, this bit should be set to zero

1: 0 Flush mode (FMODE) -

line flush:
"01": Invalidate one line
"10": Write-back one line (if line is dirty)
"11": Invalidate & Write-back one line (if line is dirty).

way flush:
"01": Update Valid/Dirty bits according to register bit[8:7] and TAG according to register bits[31:10]
"10": Write-back dirty lines to memory
"11": Update Valid/Dirty bits according to register bits [8:7] and TAG according to register bits[31:10], and Write-back dirty lines to memory.

97.4.5 Access Counter Register

*Table 1898.*0x10 - L2CACC - Access counter register

31	0
Access counter	
0	
wc	

31 : 0 Access counter. Write 0 to clear internal access/hit counter and update access/hit counter register.

97.4.6 Hit Counter Register

*Table 1899.*0x14 - L2CHIT - Hit counter register

31	0
Hit counter	
0	
wc	

31 : 0 Hit counter.

97.4.7 Front-side Bus Cycle Counter Register

*Table 1900.*0x18 - L2CFSCCNT - Front-side bus cycle counter register

31	0
Bus cycle counter	
0	
wc	

31 : 0 Bus cycle counter. Write 0 to clear internal bus cycle/usage counter and update bus cycle/usage counter register.

97.4.8 Front-side Bus Usage Counter Register

*Table 1901.*0x1C - L2CFSUCNT - Front-side bus usage counter register (address offset 0x1C)

31	0
Bus usage counter	
0	
wc	

31 : 0 Bus usage counter.

97.4.9 Error Status/Control

*Table 1902.*0x20 - L2CERR - L2CError status/control register

31 28	27	26 24	23	22	21	20	19	18 16	15 12	11 8	7 6	5 4	3	2	1	0
AHB master index	SCRUB	TYPE	TAG/DATA	COR/UCOR	MULTI	VALID	DISERESP	Correctable error counter	IRQ pending	IRQ mask	Select CB	Select TCB	XCB	RCB	COMP	RST
NR	NR	NR	NR	NR	NR	NR	0	NR	NR	0	0	0	0	0	0	0
r	r	r	r	r	r	r	rw	r	r	rw	rw	rw	rw	rw	rw	w

31: 28 AHB master that generated the access

27 Scrub error (SCRUB) - Indicates that the error was trigged by the scrubber.

Table 1902. 0x20 - L2CERR - L2CError status/control register

26: 24	Access/Error Type: (TYPE) - 000: cache read, 001: cache write, 010: memory fetch, 011: memory write, 100: Write-protection hit, 101: backend read AHB error, 110: backend write AHB error
23	Tag or data Error - 0 tag error, 1: data error
22	Correctable or uncorrectable error - 0: correctable error, 1: uncorrectable error
21	Multiple error (MULTI) - set when multiple errors has been detected.
20	Error status valid (VALID) - register contains valid error status.
19	Disable error responses for uncorrectable EDAC error (DISERESP).
18: 16	Correctable error counter
15: 12	Interrupt pending bit3: Backend AHB error bit2: Write-protection hit bit1: Uncorrectable EDAC error bit0: Correctable EDAC error
11: 8	Interrupt mask (if set this interrupt is unmasked) bit3: Backend AHB error bit2: Write-protection hit bit1: Uncorrectable EDAC error bit0: Correctable EDAC error
7: 6	Selects (CB) - data-check-bits for diagnostic data write: 00: use generated check-bits 01: use check-bits in the data-check-bit register 10: XOR check-bits with the data-check-bit register 11: use generated check-bits
5: 4	Selects (TCB) - tag-check-bits for diagnostic tag write: 00: use generated check-bits 01: use check-bits in the tag-check-bit register 10: XOR check-bits with the tag-check-bit register 11: use generated check-bits
3	Xor check-bits (XOR) - If set, the check-bits for the next data write or tag replace will be XOR:ed withe the check-bit register. Default value is 0.
2	Read check-bits (RCB) - If set, a diagnostic read to the cache data area will return the check-bits related to that data.When this bit is set, check bits for the data at offset 0x0 - 0xc can be read at offset 0x0, the check bits for data at offset 0x10 - 0x1c can be read at offset 0x10, ...
1	Compare error status (COMP) - If set, a read access matching a uncorrectable error stored in the error status register will generate a AHB error response. Default value is 0.
0	Resets (RST) - clear the status register to be able to store a new error. After power up the status register needs to be cleared before any valid data can be read out. Writing ‘1’ to this bit clears: Bit[21] Multiple error Bit[20] Error status valid Bit[18:16] Correctable error counter Bit[15:12] Interrupt pending

97.4.10 Error Address Register

*Table 1903.*0x24 - L2CERRA - L2C Error address register

31	0
Error address (EADDR)	
NR	
r	

31 : 0 Error address (EADDR). Depending on which type of access (AHB access or scrubber access) that triggered the error, the address has different formats.
- When an AHB access triggers the error this register contains the address of the access. Bit[3:0] may be cleared.
- When the scrubber triggers the error this register contains the index and way of the cache line where the error was detected. The format is: [0..0 & index & 0..0 & way]. The bit rage of the index depends on the cache line size (lower limit of index) and cache way size (upper limit of index). The way field is defined as: 0001 for way 1, 0010 for way 2, 0100 for way 3, and 1000 for way 4.

97.4.11 Tag-check-bit Register

*Table 1904.*0x28 - L2CTCB - L2C TAG-Check-Bits (or TAG-bit) register (When injecting in Check-bits)

31 7	6 0
RESERVED	TCB
0	0
r	rw

31 : 7 RESERVED

6 : 0 TAG Check-bits (TCB) - Check-bits which can be selected by the “Select check-bit“ field in the error status/control register for TAG updates

*Table 1905.*0x28 - L2CTCB - L2C TAG-Check-Bits (or TAG-bit) register (When injecting in TAG via the error injection register)

31	0
TAG	
0	
rw	

31 : 0 TAG (TAG) - When configured to inject error in the TAG, the TAG will be xor:ed with this register to inject error when the error injection register is used. Depending on the cache configuration (size) some of the lower bits may be unused.

97.4.12 Data-check-bit Register

*Table 1906.*0x2C - L2CCB - L2C Data-Check-Bits (or Data-Bits) register (When injecting in check-bits)

31 28	27 0
RESERVED	DCB
0	0
r	rw

31 : 28 RESERVED

27 : 0 Data Check-bits (DCB) - Check-bits which can be selected by the “Select check-bit“ field in the error status/control register for cache data updates
bit[27:21] = check-bits for data word at offset 0x0.
bit[20:14] = check-bits for data word at offset 0x4.
bit[13:7] = check-bits for data word at offset 0x8.
bit[6:0] = check-bits for data word at offset 0xc.

*Table 1907.*0x2C - L2CCB - L2C Data-Check-Bits (or Data-Bits) register (When injecting in data via the error injection register)

31 0
DATA
0
rw

27 : 0 Data (DATA) - When configured to inject error in the cache data, the data will be xor:ed with this register to inject error when the error injection register is used.

97.4.13 Scrub Control/Status Register

*Table 1908.*0x30 - L2CSCRUB - L2C Scrub control/status register

31 16	15 6	5 4	3 2	1	0
INDEX	RESERVED	WAY	RES	PEN	EN
0	0	0	0	0	0
rw	r	rw	r	rw	rw

31: 16 Scrub Index (INDEX) - Index for the next line scrub operation

15: 6 RESERVED

5: 4 Scrub Way (WAY) - Way for the next line scrub operation

3: 2 RESERVED

1 Scrub Pending (PEN) - Indicates when a line scrub operation is pending. When the scrubber is disabled, writing '1' to this bit scrubs one line.

0 Scrub Enable (EN) - Enables / disables the automatic scrub functionality.

97.4.14 Scrub Delay Register

*Table 1909.*0x34 - L2CSDEL - L2C Scrub delay register

31 16	15 0
RESERVED	DEL
0	0
r	rw

31: 16 RESERVED

15: 0 Scrub Delay (DEL) - Delay the scrubber waits before issue the next line scrub operation

97.4.15 Error Injection Register

*Table 1910.*0x38 - L2CEINJ0 - L2C Error injection register (Mode 0)

31 2	1	0
ADDR	R	INJ
0	0	0
rw	r	rw

31: 2 Error Inject address (ADDR). ADDR specify address bits[31:2]. Address bit[1:0] is set to zero (unused by the error injection functionality).

1: RESERVED

0 Inject error (INJ) - Set to '1' to inject a error at "address".

*Table 1911.*0x38 - L2CEINJ1 - L2C Error injection register (Mode 1)

31 28	27 5	4 2	1	0
WAY	INDEX	OFFSET	R	INJ
0	0	0	0	0
rw	rw	rw	r	rw

31: 28 Error Inject cache-line way (WAY)

27: 5 Error Injection cache-line Index (INDEX)

4: 2 Error Injection cache-line offset (OFFSET)

1: RESERVED

0 Inject error (INJ) - Set to '1' to inject a error at "address".

97.4.16 Access control register

*Table 1912.*0x3C - L2CACCC - L2C Access control register

31 16	15		12	11	10	9	8	7	6	5	4	3	2	1	0
AxCACHE	RES	DSC	SH	RFCL	PS	SPLITQ	NHM	BERR	OAPM	FLINE	DBPF	128WF	R	DBPWS	SPLIT	R
0xFFEE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	rw	rw	r

31: 16 AXI CACHE configuration (AxCACHE) - (only available when AXI backend bus is implemented)
Bit[31:28]: ARCACHE (used when the cache fetches a cache line from memory)
Bit[27:24]: AWCACHE (used when the cache writes back a cache line to memory)
Bit[23:20]: ARCACHE (used for accesses that bypass the cache and reads from memory)
Bit[19:16]: AWCACHE (used for accesses that bypass the cache and writes to memory)

15 RESERVED

14 Disable cancellation and reissue of scrubber operation (DSC) - When set to ’0’, a write access to the same index as an ongoing scrubber operation will cancel and reissue the scrubber operation. When set to ’1’ the scrubber operation will complete without detection of the write access.

13 Scrubber hold (SH) - When set to ’1’ the cache will delay any new access until the current scrubber operation is complete.

12 Replace full cache line (RFCL) - When set and dirty cache line is updated, the entire cache line is written back to memory.

11 Priming statistic (PS) - When set, priming operation is included in the access/hit/miss statistics.

10 SPLIT queue write order (SPLITQ) When set, all write accesses (except locked) will be placed in the split queue when the split queue is not empty

9 No hit for cache misses (NHM) - When set, the unsplited read access for a read miss will not trig the access/hit counters.

8 Bit error status (BERR) - When set, the error status signals will represent the actual error detected rather then if the error could be corrected by refetching data from memory.

7 One access/master (OAPM) - When set, only one ongoing access per master is allowed to enter the cache. A second access would receive a SPLIT response

6 (FLINE) - When set, a cache line fetched from memory can be replaced before it has been read out by the requesting master.

5 Disable bypass prefetching (DBPF) - When set, bypass accesses will be performed as single accesses towards memory.

4 128-bit write line fetch (128WF) - When set, a 128-bit write miss will fetch the rest of the cache from memory.

3 RESERVED

2 Disable wait-states for discarded bypass data (DBPWS) - When set, split response is given to a bypass read access which data has been discarded and needs to refetch data from memory.

1 Enabled SPLIT response (SPLIT) - When set the cache will issue a AMBA SPLIT response on cache miss

0 RESERVED

97.4.17 Priming start register 0

*Table 1913.*0x40 - PSTART0 - L2C priming start register

31 5	4 1	0
ADDR	RES	P	EN
0	0	0	0
rw	r	rw	rw

31: 2 Priming start address (ADDR)

4: 2 RESERVED

Table 1913. 0x40 - PSTART0 - L2C priming start register

1	Priming access pending (P) - To start the priming operation this bit and the priming enable bit need to be set to '1'.
0	Priming enable (EN) - This indicates that the first area (defined by PSTART0.ADDR to PSTOP0.ADDR) should be primed.

97.4.18 Priming stop register 0

Table 1914. 0x40 - PSTOP0 - L2C priming stop register

31 5	4 0
ADDR	RES
0	0
rw	r

31: 5 Priming stop address (ADDR)

4: 0 RESERVED

97.4.19 Priming start register 1

Table 1915. 0x48 - PSTART1 - L2C priming start (second area) register

31 5	4 1	0
ADDR	RES	P	EN
0	0	0	0
rw	r	r	rw

31: 2 Priming start address (ADDR)

4: 2 RESERVED

1 Priming access pending (P) - This bit is read only and indicates that a priming operation on the second priming area is executing.

0 Priming enable (EN) - This indicates that the first area (defined by PSTART1.ADDR to PSTOP1.ADDR) should be primed.

97.4.20 Priming stop register 1

Table 1916. 0x4C - PSTOP1 - L2C priming stop (second area) register

31 5	4 0
ADDR	RES
0	0
rw	r

31: 5 Priming stop address (ADDR)

4: 0 RESERVED

97.4.21 Error Handling / Injection configuration

Table 1917. 0x50 - L2CEINJCFG - L2C injection configuration register

31 11	10	9	8	7 4	3			0
RESERVED	EDI	TER	IMD	RES	M	PI	DT	CB
0	0	0	0	0	0	0	0	0
r	rw	rw	rw	r	rw	rw	rw	rw

31: 11 RESERVED

10 (EDI) - Enable invalidation off cache line with un-correctable data error.
When set to 1 and a un-correctable data error is detected, the cache line will be invalidated (removing the error form the cache).

9 (TER) - Disable error response on un-correctable TAG error detection.
When set to 0 the access detecting a un-correctable TAG error would generate a AMBA error response. When set to 1 this access would not generate an error response.

Table 1917. 0x50 - L2CEINJCFG - L2C injection configuration register

8	(IMD) - Disable index match only after un-correctable TAG error. When set to 1 the TAG and INDEX are matched against the error address register after a detected un-correctable TAG error. When set to 0 only the INDEX are matched against the error address register.
7: 4	RESERVED
3	Error Injection Mode (M) 0: (Mode0) Error injection register layout is defined as a address. 1: (Mode1) Error injection register layout is defined as way, index, offset.
2	Prevent Error Injection on error (PI) 0: Ignore already existing error in cache-line 1: Prevent error injection when error exists in cache-line
1	Data/TAG Error injection (DT) 0: Inject error in data 1: Inject error in TAG
0	Check-Bit Error Injection (CB) 0: Inject error in check-bits 1: Inject error in data or TAG

97.4.22 Memory Type Range Register

Table 1918. 0x80-FC - L2CMTRR - L2C Memory type range register

31 18	17 16	15 2	1	0
ADDR	ACC	MASK	WP	AC
0	0	0	0	0
rw	rw	rw	rw	rw

31: 18	Address field (ADDR) - to be compared to the cache address [31:18]
17: 16	Access field (ACC) - 00: uncached, 01: write-through
15: 2	Address mask (MASK) - Only bits set to 1 will be used during address comparison
1	Write-protection (WP) - 0: disabled, 1: enabled
0	Access control field (AC) -. 0: disabled, 1: enabled

97.5 Core versions

The L2 cache controller exists in several different versions. The latest version is v3 and this is the default version used in GRLIB. For existing users of version 2 that want to continue with this version, the GRLIB design Makefile can be modified to add:

```
DIRSKIP=l2cache/v3
DIRADD=l2cache/v2
```

Performing the modification above will mean that the scripts will be generated to use version 2 of the Level-2 cache.

97.6 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x04B. For description of vendor and device identifier see GRLIB IP Library User's Manual

97.7 Implementation

97.7.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers if the GRLIB config package setting *grlib_async_reset_enable* is set.

97.7.2 RAM usage

The L2C uses single-port RAM to implement both cache tags and data memory. The tags are implemented using the SYNCRAM core, with the width and depth depending on the cache size configuration. The data memory is implemented using the SYNCRAM_128BW or SYNCRAM_156BW core, which is a 128-bit or 156-bit wide RAM wrapper with byte enables. The SYNCRAM_156BW is used when memory protection (EDAC) is implemented. For multi-way caches, each way's tag is implemented with a separate SYNCRAM block. The data memory can be implemented with separate SYNCRAM_128BW/156BW cores, or merged into the same SYNCRAM_128BW/156BW if the ARCH generic is set to 1. This will reduce the number of SYNCRAM_128BW/156BW core in multi-ways cache to one. The valid/dirty bits are stored in a SYNCRAM_2PFT core.

97.7.3 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

97.8 Configuration options

Table 1919 shows the configuration options of the core (VHDL generics).

*Table 1919.*Configuration options

Generic name	Function	Allowed range	Default
AHB slave interface			
hmstndx	AHB master index.	0 - NAHBMST-1	0
hslvndx	AHB slave index.	0 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR (for data access).	0 - 16#FFF#	16#F00#
hmask	MASK field of the AHB BAR.	0 - 16#FFF#	16#F00#
ioaddr	ADDR field of the AHB BAR (for register and diagnostic access).	0 - 16#FFF#	16#F00#
AHB master interface			
hmstndx	AHB master index.	0 - NAHBMST-1	0
bbuswidth	Maximum bus width on master AHB interface	32, 64, 128	128
bioaddr	ADDR field of the AHB BAR (for backend ioarea). Appears in the bridge's slave interface user-defined register 1.	0 - 16#FFF#	0
biomask	MASK field of the AHB BAR.	0 - 16#FFF#	0
wbmask	Wide-bus mask. Each bit in this value represent a 256Mbyte address range. To enabled wide accesses (>32-bit) to an address range, set the corresponding bit to '1'.	0 - 16#FFFF#	16#FFFF#
AXI master interface			
axiid	AXI ID	0 - 15	0
Cache configuration			
memtech	The memory technology used for the internal FIFOs.	0 - NTECH	0
cached	Fixed cachability mask.	0 - 16#FFFF#	16#0000#
hirq	Interrupt line used by the core.	0 - NAHBIRQ-1	0

*Table 1919.*Configuration options

Generic name	Function	Allowed range	Default
cen	Reset value for cache enable. 1 = cache enabled.	0 - 1	0
hproten	Reset value for enabling hprot functionality (Only available in version 2 of the core)	0 - 1	0
wp	Reset value for write-policy: 0 = copy-back, 1 = write-through (Only available in version 2 of the core)	0 - 1	0
repl	Reset value for replacement policy: 0 = LRU, 1 = pseudo-random (Only available in version 2 of the core)	0 - 1	0
ways	Number of cache ways	1 - 4	1
waysize	Size of each cache way in kBytes	1 - 512	1
linesize	Cache line size in bytes	32, 64	32
sbus	The number of the AHB bus to which the slave interface is connected. The value appears in bits [1:0] of the user-defined register 0 in the slave interface configuration record and master configuration record.	0-3	0
mbus	The number of the AHB bus to which the master interface is connected. The value appears in bits [3:2] of the user-defined register 0 in the slave interface configuration record and master configuration record.	0-3	1
stat	Enables the statistics counters. 0 all counters is disabled. 1 enables the access/hit counter. 2 enables the bus usage counter in addition to the access/hit counter.	0-2	0
arch	Selects between separate (0) or shared (1) RAM in multi-way configurations (see text below)	0 - 1	0
mtrr	Number of MTRR registers	0 - 32	0
edacen	Default value for the EDACEN field in the cache control register	0 - 1	0
rmw	Enables Read-Modify-Write for sub-word writes.	0 - 1	0
ft	Enables the memory protection (EDAC) implementation 1: Enable generic EDAC protection 2. Enable technology specific EDAC protection. Technology specific protection is further documented in the GRLIB-FT User's Manual (grlib-ft.pdf).	0 - 2	0
fttiming	Simulate access timing as if memory protection was enabled. (Only for prototype testing)	0 - 1	0

97.9 Signal descriptions

Table 1920 shows the interface signals of the core (VHDL ports).

*Table 1920.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
AXIMI	*	Input	AXI master input signals	-
AXIMO	*	Output	AXI master output signals	-
AHBSOV	*	Input	Vector of all AHB slave outputs on the backend AHB bus.	
STO	bit[2]: Access bit[1]: Miss bit[0]: Hit	Output	Statistic output.	

*) see GRLIB IP Library User's Manual.

97.10 Library dependencies

Table 1921 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1921.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	L2CACHE	Component	Component declaration

97.11 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.stdlib.all;
use grlib.tech.all;
library gaisler;
use gaisler.l2cache.all;

entity l2c_ex is
  port (
    clk : in std_ulogic;
    rst : in std_ulogic
  );
end;
.
.
signal ahbsi  : ahb_slv_in_type;
signal ahbso  : ahb_slv_out_vector := (others => ahbs_none);
signal ahbmi  : ahb_mst_in_type;
signal ahbmo  : ahb_mst_out_vector := (others => ahbm_none);
```

```
signal ahbsi2 : ahb_slv_in_type;
signal ahbso2 : ahb_slv_out_vector := (others => ahbs_none);
signal ahbmi2 : ahb_mst_in_type;
signal ahbmo2 : ahb_mst_out_vector := (others => ahbm_none);
signal aximi : ahb_somi_type;
signal aximo : ahb_mosi_type;


architecture rtl of l2c_ex is

begin

(AHB backend instantiation)
...

  l2c0 : l2c
  generic map(hslvidx => 5, hmstidx => 1, cen => 0, haddr => 16#400#, hmask => 16#C00#,
              ioaddr => 16#FF4#, cached => 16#00F3#, repl => 0, ways => 1,
              linesize => 32, waysize => 512, memtech => 0, bbuswidth => 64)
  port map(rst => rst, clk => clk, ahbsi => ahbsi, ahbso => ahbso(5),
           ahbmi => ahbmi2, ahbmo => ahbmo2(1), ahbsov => ahbso2);

...

(AXI backend instantiation)
...

  l2c0 : l2c_axi_be
  generic map(hslvidx => 5, axiid=> 0, cen => 0, haddr => 16#400#, hmask => 16#C00#,
              ioaddr => 16#FF4#, cached => 16#00F3#, repl => 0, ways => 1,
              linesize => 32, waysize => 512, memtech => 0)
  port map(rst => rst, clk => clk, ahbsi => ahbsi, ahbso => ahbso(5),
           aximi => aximi, aximo => aximo);

...

end;
```

98 L2C-Lite - Level 2 Cache controller

98.1 Overview

L2C-L implements a level-2 cache for processors with an AHB interface. It uses the GRLIB core "Generic Bus Master" as a bridge between AHB-AHB or AHB-AXI. The cache supports both big and little endian, making it suitable for NOEL-V and LEON designs. The frontend and backend support bus widths between 32-128 bits.

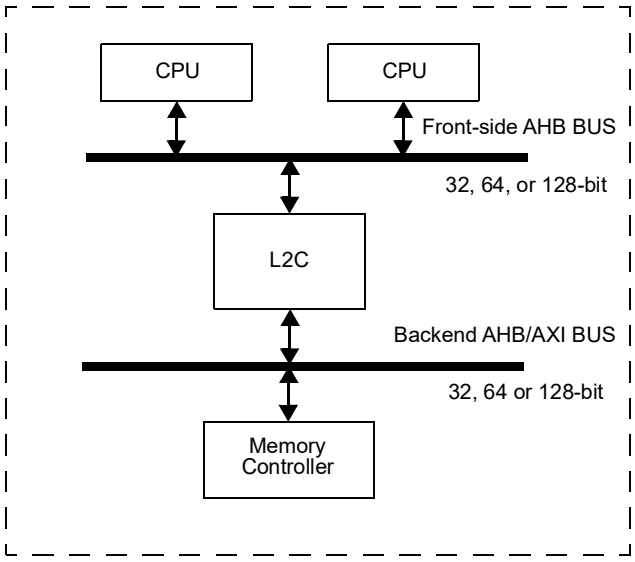


Figure 257. Block diagram

98.2 Configuration

L2C-L can be configured as a direct-mapped or multi-way associative cache. The cache supports 1, 2, 4, 8, 16, or 32 ways, with a cache line size of 32 or 64 Bytes. The way size is configurable between 1 KiB and 2 MiB, but can be limited by the techmap.

98.2.1 Replacement policy

L2C-L implements Pseudo LRU and Random replacement policies. Pseudo LRU mimics LRU with less memory overhead, this is especially beneficial when using many ways. The random replacement selects cache lines to evict based on a counter that is incremented each clock cycle.

98.2.2 Write policy

The cache uses a copy-back write policy. A write access fetches the associated cache line from memory (unless a cache hit), updates the cache line, and sets the valid and dirty bits. Evicted cache lines are written back to memory.

98.2.3 Cachability

The core uses the VHDL generic, “cached”, to determine which address range is cachable. Each bit in the 16-bit value defines the cachability of a 256 Mbyte address block on the AMBA AHB bus. A value of 16#00F3# will thus define cachable areas in 0 - 0x20000000 and 0x40000000 - 0x80000000

98.2.4 AHB address mapping

The AHB slave interface occupies two AHB address ranges. The first AHB memory bar is used for the cache data range and is set up using VHDL generics “haddr” and “hmask”. The second range is used for accessing I/O registers and its address is set up using VHDL generic “ioaddr”. The size of the I/O area is statically set to 1 kB.

98.3 Operation

98.3.1 Read

A read access starts with a tag lookup. If the tag is found and valid, the cache delivers the requested data. Burst reads within a cacheline have no wait states after the initial tag matching. If a burst crosses a cacheline, wait states are inserted and a new tag lookup is performed. If the tag is not found, the replacement policy selects the cache line to be replaced. Evicted cache lines are buffered and written back to main memory once the new line is fetched. Wait states are inserted during the fetch. Non-cacheable read accesses issue single read accesses to the backend, affecting performance.

98.3.2 Write

A write access begins with a tag lookup. If the tag is found, writes are buffered and combined, then written back to memory. For write burst accesses, the tag lookup is repeated for each cacheline. Writing to a cacheline sets the dirty bit, triggering eviction during replacement. Write misses allocate new cache lines.

98.3.3 AHB slave interface

AHB Slave Interface: The slave interface connects the core to the CPU and level 1 cache. It supports 8-bit (byte), 16-bit (half word), 32-bit (word), 64-bit, and 128-bit single and burst accesses. Fixed increment and WRAP accesses are not supported.

98.3.4 AXI/AHB Master interface

The master interface connects the core to the memory controller using the Generic Bus Master IP core. It supports AXI and AHB and can be configured as 32, 64, or 128 bits.

98.3.5 Endianness

The core is compatible with big and little-endian systems.

98.3.6 Performance counters

Two 31-bit performance counters are available for access and misses. The overflow indicator is set in the last bit, which can only be cleared through the register interface. The counters can be read and reset. The hit counter can be calculated by subtracting the miss counter from the access counter.

98.3.7 Flush

L2C-L supports 5 flush modes: Flushing, Flush+Invalidate, Invalidate, Flush Address, and Flush Address + Invalidate.

98.3.8 Diagnostics

Diagnostic access to the tag and data RAM is available through the register interface. Internal RAMs can be read and written to.

98.4 Registers

The core is configured via registers mapped into the AHB memory address space. Only 32-bit single-accesses to the registers are supported.

*Table 1922.*L2C-Lite: AHB registers

AHB address offset	Register
0x00	Configuration Register
0x04	Control Register
0x08	Flush Control Register
0x0C	Access Counter
0x10	Miss Counter
0x14	Diagnostics Control Register
0x40 - 0x7F	Diagnostics Data Register
0x80	Diagnostics Tag Register

98.4.1 Configuration Register

*Table 1923.*0x00 - L2C-Lite - Configuration

31 30	29 28	27 20	19 16
REPL	RESERVED	WAYS	LINE-SIZE
*	*	*	*
r	r	r	r

15 14	13 0
RESERVED	WAY-SIZE
*	*
r	r

31: 30	Replacement policy (REPL), 0 = Pseudo random, 1 = Pseudo LRU
29:28	RESERVED
27:20	Multi-way configuration (WAYS) - (Associativity - 1)
19:16	LINE-SIZE - 1 : 32 B 2 : 64 B
15:14	RESERVED
13:0	WAY-SIZE - Size in KiB

98.4.2 Control Register

*Table 1924.*0x04 - L2C-Lite - Control

31 1	0	0
Reserved	REPL	EN
0	*	*
r	r/w	r/w

31 : 2	Reserved.
1	Replacement Policy, 0 Random, 1 for pLRU
0	Enable, Cache enable/disable

98.4.3 Flush Control Register

*Table 1925.*0x08 - L2C-Lite - Flush Control

31 5	4	3:1	0
Address	Res	Mode	EN
0	0	*	*
r/w	r	r/w	r/w

31 : 5	Address
4	Reserved
3:1	Mode: Flush = “000” Flush-Invalidate = “001” Invalidate = “010” Flush Address= “011” Flush-Invalidate Address = “100”
0	Enable, Starts flush

98.4.4 Access Counter

*Table 1926.*0x0C - L2C-Lite - Access counter register

31	30 … 0
OF	Access counter
0	0
r/w	r/w

31 Overflow, Need to be reset manually

30 : 0 Access counter. Write 0 to clear counter.

98.4.5 Miss Counter

*Table 1927.*0x10 - L2C-Lite - Miss counter register

31	30 … 0
OF	Miss counter
0	0
r/w	r/w

31 Overflow, Need to be reset manually

30 : 0 Miss counter. Write 0 to clear counter.

98.4.6 Diagnostics Control Register

*Table 1928.*0x14 - L2C-Lite - Diagnostics Control

31 … 16	15 … 8	7 : 3	2	1	0
Index	way	Reserved	write	pend	v
*	*	0	*	*	*
r/w	r/w	r	r/w	r/w	r

31 : 16 Index, RAM address

15 : 8 Way, Selected way

7 : 3 Reserved

2 Write, R/W to L2C RAMs

1 Pending,

Starts diagnostics access. Clears when access is finished
If write = ‘1’, Data & Tag registers are written to Index & Way selected in this register

If write = ‘0’, Reads out cacheline from Index & Way

0 v,

Valid bit is set when cacheline read is finished. Cleared when selected cacheline is modified or when a write is made to the Diagnostics Control Register

98.4.7 Diagnostics Tag Register

*Table 1929.*0x80 - L2C-Lite - Diagnostics Tag

31 … 2	1	0
Tag	Valid	Dirty
*	*	*
r/w	r/w	r/w

31 : 2 Tag, size depends on cacheline size & way size

1 Valid

0 Dirty

98.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Cobham Gaisler) and device identifier 0x0D0. For description of vendor and device identifier see GRLIB IP Library User's Manual.

98.6 Implementation

98.6.1 RAM usage

The L2C-Lite uses dual-port RAM to implement both cache tags and data memory. The tags are implemented using the SYNCRAM_2P core, with the width and depth depending on the cache size configuration. The data memory is implemented using the SYNCRAM_2P which means that it can limit the line size and way size depending on which technology is used. See SYNCRAM_2P for technology specific limitations.

98.7 Configuration options

Table 1930 shows the configuration options of the core (VHDL generics).

*Table 1930.*Configuration options

Generic name	Function	Allowed range	Default
tech	The memory technology used for the internal Syncram.	0 - NTECH	0
hmindex	Master index	0 - NAHBMST-1	0
hsindex	Slave index.	0 - NAHBSLV-1	0
ways	Number of cache ways	1, 2, 4, 8, 16, 32	2
waysize	Size of ways (KiB)	1,2,4,8,16,32,64, 128 ...	64
linesize	MASK field of the AHB BAR.	32/64	32
repl	Replacement policy: 0 = pseudo-random, 1 = pseudo-LRU/pseudo-random	0-1	0
haddr	ADDR field of the AHB BAR	0 - 16#FFF#	0
hmask	MASK field of the AHB BAR.	0 - 16#FFF#	0
ioaddr	ADDR field of the AHB I/O BAR	0 - 16#FFF#	16#000#
cached	Cacheable memory ranges	0 - 16#FFFF#	16#FFFF#
be_dw	Backend bus width	32, 64, 128	32

98.8 Signal descriptions

Table 1931 shows the interface signals of the core (VHDL ports).

*Table 1931.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
AXIMI	*	Input	AXI master input signals	-
AXIMO	*	Output	AXI master output signals	-

*) see GRLIB IP Library User's Manual.

98.9 Library dependencies

Table 1932 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1932.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	L2C-L	Component	Component declaration
GRLIB	generic_bm	Component	Component declaration

98.10 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.stdlib.all;
use grlib.tech.all;
library gaisler;
use gaisler.l2c_lite.all;

entity l2c_lite_ex is
  port (
    clk : in std_ulogic;
    rst : in std_ulogic
  );
end;
.
.
signal ahbsi  : ahb_slv_in_type;
signal ahbso  : ahb_slv_out_vector := (others => ahbs_none);
signal ahbmi  : ahb_mst_in_type;
signal ahbmo  : ahb_mst_out_vector := (others => ahbm_none);
signal ahbsi2 : ahb_slv_in_type;
signal ahbso2 : ahb_slv_out_vector := (others => ahbs_none);
signal ahbmi2 : ahb_mst_in_type;
signal ahbmo2 : ahb_mst_out_vector := (others => ahbm_none);
```

```
signal aximi : axi_somi_type;
signal aximo : axi_mosi_type;


architecture rtl of l2c_lite_ex is

begin

(AHB backend instantiation)
...

  l2c_lite0 : l2c_lite_ahb
  generic map(tech => 0, hmindex => 1, hsindex => 1, ways => 1, waysize => 64,
              linesize => 32, repl => 0, haddr => 16#400#, hmask => 16#C00#,
              ioaddr => 16#FF4#, cached => 16#00F3#, be_dw => 32)
  port map(rst => rst, clk => clk, ahbsi => ahbsi, ahbso => ahbso(1),
           ahbmi => ahbmi2, ahbmo => ahbmo2(1), ahbsov => ahbso2);

...

(AXI backend instantiation)
...

 l2c_lite0 : l2c_lite_axi3
 generic map(tech => 0, hmindex => 1, hsindex => 1, ways => 1, waysize => 64,
              linesize => 32, repl => 0, haddr => 16#400#, hmask => 16#C00#,
              ioaddr => 16#FF4#, cached => 16#00F3#, be_dw => 32)
  port map(rst => rst, clk => clk, ahbsi => ahbsi, ahbso => ahbso(1),
           aximi => aximi, aximo => aximo);

...

end;
```

99 L3STAT - LEON3 Statistics Unit

99.1 Overview

The LEON3 Statistics Unit (L3STAT) is used to count events in the LEON3 processor and on the AHB bus, in order to create performance statistics for various software applications.

L3STAT consists of a configurable number of 32-bit counters, which increment on a certain event. The counters roll over to zero when reaching their maximum value, but can also be automatically cleared on reading to facilitate statistics building over longer periods. Each counter has a control register where the event type is selected. In multi-processor systems, the control registers also indicates which particular processor core is monitored. The table 1933 below shows the event types that can be monitored.

NOTE: L3STAT does currently not support double-clocked processor configurations. The processors and statistics unit must be run on the same frequency as the AMBA buses for L3STAT to function correctly.

*Table 1933.*Event types and IDs

ID	Event description
Processor events:	
0x00	Instruction cache miss
0x01	Instruction MMU TLB miss
0x02	Instruction cache hold
0x03	Instruction MMU hold
0x08	Data cache (read) miss
0x09	Data MMU TLB miss
0x0A	Data cache hold
0x0B	Data MMU hold
0x10	Data write buffer hold
0x11	Total instruction count
0x12	Integer instructions
0x13	Floating-point unit instruction count
0x14	Branch prediction miss
0x15	Execution time, excluding debug mode
0x17	AHB utilization (per AHB master) (implementation depedent)
0x18	AHB utilization (total, master/CPU selection is ignored) (implementation dependent)
0x22	Integer branches
0x28	CALL instructions
0x30	Regular type 2 instructions
0x38	LOAD and STORE instructions
0x39	LOAD instructions
0x3A	STORE instructions
AHB events (only available if core is connected to a LEON3 Debug Support Unit):	
0x40	AHB IDLE cycles.
0x41	AHB BUSY cycles.
0x42	AHB NON-SEQUENTIAL transfers. Filtered on CPU/AHBM if SU(1) = ‘1’
0x43	AHB SEQUENTIAL transfers. Filtered on CPU/AHBM if SU(1) = ‘1’
0x44	AHB read accesses. Filtered on CPU/AHBM if SU(1) = ‘1’
0x45	AHB write accesses. Filtered on CPU/AHBM if SU(1) = ‘1’

*Table 1933.*Event types and IDs

ID	Event description
0x46	AHB byte accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x47	AHB half-word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x48	AHB word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x49	AHB double word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x4A	AHB quad word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x4B	AHB eight word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x4C	AHB waitstates. Filtered on CPU/AHBM if SU(1) = '1'
0x4D	AHB RETRY responses. Filtered on CPU/AHBM if SU(1) = '1'
0x4E	AHB SPLIT responses. Filtered on CPU/AHBM if SU(1) = '1'
0x4F	AHB SPLIT delay. Filtered on CPU/AHBM if SU(1) = '1'
0x50	AHB bus locked. Filtered on CPU/AHBM if SU(1) = '1'
0x51-0x5F	Reserved
Implementation specific events:	
0x60 - 0x6F	External event 0 - 15. Filtered on CPU/AHBM if SU(1) = '1'.
AHB events (only available if core is connected to a standalone AHB trace buffer):	
0x70	AHB IDLE cycles.
0x71	AHB BUSY cycles. Filtered on CPU/AHBM if SU(1) = '1'
0x72	AHB NON-SEQUENTIAL transfers. Filtered on CPU/AHBM if SU(1) = '1'
0x73	AHB SEQUENTIAL transfers. Filtered on CPU/AHBM if SU(1) = '1'
0x74	AHB read accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x75	AHB write accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x76	AHB byte accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x77	AHB half-word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x78	AHB word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x79	AHB double word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x7A	AHB quad word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x7B	AHB eight word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x7C	AHB waitstates. Filtered on CPU/AHBM if SU(1) = '1'
0x7D	AHB RETRY responses. Filtered on CPU/AHBM if SU(1) = '1'
0x7E	AHB SPLIT responses. Filtered on CPU/AHBM if SU(1) = '1'
0x7F	AHB SPLIT delay. Filtered on CPU/AHBM if SU(1) = '1'
Events generated from REQ/GNT signals (only available if core has been implemented with VHDL generic reqsel /=0)	
0x80 - 0x8F	Active when master selected by CPU/AHBM field has request asserted while grant is asserted for the master correspoding to the least significant nibble of the event ID. 0x80 is master 0 grant, 0x81 is master 1 grant, .., and so on.
0x90 - 0x9F	Active when master selected by CPU/AHBM field has request asserted while grant is deasserted for the master correspoding to the least significant nibble of the event ID. 0x90 is master 0 grant, 0x91 is master 1 grant, .., and so on.

Note that IDs 0x39 (LOAD instructions) and 0x3A (STORE instructions) will both count all LDST and SWAP instructions. The sum of events counted for 0x39 and 0x3A may therefore be larger than the number of events counted with ID 0x38 (LOAD and STORE instructions).

Event 0x00 - 0x3A can be counted of the core has been connected to one or several LEON3 processor cores. Counting of events 0x40 - 0x5F requires that the core is connected to a LEON3 Debug Support Unit (DSU). The core's Counter control registers have a field that shows if the core has been imple-

mented with this connection. The documentation for the Debug Support Unit contains more information on events 0x40 - 0x5F. Please note that the statistical outputs from the DSU may be subject to AHB trace buffer filters. The same applies to events 0x70 - 0x7F that can come from an AHBTRACE core.

The core can also be implemented with support for counting up to 15 external events. These events can come from any source, but should be clocked by a clock which is synchronous with the AMBA clock used for the L3STAT core.

99.2 Multiple APB interfaces

The core can be implemented with two AMBA APB interfaces. The first APB interface always has precedence when both interfaces handle write operations to the same address.

99.3 Registers

The L3STAT core is programmed through registers mapped into APB address space.

Table 1934. L3STAT counter control register

APB address offset	Register
0x00	Counter 0 value register
0x04	Counter 1 value register
4 * *n*	Counter *n* value register
0x100	Counter 0 control register
0x104	Counter 1 control register
0x100 + (4 * *n*)	Counter *n* control register
0x200	Counter 0 max/latch register
0x204	Counter 1 max/latch register
0x200 + (4 * *n*)	Counter *n* max/latch register
0x300	Timestamp register

Note: Revision 0 of this IP core had control registers starting at 0x80, max/latch registers starting at 0x100 and the timestamp register at 0x180. This IP core documentation is valid for revision 1 of the IP core.

99.3.1 Counter Value Register

Table 1935. 0x000+n.4 - CVALn - Counter value register

31	0
CVAL	
NR	
rw	

31: 0 Counter value (CVAL) - This register holds the current value of the counter. If the core has been implemented with support for keeping the maximum count (MC field of Counter control register is '1') and the Counter control register field CD is '1', then the value displayed by this register will be the maximum counter value reached with the settings in the counter's control register. Writing to this register will write both to the counter and, if implemented, the hold register for the maximum counter value.

99.3.2 Counter Control Register

Table 1936. 0x100+n.4 - CCTRLn - Counter control register

31 23	22	21	20	19	18	17	16	15 14	13	12	11 4	3 0
NCNT	MC	IA	DS	EE	AE	EL	CD	SU	CL	EN	EVENT ID	CPU/AHBM
*	*	*	*	*	*	NR	NR	NR	NR	0	NR	NR
r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw

31: 23 Number of counters (NCNT) - Number of implemented counters - 1

Note revision 0 of this core used bits 31:28 to indicate the number of CPUs. This manual applies to revision 1 of the core.

22 Maximum count (MC) - If this field is '1' then this counter has support for keeping the maximum count value

21 Internal AHB count (IA) - If this field is '1' the core supports events 0x17 and 0x18

20 DSU support (DS) - If this field is '1' the core supports events 0x40-0x5F

19 External events (EE) - If this field is '1' the core supports external events (events 0x60 - 0x6F)

18 AHBTRACE Events (AE) - If this field is '1' the core supports events 0x70 - 0x7F.

17 Event Level (EL) - The value of this field determines the level where the counter keeps running when the CD field below has been set to '1'. If this field is '0' the counter will count the time between event assertions. If this field is '1' the counter will count the cycles where the event is asserted. This field can only be set if the MC field of this register is '1'.

16 Count maximum duration (CD) - If this bit is set to '1' the core will save the maximum time the selected event has been at the level specified by the EL field. This also means that the counter will be reset when the event is activated or deactivated depending on the value of the EL field.

When this bit is set to '1', the value shown in the counter value register will be the maximum current value which may be different from the current value of the counter.

This field can only be set if the MC field of this register is '1'.

15: 14 Supervisor/User mode filter (SU) - "01" - Only count supervisor mode events, "10" - Only count user mode events, others values - Count events regardless of user or supervisor mode. This setting only applies to events 0x0 - 0x3A.

When SU = "1x" only events generated by the CPU/AHB master specified in the CPU/AHBM field will be counted. This applies to events 0x40 - 0x7F.

: 13 Clear counter on read (CL) - If this bit is set the counter will be cleared when the counter's value is read. The register holding the maximum value will also be cleared, if implemented.

If an event occurs in the same cycle as the counter is cleared by a read then the event will not be counted. The counter latch register can be used to guarantee that no events are lost

12 Enable counter (EN) - Enable counter

11: 4 Event ID to be counted

3: 0 CPU or AHB master to monitor.(CPU/AHBM) - The value of this field does not matter when selecting one of the events coming from the Debug Support Unit or one of the external events.

99.3.3 Counter max/latch Register

*Table 1937.*0x200+4n - CSVALn - Counter max/latch register

31	0
CSVAL	
NR	
rw*	

31: 0 Counter max/latch value (CSVAL) - This register holds the current value of the counter max/latch register. It is only available for a specific counter If the core has been implemented with support for keeping the maximum count (MC field of Counter control register is ‘1’).

If the counter control register field CD is ‘1’, then the value displayed by this register will be the maximum counter value reached with the settings in the counter’s control register.

If the counter control register field CD is ‘0’, then the value displayed by this register is the latched (saved) counter value. The counter value is saved whenever a write access is made to the core in address range 0x100 - 0x1FC (all counters are saved simultaneously). If the counter control register CL field is set, then the current counter value will be cleared when the counter value is saved into this register.

99.3.4 Timestamp Register

*Table 1938.*0x300 - TSTAMP - Timestamp register

31	0
TSTAMP	
NR	
rw*	

31: 0 Timestamp (TSTAMP) - Timestamp taken at latch of counters

99.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x098. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

99.5 Implementation

99.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User’s Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

99.6 Configuration options

Table 1939 shows the configuration options of the core (VHDL generics).

*Table 1939.*Configuration options

Generic	Function	Allowed range	Default
pindex	Selects which APB select signal (PSEL) will be used to access the statistical unit	0 to NAPBMAX-1	0
paddr	The 12-bit MSB APB address	0 to 16#FFF#	0
pmask	The APB address mask. Needs to be set to 16#FFE# if max/latch registers are enabled (VHDL generic clatch).	0 to 16#FFF#	16#FFF#
ncnt	Defines the number of counters	1 to 32	4
ncpu	Defines the number of CPUs monitored	1 - 16	1
nmax	If this generic is > 0, the core will include functionality for tracking the longest consecutive time that an event is active or inactive. The functionality will be available for the *nmax* first counters.	0 - 32	0
lahben	If this generic is 1, the core makes use of the AHBSI input for events 0x17 and 0x18, otherwise the AHBSI input is unused and events 0x17 and 0x18 will never increment a counter.	0 - 1	0
dsuen	If this generic is 1, the core makes use of the DSUO input for events 0x40 - 0x5F, otherwise the DSUO input is unused and events 0x40 - 0x5F will never increment a counter.	0 - 1	0
nextev	Defines the number of external events monitored	0 - 16	0
apb2en	Enables the second APB port on the core.	0 - 1	0
pindex2	Selects which APB select signal (PSEL) will be used to access the second interface of the statistical unit	0 to NAPBMAX-1	0
paddr2	The 12-bit MSB APB address for second interface	0 to 16#FFF#	0
pmask2	The APB address mask for second interface	0 to 16#FFF#	16#FFF#
astaten	If this generic is 1, the core makes use of the ASTAT input for events 0x70 - 0x7F, otherwise the ASTAT input is unused and events 0x70 - 0x7F will never increment a counter.	0 - 1	0
selreq	Support STATI.REQ/SEL events. If selreq is nonzero then the value+1 defines the number of monitored sel/req signals.	0 - 15	0
clatch	Include support for counter max/latch registers and time-stamp.	0 - 1	0
forcer0	For core to revision 0. If this generic is set to 1 then the maximum number of supported counters is 32 and the register address offsets and layout of the control register changes to comply to revision 0 of the IP core.	0 - 1	0

99.7 Signal descriptions

Table 1940 shows the interface signals of the core (VHDL ports).

*Table 1940.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBSI	*	Input	AHB slave input signals	-
DBGO		Input	LEON3 debug output signals	-
DSUO	ASTAT	Input	DSU3 output signals	-
STATI	EVENT[15:0]	Input	Input for 16 user defined events	High
	ESOURCE [15:0][3:0]	Input	CPU/AHBM for each EVENT	-
	REQ[15:0]	Input	Input for 16 user defined request signals	High
	SEL[15:0]	Input	Input for 16 user defined select/grant signals	-
	TIMER[31:0]	Input	Time stamp value used when VHDL generic clatch /= 0	-
APB2I	*	Input	Secondary APB slave input signals	-
APB2O	*	Output	Secondary APB slave output signals	-
ASTAT	*	Input	AHBTRACE output signals	-

* see GRLIB IP Library User's Manual

99.8 Library dependencies

Table 1941 shows libraries used when instantiating the core (VHDL libraries).

*Table 1941.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	LEON3	Signals, component	Signal definitions and component declaration

99.9 Component declaration

The core has the following component declaration.

```
library gaisler;
use gaisler.leon3.all;

entity l3stat is
  generic (
    pindex      : integer := 0;
    paddr       : integer := 0;
    pmask       : integer := 16#fff#;
    ncnt        : integer := 4;
    ncpu        : integer := 1
    );
  port (
    rstn   : in std_ulogic;
    clk    : in std_ulogic;
    apbi   : in apb_slv_in_type;
```

```
    apbo   : out apb_slv_out_type;
    ahbsi  : in  ahb_slv_in_type;
    dbgo   : in l3_debug_out_vector(0 to NCPU-1));
end;
```

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.leon3.all;

begin

l3sgen : if CFG_L3S_ENABLE = 1 generate
    l3stat0 : l3stat
      generic map (pindex => 11, paddr => 11, ncnt => CFG_L3S_CNT, ncpu => CFG_NCPU)
      port map (rstn, clkm, apbi, apbo(11), ahbsi, dbgo);
end generate;
```

100 L4STAT - LEON4 Statistics Unit

100.1 Overview

The LEON4 Statistics Unit (L4STAT) is used count events in the LEON4 processor and on the AHB bus, in order to create performance statistics for various software applications.

L4STAT consists of a configurable number of 32-bit counters, which increment on a certain event. The counters roll over to zero when reaching their maximum value, but can also be automatically cleared on reading to facilitate statistics building over longer periods. Each counter has a control register where the event type is selected. In multi-processor systems, the control registers also indicates which particular processor core is monitored. The table 1942 below shows the event types that can be monitored.

NOTE: L4STAT does currently not support double-clocked processor configurations. The processors and statistics unit must be run on the same frequency as the AMBA buses for L4STAT to function correctly.

*Table 1942.*Event types and IDs

ID	Event description
Processor events:	
0x00	Instruction cache miss
0x01	Instruction MMU TLB miss
0x02	Instruction cache hold
0x03	Instruction MMU hold
0x08	Data cache (read) miss
0x09	Data MMU TLB miss
0x0A	Data cache hold
0x0B	Data MMU hold
0x10	Data write buffer hold
0x11	Total instruction count
0x12	Integer instructions
0x13	Floating-point unit instruction count
0x14	Branch prediction miss
0x15	Execution time, excluding debug mode
0x17	AHB utilization (per AHB master) (implementation depedent)
0x18	AHB utilization (total, master/CPU selection is ignored) (implementation dependent)
0x22	Integer branches
0x28	CALL instructions
0x30	Regular type 2 instructions
0x38	LOAD and STORE instructions
0x39	LOAD instructions
0x3A	STORE instructions
AHB events (only available if core is connected to a LEON4 Debug Support Unit):	
0x40	AHB IDLE cycles. Filtered on CPU/AHBM if SU(1) = ‘1’
0x41	AHB BUSY cycles. Filtered on CPU/AHBM if SU(1) = ‘1’
0x42	AHB NON-SEQUENTIAL transfers. Filtered on CPU/AHBM if SU(1) = ‘1’
0x43	AHB SEQUENTIAL transfers. Filtered on CPU/AHBM if SU(1) = ‘1’
0x44	AHB read accesses. Filtered on CPU/AHBM if SU(1) = ‘1’
0x45	AHB write accesses. Filtered on CPU/AHBM if SU(1) = ‘1’

*Table 1942.*Event types and IDs

ID	Event description
0x46	AHB byte accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x47	AHB half-word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x48	AHB word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x49	AHB double word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x4A	AHB quad word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x4B	AHB eight word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x4C	AHB waitstates. Filtered on CPU/AHBM if SU(1) = '1'
0x4D	AHB RETRY responses. Filtered on CPU/AHBM if SU(1) = '1'
0x4E	AHB SPLIT responses. Filtered on CPU/AHBM if SU(1) = '1'
0x4F	AHB SPLIT delay. Filtered on CPU/AHBM if SU(1) = '1'
0x50	AHB bus locked. Filtered on CPU/AHBM if SU(1) = '1'
0x51-0x5F	Reserved
Implementation specific events:	
0x60 - 0x6F	External event 0 - 15. Filtered on CPU/AHBM if SU(1) = '1'.
AHB events (only available if core is connected to a standalone AHB trace buffer):	
0x70	AHB IDLE cycles. Filtered on CPU/AHBM if SU(1) = '1'
0x71	AHB BUSY cycles. Filtered on CPU/AHBM if SU(1) = '1'
0x72	AHB NON-SEQUENTIAL transfers. Filtered on CPU/AHBM if SU(1) = '1'
0x73	AHB SEQUENTIAL transfers. Filtered on CPU/AHBM if SU(1) = '1'
0x74	AHB read accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x75	AHB write accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x76	AHB byte accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x77	AHB half-word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x78	AHB word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x79	AHB double word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x7A	AHB quad word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x7B	AHB eight word accesses. Filtered on CPU/AHBM if SU(1) = '1'
0x7C	AHB waitstates. Filtered on CPU/AHBM if SU(1) = '1'
0x7D	AHB RETRY responses. Filtered on CPU/AHBM if SU(1) = '1'
0x7E	AHB SPLIT responses. Filtered on CPU/AHBM if SU(1) = '1'
0x7F	AHB SPLIT delay. Filtered on CPU/AHBM if SU(1) = '1'
Events generated from REQ/GNT signals (only available if core has been implemented with VHDL generic reqsel /=0)	
0x80 - 0x8F	Active when master selected by CPU/AHBM field has request asserted while grant is asserted for the master correspoding to the least significant nibble of the event ID. 0x80 is master 0 grant, 0x81 is master 1 grant, .., and so on.
0x90 - 0x9F	Active when master selected by CPU/AHBM field has request asserted while grant is deasserted for the master correspoding to the least significant nibble of the event ID. 0x90 is master 0 grant, 0x91 is master 1 grant, .., and so on.

Note that IDs 0x39 (LOAD instructions) and 0x3A (STORE instructions) will both count all LDST and SWAP instructions. The sum of events counted for 0x39 and 0x3A may therefore be larger than the number of events counted with ID 0x38 (LOAD and STORE instructions).

Event 0x00 - 0x3A can be counted of the core has been connected to one or several LEON4 processor cores. Counting of events 0x40 - 0x5F requires that the core is connected to a LEON4 Debug Support Unit (DSU). The core's Counter control registers have a field that shows if the core has been imple-

mented with this connection. The documentation for the Debug Support Unit contains more information on events 0x40 - 0x5F. Please note that the statistical outputs from the DSU may be subject to AHB trace buffer filters. The same applies to events 0x70 - 0x7F that can come from an AHBTRACE core.

The core can also be implemented with support for counting up to 15 external events. These events can come from any source, but should be clocked by a clock which is synchronous with the AMBA clock used for the L4STAT core.

100.2 Multiple APB interfaces

The core can be implemented with two AMBA APB interfaces. The first APB interface always has precedence when both interfaces handle write operations to the same address.

100.3 Registers

The L4STAT core is programmed through registers mapped into APB address space.

Table 1943. L4STAT counter control register

APB address offset	Register
0x00	Counter 0 value register
0x04	Counter 1 value register
4 * *n*	Counter *n* value register
0x100	Counter 0 control register
0x104	Counter 1 control register
0x100 + (4 * *n*)	Counter *n* control register
0x200	Counter 0 max/latch register
0x204	Counter 1 max/latch register
0x200 + (4 * *n*)	Counter *n* max/latch register
0x300	Timestamp register

Note: Revision 0 of this IP core had control registers starting at 0x80, max/latch registers starting at 0x100 and the timestamp register at 0x180. This IP core documentation is valid for revision 1 of the IP core.

100.3.1 Counter Value Register

*Table 1944.*0x000+n.4 - CVALn - Counter value register

31	0
CVAL	
NR	
rw	

31: 0 Counter value (CVAL) - This register holds the current value of the counter. If the core has been implemented with support for keeping the maximum count (MC field of Counter control register is '1') and the Counter control register field CD is '1', then the value displayed by this register will be the maximum counter value reached with the settings in the counter's control register. Writing to this register will write both to the counter and, if implemented, the hold register for the maximum counter value.

100.3.2 Counter Control Register

*Table 1945.*0x100+n.4 - CCTRLn - Counter control register

31 23	22	21	20	19	18	17	16	15 14	13	12	11 4	3 0
NCNT	MC	IA	DS	EE	AE	EL	CD	SU	CL	EN	EVENT ID	CPU/AHBM
*	*	*	*	*	*	NR	NR	NR	NR	0	NR	NR
r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw

31: 23 Number of counters (NCNT) - Number of implemented counters - 1

Note revision 0 of this core used bits 31:28 to indicate the number of CPUs. This manual applies to revision 1 of the core.

22 Maximum count (MC) - If this field is '1' then this counter has support for keeping the maximum count value

21 Internal AHB count (IA) - If this field is '1' the core supports events 0x17 and 0x18

20 DSU support (DS) - If this field is '1' the core supports events 0x40-0x5F

19 External events (EE) - If this field is '1' the core supports external events (events 0x60 - 0x6F)

18 AHBTRACE Events (AE) - If this field is '1' the core supports events 0x70 - 0x7F.

17 Event Level (EL) - The value of this field determines the level where the counter keeps running when the CD field below has been set to '1'. If this field is '0' the counter will count the time between event assertions. If this field is '1' the counter will count the cycles where the event is asserted. This field can only be set if the MC field of this register is '1'.

16 Count maximum duration (CD) - If this bit is set to '1' the core will save the maximum time the selected event has been at the level specified by the EL field. This also means that the counter will be reset when the event is activated or deactivated depending on the value of the EL field.

When this bit is set to '1', the value shown in the counter value register will be the maximum current value which may be different from the current value of the counter.

This field can only be set if the MC field of this register is '1'.

15: 14 Supervisor/User mode filter (SU) - "01" - Only count supervisor mode events, "10" - Only count user mode events, others values - Count events regardless of user or supervisor mode. This setting only applies to events 0x0 - 0x3A.

When SU = "1x" only events generated by the CPU/AHB master specified in the CPU/AHBM field will be counted. This applies to events 0x40 - 0x7F

: 13 Clear counter on read (CL) - If this bit is set the counter will be cleared when the counter's value is read. The register holding the maximum value will also be cleared, if implemented.

If an event occurs in the same cycle as the counter is cleared by a read then the event will not be counted. The counter latch register can be used to guarantee that no events are lost.

12 Enable counter (EN) - Enable counter

11: 4 Event ID to be counted

3: 0 CPU or AHB master to monitor.(CPU/AHBM) - The value of this field does not matter when selecting one of the events coming from the Debug Support Unit or one of the external events.

100.3.3 Counter Max/Latch Register

*Table 1946.*0x200+4n - CSVALn - Counter max/latch register

31	0
CSVAL	
NR	
rw*	

31: 0 Counter max/latch value (CSVAL) - This register holds the current value of the counter max/latch register. It is only available for a specific counter If the core has been implemented with support for keeping the maximum count (MC field of Counter control register is ‘1’).

If the counter control register field CD is ‘1’, then the value displayed by this register will be the maximum counter value reached with the settings in the counter’s control register.

If the counter control register field CDis ‘0’, then the value displayed by this register is the latched (saved) counter value. The counter value is saved whenever a write access is made to the core in address range 0x100 - 0x1FC (all counters are saved simultaneously). If the counter control register CL field is set, then the current counter value will be cleared when the counter value is saved into this register.

100.3.4 Timestamp Register

*Table 1947.*0x300 - TSTAMP - Timestamp register

31	0
TSTAMP	
NR	
rw*	

31: 0 Timestamp (TSTAMP) - Timestamp taken at latch of counters

100.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x047. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

100.5 Implementation

100.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User’s Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

100.6 Configuration options

Table 1948 shows the configuration options of the core (VHDL generics).

*Table 1948.*Configuration options

Generic	Function	Allowed range	Default
pindex	Selects which APB select signal (PSEL) will be used to access the statistical unit	0 to NAPBMAX-1	0
paddr	The 12-bit MSB APB address	0 to 16#FFF#	0
pmask	The APB address mask. Needs to be set to 16#FFE# if max/latch registers are enabled (VHDL generic clatch).	0 to 16#FFF#	16#FFF#
ncnt	Defines the number of counters	1 to 32	4
ncpu	Defines the number of CPUs monitored	1 - 16	1
nmax	If this generic is > 0, the core will include functionality for tracking the longest consecutive time that an event is active or inactive. The functionality will be available for the *nmax* first counters.	0 - 32	0
lahben	If this generic is 1, the core makes use of the AHBSI input for events 0x17 and 0x18, otherwise the AHBSI input is unused and events 0x17 and 0x18 will never increment a counter.	0 - 1	0
dsuen	If this generic is 1, the core makes use of the DSUO input for events 0x40 - 0x5F, otherwise the DSUO input is unused and events 0x40 - 0x5F will never increment a counter.	0 - 1	0
nextev	Defines the number of external events monitored	0 - 16	0
apb2en	Enables the second APB port on the core.	0 - 1	0
pindex2	Selects which APB select signal (PSEL) will be used to access the second interface of the statistical unit	0 to NAPBMAX-1	0
paddr2	The 12-bit MSB APB address for second interface	0 to 16#FFF#	0
pmask2	The APB address mask for second interface	0 to 16#FFF#	16#FFF#
astaten	If this generic is 1, the core makes use of the ASTAT input for events 0x70 - 0x7F, otherwise the ASTAT input is unused and events 0x70 - 0x7F will never increment a counter.	0 - 1	0
selreq	Support STATI.REQ/SEL events. If selreq is nonzero then the value+1 defines the number of monitored sel/req signals.	0 - 15	0
clatch	Include support for counter max/latch registers and time-stamp.	0 - 1	0
forcer0	For core to revision 0. If this generic is set to 1 then the maximum number of supported counters is 32 and the register address offsets and layout of the control register changes to comply to revision 0 of the IP core.	0 - 1	0

100.7 Signal descriptions

Table 1949 shows the interface signals of the core (VHDL ports).

*Table 1949.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBSI	*	Input	AHB slave input signals	-
DBGO		Input	LEON4 debug output signals	-
DSUO	ASTAT	Input	DSU4 output signals	-
STATI	EVENT[15:0]	Input	Input for 16 user defined events	High
	ESOURCE[3:0]	Input	CPU/AHBM for each EVENT	-
	REQ[15:0]	Input	Input for 16 user defined request signals	High
	SEL[15:0]	Input	Input for 16 user defined select/grant signals	-
	TIMER[31:0]	Input	Time stamp value used when VHDL generic clatch /= 0	-
APB2I	*	Input	Secondary APB slave input signals	-
APB2O	*	Output	Secondary APB slave output signals	-
ASTAT	*	Input	AHBTRACE output signals	-

* see GRLIB IP Library User's Manual

100.8 Library dependencies

Table 1950 shows libraries used when instantiating the core (VHDL libraries).

*Table 1950.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	LEON3	Signals	Signal definitions
GAISLER	LEON4	Signals, component	Component declaration

100.9 Component declaration

The core has the following component declaration.

```
library gaisler;
use gaisler.leon3.all;
use gausler.leon4.all;

entity l4stat is
  generic (
    pindex      : integer := 0;
    paddr       : integer := 0;
    pmask       : integer := 16#fff#;
    ncnt        : integer := 4;
    ncpu        : integer := 1
    );
  port (
    rstn   : in std_ulogic;
    clk    : in std_ulogic;
```

```
    apbi   : in apb_slv_in_type;
    apbo   : out apb_slv_out_type;
    ahbsi  : in  ahb_slv_in_type;
    dbgo   : in l4_debug_out_vector(0 to NCPU-1));
end;
```

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.leon4.all;

begin

l4sgen : if CFG_L4S_ENABLE = 1 generate
    l4stat0 : l4stat
      generic map (pindex => 11, paddr => 11, ncnt => CFG_L4S_CNT, ncpu => CFG_NCPU)
      port map (rstn, clkm, apbi, apbo(11), ahbsi, dbgo);
end generate;
```

101 LEON_DSU_STAT_BASE - LEON3/4 SUBSYSTEM

101.1 Overview

LEON_DSU_STAT_BASE is a subsystem component that can be used to instantiate LEON3 or LEON4 processors together with their respective debug support unit (DSU) and a statistics unit (performance counters).

LEON_DSU_STAT_BASE allows to select between LEON3 and LEON4, with FT options optionally enabled for both variants. The subsystem also supports sharing one FPU between several processors.

The subsystem makes use of the following interfaces:

- Clock and reset input
- One AHB master output per processor, shared AHB master input and AHB slave inputs for all processors
- One interrupt controller interface per processor
- One AMBA APB port for, optional, performance counters
- Performance counter inputs
- DSU AHB slave interface and trace buffer inputs
- DSU enable, break, active and timer stop signals

The subsystem does not provide any additional functionality to the LEON, DSU and statistics IP cores. The subsystem is provided as a convenient way to include the three separate IP cores and to decrease the amount of top-level code required in designs that can be configured to select between LEON3 and LEON4.

The subsystem is primarily targeted to FPGA designs and for designs without a clock gate unit. The design does not propagate scan test settings and memory BIST signals.

101.2 Operation

101.2.1 Operational model

For a subsystem with LEON3, please refer to the documentation for:

- LEON3, section 102
- DSU3, section 32
- L3STAT, section 99

For a subsystem with LEON4, please refer to the documentation for:

- LEON4, section 103
- DSU4, section 33
- L4STAT, section 100

101.2.2 Bus widths

LEON_DSU_STAT_BASE allows to select between LEON3 and LEON4. If LEON4 is selected then the GRLIB AMBA bus width needs to be set to 64 or 128 bits. This is accomplished by changing the GRLIB configuration (GRLIB_CONFIG) package. The configuration package is described in GRLIB IP Library User's Manual (grlib.pdf). See also section 101.7 for references to example usage of LEON_DSU_STAT_BASE where the designs select the bus width depending on the selection between LEON3 and LEON4.

101.3 Implementation

Please refer to the IP core documentation listed in section 101.2.1.

101.4 Configuration options

Table 1951 shows the configuration options of the core (VHDL generics).

*Table 1951.*Configuration options

Generic name	Function	Allowed range	Default
leon	Selects between LEON3 (3) and LEON4 (4)	0, 3, 4	0
ncpu	Number of processors. Propagated to generic with the same name on LEON and DSU instances.	1 - 16	1
fabtech	Propagated to generic with the same name on LEON instance(s) and (optional) shared GRFPU.	0 - NTECH	0
memtech	Propagated to generic with the same name on LEON instance(s). Propagated to *tech* generic on DSU. See also memtechmod generic below	0 - NTECH	0
memtechmod	memtech + memtechmod are assigned to the memtech VHDL generic on the LEON3 and LEON4 instances. memtech mod can be used to modify memtech to set the high bits forcing inferred memory for registers files and MMU memories. See documentation for memtech in LEON3 and LEON4 documentation.	-	0
nwindows	Propagated to generic with same name on LEON instance(s).	2 - 32	8
dsu	Propagated to generic with same name on LEON instance(s). Selects if DSU core is instantiated in subsystem.	0 - 1	0
fpu	Propagated to generic with same name on LEON instance(s). Corresponding generic on LEON is assigned with the value: *fpu + 32*grfpush*	0 - 63	0
v8	Propagated to generic with same name on LEON instance(s).	0 - 63	0
cp	Unused. cp generic on LEON instances is hardcoded to 0.	0 - 1	0
mac	Propagated to generic with same name on LEON instance(s).	0 - 1	0
pclow	Propagated to generic with same name on LEON instance(s).	0 - 2	2
notag	Propagated to generic with same name on LEON instance(s).	0 - 1	0
nwp	Propagated to generic with same name on LEON instance(s).	0 - 4	0
icen	Propagated to generic with same name on LEON instance(s).	0 - 1	0
irepl	Propagated to generic with same name on LEON instance(s).	0 - 2	2
isets	Propagated to generic with same name on LEON instance(s).	1 - 4	4
ilinesize	Propagated to generic with same name on LEON instance(s).	4 - 8	4
isetsize	Propagated to generic with same name on LEON instance(s).	1 - 256	1

*Table 1951.*Configuration options

Generic name	Function	Allowed range	Default
isetlock	Propagated to generic with same name on LEON instance(s).	0 - 1	0
dcen	Propagated to generic with same name on LEON instance(s).	0 - 1	0
drepl	Propagated to generic with same name on LEON instance(s).	0 - 2	2
dsets	Propagated to generic with same name on LEON instance(s).	1 - 4	1
dlinesize	Propagated to generic with same name on LEON instance(s).	4 - 8	4
dsetsize	Propagated to generic with same name on LEON instance(s).	1 - 256	1
dsetlock	Propagated to generic with same name on LEON instance(s).	0 - 1	0
dsnoop	Propagated to generic with same name on LEON instance(s).	0 - 6	0
ilram	Propagated to generic with same name on LEON instance(s).	0 - 1	0
ilramsize	Propagated to generic with same name on LEON instance(s).	1 - 512	1
ilramstart	Propagated to generic with same name on LEON instance(s).	0 - 255	16#8e#
dlram	Propagated to generic with same name on LEON instance(s).	0 - 1	0
dlramsize	Propagated to generic with same name on LEON instance(s).	1 - 512	1
dlramstart	Propagated to generic with same name on LEON instance(s).	0 - 255	16#8f#
mmuen	Propagated to generic with same name on LEON instance(s).	0 - 2	0
itlbnum	Propagated to generic with same name on LEON instance(s).	2 - 64	8
dtlbnum	Propagated to generic with same name on LEON instance(s).	2 - 64	8
tlb_type	Propagated to generic with same name on LEON instance(s).	0 - 3	1
tlb_rep	Propagated to generic with same name on LEON instance(s).	0 - 1	0
lddel	Propagated to generic with same name on LEON instance(s).	1 - 2	2
disas	Propagated to generic with same name on LEON instance(s).	0 - 2	0
tbuf	Propagated to generic with same name on LEON instance(s).	0 - 128	0
pwd	Propagated to generic with same name on LEON instance(s).	0 - 2	2
svt	Propagated to generic with same name on LEON instance(s).	0 - 1	1
rstaddr	Propagated to generic with same name on LEON instance(s).	-	0

*Table 1951.*Configuration options

Generic name	Function	Allowed range	Default
smp	Propagated to generic with same name on LEON instance(s).	0 - 31	0
cached	Propagated to generic with same name on LEON instance(s).	-	0
clk2x	Double clocking is currently unsupported with LEON_DSU_STAT_BASE. Leave at 0.	-	0
wbmask	Propagated to generic with same name on LEON4 instance(s). Not used for LEON3.	-	0
busw	Propagated to generic with same name on LEON4 instance(s). Not used for LEON3.	-	64
netlist	Propagated to generic with same name on LEON instance(s).	-	0
ft	Configured fault-tolerance. For LEON3 this value combines the settings for *iuft*, *fpft* and *cmft* in the following way: *iuft + (cmft) * 8 + fpft*2048*. For LEON4 the *ft* value is directly assigned to the generic by the same name.	-	0
npasi	Propagated to generic with same name on LEON instance(s).	0 - 1	0
pwrpsr	Propagated to generic with same name on LEON instance(s).	0 - 1	0
rex	Propagated to generic with same name on LEON instance(s).	0 - 1	0
altwin	Propagated to generic with same name on LEON3 instance(s). Not used for LEON4.	0 - 1	0
ahbpipe	Unused	-	0
mmupgsz	Propagated to generic with same name on LEON instance(s).	0 - 4	0
grfpush	Enabled shared FPU between instantiated LEONs.	0 - 1	0
dsu_hindex	Propagated to *hindex* generic on DSU instance.	-	2
dsu_haddr	Propagated to *haddr* generic on DSU instance.	-	16#900#
dsu_hmask	Propagated to *hmask* generic on DSU instance.	-	16#F00#
atbsz	Propagated to *kbytes* generic on DSU instance.	-	4
stat	Enables LEON statistics unit corresponding to *leon* selection of processor.	0 - 1	0
stat_pindex	Propagated to *pindex* generic on L*STAT instance.	-	0
stat_paddr	Propagated to *paddr* generic on L*STAT instance.	-	0
stat_pmask	Propagated to *pmask* generic on L*STAT instance.	-	16#FFC#
stat_ncnt	Propagated to *ncnt* generic on L*STAT instance.	-	1
stat_nmax	Propagated to *nmax* generic on L*STAT instance.	-	0

101.5 Signal descriptions

Table 1952 shows the interface signals of the core (VHDL ports).

*Table 1952.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
AHBCLK	N/A	Input	System clock	-
CPUCLK	N/A	Input	System clock	-
HCLKEN	N/A	Input	Double-clocking signal. Tie HIGH	HIGH
LEON_AHBMI	*	Input	LEON AHB master input signals	-
LEON_AHBMO[]	*	Output	LEON AHB master output signals, one per processor	-
LEON_AHBSI	*	Input	LEON AHB slave input signals (for snooping)	-
LEON_AHBSO	*	Input	LEON AHB slave output signals (for plug&play decoding)	-
IRQI	-	Input	Interrupt controller interface see LEON documentation	-
IRQO	-	Output		-
STAT_APBI	*	Input	L*STAT APB slave input signals	-
STAT_APBO	*	Output	L*STAT APB slave output signals	-
STAT_AHBSI	*	Input	L*STAT AHB slave input signals	-
STATI	-	Input	L*STAT input signal. See L3STAT/L4STAT documentation.	-
DSU_AHBSI	*	Input	DSU AHB slave input signals	-
DSU_AHBSO	*	Output	DSU AHB slave output signals	-
DSU_TAHBMI	*	Input	DSU AHB master input signals, used for AHB tracing	-
DSU_TAHBSI	*	Input	DSU AHB slave input signals, used for AHB tracing	-
SYSI	DSU_ENABLE	Input	Connected to DSU DSUI.ENABLE input	HIGH
	DSU_BREAK	Input	Connected to DSU DSUI.BREAK input	HIGH
SYSO	DSU_ACTIVE	Output	Connected to DSU DSUO.ACTIVE output	HIGH
	DSU_TSTOP	Output	Connected to DSU DSIO.TSTOP output	HIGH

* see GRLIB IP Library User's Manual

101.6 Library dependencies

Table 1953 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1953.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	SUBSYS	Component, signals	Component and signal definitions
GAISLER	LEON3	Signals	Interrupt controller signals
GRLIB	AMBA	Signals	AMBA signal definitions

101.7 Instantiation

Examples on how to use the subsystem can be seen in several GRLIB template designs, including:

- designs/leon3-gr-cpci-xc4v

- designs/leon3-gr-cpci-xc7k
- designs/leon3-gr-pci-xc5v
- designs/leon3-xilinx-kc705
- designs/leon3-xilinx-ml50x
- designs/leon3-xilinx-ml510
- designs/leon3-xilinx-vc707

Note that the Makefile in these designs also contains special conditions that depend on the selection between LEON3 and LEON4.

102 LEON3/FT - High-performance SPARC V8 32-bit Processor

102.1 Overview

LEON3 is a 32-bit processor core conforming to the IEEE-1754 (SPARC V8) architecture. It is designed for embedded applications, combining high performance with low complexity and low power consumption.

The LEON3 core has the following main features: 7-stage pipeline with Harvard architecture, separate instruction and data caches, memory management unit, hardware multiplier and divider, on-chip debug support and multi-processor extensions.

The LEON3 processor can be enhanced with fault-tolerance against SEU errors (referred to as LEON3FT). The fault-tolerance is focused on the protection of on-chip RAM blocks, which are used to implement IU/FPU register files and the cache memory. Configuring the processor to implement fault-tolerance enables additional internal registers, register fields and changes the processor's plug&play device ID. This documentation describes both the LEON3 and LEON3FT versions of the processor.

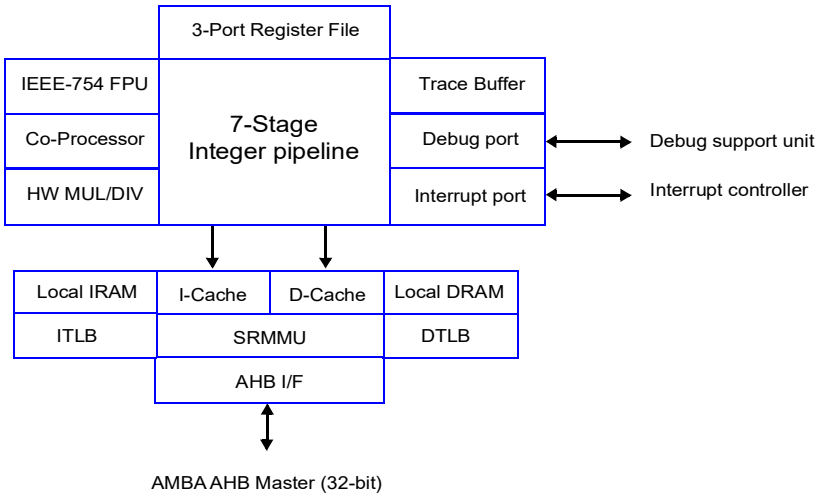


Figure 258. LEON3 processor core block diagram

Note: This manual describes the full functionality of the LEON3 core. Through the use of VHDL generics, parts of the described functionality can be suppressed or modified to generate a smaller or faster implementation. A comparison with earlier revisions can be found in section 102.13.

Please also refer to the *LEON/GRLIB Configuration and Development Guide* for recommendations on system design and LEON3 configuration.

102.1.1 Integer unit

The LEON3 integer unit implements the full SPARC V8 manual, including hardware multiply and divide instructions. The number of register windows is configurable within the limit of the SPARC manual (2 - 32), with a default setting of 8. The pipeline consists of 7 stages with a separate instruction and data cache interface (Harvard architecture).

102.1.2 Cache sub-system

LEON3 has a highly configurable cache system, consisting of a separate instruction and data cache. Both caches can be configured with 1 - 4 ways, 1 - 256 KiB/way, 16 or 32 bytes per line. The instruc-

tion cache maintains one valid bit per 32-bit word and uses streaming during line-refill to minimize refill latency. The data cache has one valid bit per cache line, uses write-through policy and implements a double-word write-buffer. Bus-snooping on the AHB bus can be used to maintain cache coherency for the data cache. Local scratch pad ram can be added to either of the instruction and data caches to allow 0-waitstates access instruction or data memory without any AHB bus access.

102.1.3 Floating-point unit and co-processor

The LEON3 integer unit provides interfaces for a floating-point unit (FPU), and a custom co-processor. Two FPU controllers are available, one for the high-performance GRFPU and one for the GRFPU-Lite core. The floating-point processors and co-processor execute in parallel with the integer unit, and does not block the operation unless a data or resource dependency exists. Note that the FPUs are provided separately.

102.1.4 Memory management unit

A SPARC V8 Reference Memory Management Unit (SRMMU) can optionally be enabled. The SRMMU implements the full SPARC V8 MMU specification, and provides mapping between multiple 32-bit virtual address spaces and physical memory. A three-level hardware table-walk is implemented, and the MMU can be configured to up to 64 fully associative TLB entries per implemented TLB.

102.1.5 On-chip debug support

The LEON3 pipeline includes functionality to allow non-intrusive debugging on target hardware. To aid software debugging, up to four watchpoint registers can be enabled. Each register can cause a breakpoint trap on an arbitrary instruction or data address range. When the (optional) debug support unit is attached, the watchpoints can be used to enter debug mode. Through a debug support interface, full access to all processor registers and caches is provided. The debug interfaces also allows single stepping, instruction tracing and hardware breakpoint/watchpoint control. An internal trace buffer can monitor and store executed instructions, which can later be read out via the debug interface.

102.1.6 Interrupt interface

LEON3 supports the SPARC V8 interrupt model with a total of 15 asynchronous interrupts. The interrupt interface provides functionality to both generate and acknowledge interrupts.

102.1.7 AMBA interface

The cache system implements an AMBA AHB master to load and store data to/from the caches. The interface is compliant with the AMBA-2.0 standard. During line refill, incremental bursts are generated to optimise the data transfer. The processor also has a snoop AHB slave input port which is used to monitor the accesses made by other masters, if snooping has been enabled.

102.1.8 Power-down mode

The LEON3 processor core implements a power-down mode, which halts the pipeline and caches until the next interrupt. The processor supports optional clock gating during the power down period by providing a clock-enable signal that can be tied to an external clock gate cell, and by providing a separate clock input for the small part of the CPU that needs to run during power-down to check for wake-up conditions and maintain cache coherency.

102.1.9 Multi-processor support

LEON3 is designed to be used in multi-processor systems. Each processor has a unique index to allow processor enumeration. The write-through caches and snooping mechanism guarantees cache coherency in shared-memory systems.

102.2 LEON3 integer unit

102.2.1 Overview

The LEON3 integer unit implements the integer part of the SPARC V8 instruction set. The implementation is focused on high performance and low complexity. The LEON3 integer unit has the following main features:

- 7-stage instruction pipeline
- Separate instruction and data cache interface
- Support for 2 - 32 register windows
- Hardware multiplier with optional 16x16 bit MAC and 40-bit accumulator
- Radix-2 divider (non-restoring)
- Static branch prediction
- Single-vector trapping for reduced code size

Figure 259 shows a block diagram of the integer unit.

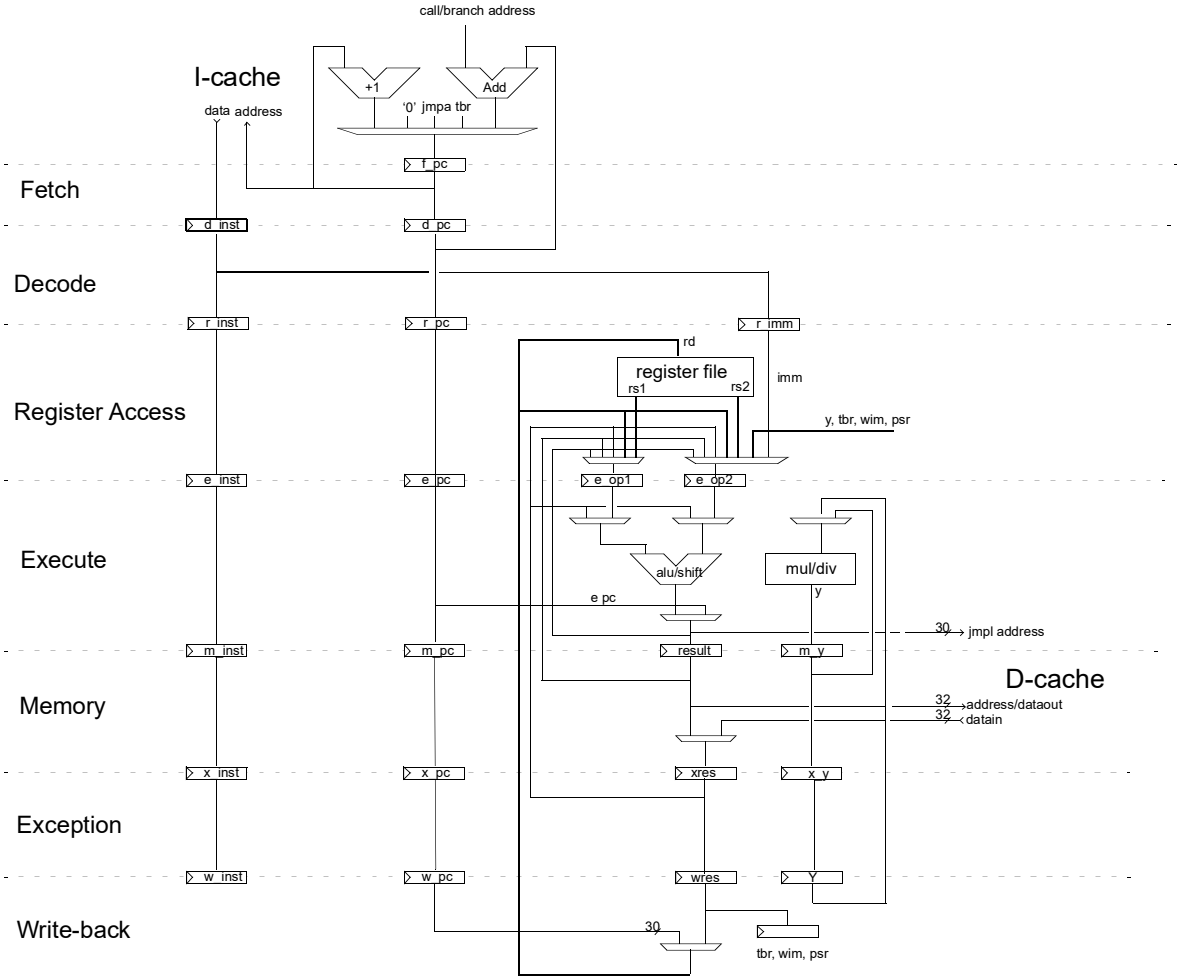


Figure 259. LEON3 integer unit datapath diagram

102.2.2 Instruction pipeline

The LEON3 integer unit uses a single instruction issue pipeline with 7 stages:

1. FE (Instruction Fetch): If the instruction cache is enabled, the instruction is fetched from the instruction cache. Otherwise, the fetch is forwarded to the AHB bus. The instruction is valid at the end of this stage and is latched inside the IU.
2. DE (Decode): The instruction is decoded and the CALL and Branch target addresses are generated.
3. RA (Register access): Operands are read from the register file or from internal data bypasses.
4. EX (Execute): ALU, logical, and shift operations are performed. For memory operations (e.g., LD) and for JMPL/RETT, the address is generated.
5. ME (Memory): Data cache is read or written at this time.
6. XC (Exception) Traps and interrupts are resolved. For cache reads, the data is aligned as appropriate.
7. WR (Write): The result of any ALU, logical, shift, or cache operations are written back to the register file.

Table 1954 lists the cycles per instruction (assuming cache hit and no icc or load interlock):

*Table 1954.*Instruction timing

Instruction	Cycles (MMU disabled)	Cycles (MMU fast-write)	Cycles (MMU slow-write)
JMPL	3[1]	3[1]	3[1]
JMPL,RETT pair	4	4	4
Double load	2	2	2
Single store	2	2	4
Double store	3	3	5
SMUL/UMUL	1/4[2]	1/4[2]	1/4 [2]
SDIV/UDIV	35	35	35
Taken Trap	5	5	5
Atomic load/store	3	3	5
All other instructions	**1**	**1**	**1**

[1] Assuming instruction in JMPL delay slot takes one cycle. Additional cycles spent in the delay slot will reduce the effective time of the JMPL to 2 or 1.

[2] Multiplication cycle count is 1 clock (1 clock issue rate, 2 clock data latency), for the 32x32 multiplier and 4 clocks (issue rate, 4/5 clocks data latency for standard/pipelined version) for the 16x16 version.

A number of conditions can extend an instruction's duration in the pipeline:

Branch interlock: When a conditional branch or trap is performed 1-2 cycles after an instruction which modifies the condition codes, 1-2 cycles of delay is added to allow the condition to be computed. If static branch prediction is enabled, this extra delay is incurred only if the branch is not taken.

Load delay: When using data resulting on a load shortly after the load, the instruction will be delayed to satisfy the pipeline's load delay. The processor pipeline can be configured for one or two cycles load delay. One cycle load delay will improve performance at a fixed speed but may degrade maximum clock frequency due to added forwarding paths in the pipeline.

Mul latency: For pipelined multiplier implementations there is 1 cycle extra data latency, accessing the result immediately after a MUL or MAC will then add one cycle pipeline delay.

Hold cycles: During cache miss processing or when blocking on the store buffer, the pipeline will be held still until the data is ready, effectively extending the execution time of the instruction causing the miss by the corresponding number of cycles. Note that since the whole pipeline is held still, hold cycles will not mask load delay or interlock delays. For instance on a load cache miss followed by a data-dependent instruction, both hold cycles and load delay will be incurred.

FPU/Coprocessor: The floating-point unit or coprocessor may need to hold the pipeline or extend a specific instruction. When this is done is specific to the FP/CP unit.

102.2.3 SPARC Implementor's ID

Frontgrade Gaisler is assigned number 15 (0xF) as SPARC implementor's identification. This value is hard-coded into bits 31:28 in the %psr register. The version number for LEON3 is 3, which is hard-coded in to bits 27:24 of the %psr.

102.2.4 Divide instructions

Full support for SPARC V8 divide instructions is provided (SDIV, UDIV, SDIVCC & UDIVCC). The divide instructions perform a 64-by-32 bit divide and produce a 32-bit result. Rounding and overflow detection is performed as defined in the SPARC V8 manual.

The divide instruction is required for full SPARC V8 compliance but can be configured out to save area using the *v8* VHDL generic.

102.2.5 Multiply instructions

The LEON processor supports the SPARC integer multiply instructions UMUL, SMUL UMULCC and SMULCC. These instructions perform a 32x32-bit integer multiply, producing a 64-bit result. SMUL and SMULCC performs signed multiply while UMUL and UMULCC performs unsigned multiply. UMULCC and SMULCC also set the condition codes to reflect the result. The multiply instructions are performed using a 32x32 pipelined hardware multiplier, or a 16x16 hardware multiplier which is iterated four times. To improve the maximum frequency on slow technologies, the 16x16 multiplier can optionally be provided with a pipeline stage.

The multiply instruction is required for full SPARC V8 compliance, but can be configured out to save area using the *v8* VHDL generic.

102.2.6 Multiply and accumulate instructions

To accelerate DSP algorithms, two multiply&accumulate instructions are implemented: UMAC and SMAC. The UMAC performs an unsigned 16-bit multiply, producing a 32-bit result, and adds the result to a 40-bit accumulator made up by the 8 lsb bits from the %y register and the %asr18 register. The least significant 32 bits are also written to the destination register. SMAC works similarly but performs signed multiply and accumulate. The MAC instructions execute in one clock but have two clocks latency, meaning that one pipeline stall cycle will be inserted if the following instruction uses the destination register of the MAC as a source operand.

The UMAC and SMAC instructions occupy the unused opcodes op=10,op3=1111110 for UMAC and op=10,op3=1111111 for SMAC.

Assembler syntax:

```
umac rs1, reg_imm, rd
smac rs1, reg_imm, rd
```

Operation:

```
prod[31:0] = rs1[15:0] * reg_imm[15:0]
result[39:0] = (Y[7:0] & %asr18[31:0]) + prod[31:0]
(Y[7:0] & %asr18[31:0]) = result[39:0]
rd = result[31:0]
```

%asr18 can be read and written using the RDASR and WRASR instructions.

The MAC instructions are an optional extension to SPARC V8, and enabled using the *mac* VHDL generic. The multiply and accumulate support also requires MUL/DIV support enabled by the *v8* VHDL generic and can only be used together with a 16x16 multiplier.

102.2.7 Compare and Swap instruction (CASA)

LEON3 implements the SPARC V9 Compare and Swap Alternative (CASA) instruction. The CASA instruction operates as described in the SPARC V9 manual. The instruction is privileged but setting ASI = 0xA (user data) will allow it to be used in user mode.

Software can determine if the processor supports CASA by checking the NOTAG field of the %asr17 register described in section 102.11.2.

102.2.8 Branch prediction

Static branch prediction can be optionally be enabled, and reduces the penalty for branches preceded by an instruction that modifies the integer condition codes (see section 102.2.2). The predictor uses a branch-always strategy, and starts fetching instruction from the branch address. On a prediction hit, 1 or 2 clock cycles are saved, and there is no extra penalty incurred for misprediction as long as the branch target can be fetched from cache. Branch prediction improves the performance with 10 - 20% on most control-type applications.

102.2.9 Register file data protection

Register file data protection is available for the fault-tolerant version of the LEON3 and is enabled via a VHDL generic. Register file data protection is described in section 102.9.2.

102.2.10Hardware breakpoints

The integer unit can be configured to include up to four hardware breakpoints. Each breakpoint consists of a pair of ancillary state registers (see section 102.11.3). Any binary aligned address range can be watched for instruction or data access, and on a breakpoint hit, trap 0x0B is generated.

102.2.11Instruction trace buffer

The (optional) instruction trace buffer consists of a circular buffer that stores executed instructions. This is enabled and accessed only through the processor's debug port via the Debug Support Unit. When enabled, the following information is stored in real time, without affecting performance:

- Instruction address and opcode
- Instruction result
- Load/store data and address
- Trap information
- 30-bit time tag

The operation and control of the trace buffer is further described in section 32.4. Note that in multi-processor systems, each processor has its own trace buffer allowing simultaneous tracing of all instruction streams.

The size of the trace buffer is configurable from 1 to 64 KiB through the *tbuf* VHDL generic. If the value of tbuf is in the 65-128 range, a two-port instruction trace buffer of size *tbuf*-64 KiB will be used, allowing contextual reading of instructions while tracing is ongoing.

102.2.12Processor configuration register

The ancillary state register 17 (%asr17) provides information on how various configuration options were set during synthesis. This can be used to enhance the performance of software, or to support enumeration in multi-processor systems. See section 102.11.2 for layout.

102.2.13Exceptions

LEON3 adheres to the general SPARC trap model. The table below shows the implemented traps and their individual priority. When PSR (processor status register) bit ET=0, an exception trap causes the processor to halt execution and enter error mode, and the external error signal will then be asserted.

*Table 1955.*Trap allocation and priority

Trap	TT	Pri	Description	Class
reset	0x00	1	Power-on reset	Interrupting
data_store_error	0x2b	2	write buffer error during data store	Interrupting
instruction_access_exception	0x01	3	Error or MMU page fault during instruction fetch	Precise
privileged_instruction	0x03	4	Execution of privileged instruction in user mode	Precise
illegal_instruction	0x02	5	UNIMP or other un-implemented instruction	Precise
fp_disabled	0x04	6	FP instruction while FPU disabled	Precise
cp_disabled	0x24	6	CP instruction while Co-processor disabled	Precise
watchpoint_detected	0x0B	7	Hardware breakpoint match	Precise
window_overflow	0x05	8	SAVE into invalid window	Precise
window_underflow	0x06	8	RESTORE into invalid window	Precise
r_register_access_error	0x20	9	register file EDAC error (LEON3FT only)	Interrupting
mem_address_not_aligned	0x07	10	Memory access to un-aligned address	Precise
fp_exception	0x08	11	FPU exception	Deferred
cp_exception	0x28	11	Co-processor exception	Deferred
data_access_exception	0x09	13	Access error during data load, MMU page fault	Precise
tag_overflow	0x0A	14	Tagged arithmetic overflow	Precise
division_by_zero	0x2A	15	Divide by zero	Precise
trap_instruction	0x80 - 0xFF	16	Software trap instruction (TA)	Precise
interrupt_level_15	0x1F	17	Asynchronous interrupt 15	Interrupting
interrupt_level_14	0x1E	18	Asynchronous interrupt 14	Interrupting
interrupt_level_13	0x1D	19	Asynchronous interrupt 13	Interrupting
interrupt_level_12	0x1C	20	Asynchronous interrupt 12	Interrupting
interrupt_level_11	0x1B	21	Asynchronous interrupt 11	Interrupting
interrupt_level_10	0x1A	22	Asynchronous interrupt 10	Interrupting
interrupt_level_9	0x19	23	Asynchronous interrupt 9	Interrupting
interrupt_level_8	0x18	24	Asynchronous interrupt 8	Interrupting
interrupt_level_7	0x17	25	Asynchronous interrupt 7	Interrupting
interrupt_level_6	0x16	26	Asynchronous interrupt 6	Interrupting
interrupt_level_5	0x15	27	Asynchronous interrupt 5	Interrupting
interrupt_level_4	0x14	28	Asynchronous interrupt 4	Interrupting
interrupt_level_3	0x13	29	Asynchronous interrupt 3	Interrupting
interrupt_level_2	0x12	30	Asynchronous interrupt 2	Interrupting
interrupt_level_1	0x11	31	Asynchronous interrupt 1	Interrupting

The prioritization follows the SPARC V8 manual in the regular non-FT case. In the FT case there is a minor difference for r_register_access_error, which has lower priority than window_over/underflow because the window condition is detected before the register file is accessed on the LEON3FT.

The fp/cp_exception traps may be either deferred or precise depending on implementation, for the GRFPU and GRFPU-Lite implementations they are deferred.

The data_store_error is delivered as a deferred exception but is non-resumable and therefore classed as interrupting. Likewise, r_register_access_error is delivered as a precise trap but since it is non-resumable it is classed as an interrupting trap.

102.2.14Single vector trapping (SVT)

Single-vector trapping (SVT) is an SPARC V8e option to reduce code size for embedded applications. When enabled, any taken trap will always jump to the reset trap handler (%tbr.tba + 0). The trap type will be indicated in %tbr.tt, and must be decoded by the shared trap handler. SVT is enabled by setting bit 13 in %asr17. The model must also be configured with the VHDL generic *svt* = 1

102.2.15Address space identifiers (ASI)

In addition to the address, a SPARC processor also generates an 8-bit address space identifier (ASI), providing up to 256 separate, 32-bit address spaces. During normal operation, the LEON3 processor accesses instructions and data using ASI 0x8 - 0xB as defined in the SPARC manual. Using the LDA/STA instructions, alternative address spaces can be accessed. The different available ASIs are described in section 102.10.

102.2.16Partial WRPSR

Partial write %PSR (WRPSR) is a SPARC V8e option that allows WRPSR instructions to only affect the %PSR.ET field. If the processor has been implemented with support for partial WRPSR and the WRPSR instruction's rd field is non-zero, then the WRPSR write will only update ET.

The model must be configured with the VHDL generic pwrpsr = 1 for partial WRPSR to be supported. Implementations that do not support partial WRPSR will write the full %PSR register regardless of the value of the WRPSR instruction's rd field.

102.2.17Alternative window pointer

Alternative window pointer (AWP) is a SPARC V8e option intended to reduce interrupt latency by allowing code that manipulates the current window pointer, mainly window over and underflow handlers and context switching code, to run with traps enabled.

Two bits are added to the PSR register, AW (alternative window) and PAW (previous alternative window). Also an AWP (alternative window pointer) field is added in an ASR register.

When the AW bit is set, the current register window used for reading/writing non-global registers is taken from the AWP register field instead of the normal CWP register field, and SAVE and RESTORE operations modify the AWP field instead of the CWP. SAVE and RESTORE can do not trigger the window over/underflow traps while AW is set.

When both AW and PAW are zero, the AWP field is kept equal to the CWP field.

When a trap occurs, the value of AW is copied into the PAW field, and AW is cleared. When returning from a trap using the RETT instruction, the PAW field is copied back into AW. The RETT will not trigger the window underflow trap if PAW is set regardless of if CWP or AWP point to an invalid window.

102.2.18Register file partitioning

Register file partitioning is an optional extension (enabled through the altwin generic) to allow a subrange of the register windows to be used as if it was the whole register file. The selected subset is connected in a ring so that the outs of the lowest register window is aliased to the ins of the highest register window in the range. Other register windows outside this range are not accessible and will be kept at their old values while the partitioning is enabled.

The partitioning is activated by setting the STWIN and CWPMAX fields of the %asr20 register. This selects the subset of windows between STWIN and STWIN+CWPMAX so that they map to CWP

values 0 to CWPMAX. STWIN and CWPMAX must be set so they map to a valid range, CWPMAX+STWIN must not exceed the highest possible CWP value supported in the normal case. Also, for correct operation, CWP must be set to a value between 0 and CWPMAX before accessing any non-global register.

Writing CWPMAX to (otherwise illegal value) 0 in %asr20 will result in writing only AWP and keeping the values of STWIN and CWPMAX.

A special write-only bit in the %asr20 register can be used to write CWP in the PSR at the same time as writing the STWIN,CWPMAX,AWP fields, this is intended to allow switching between two register file partitions without disabling interrupts.

The WIM register is not managed by the partitioning logic, therefore the lowest bits of the WIM will map to the partitioned windows. The highest bits of the WIM will be masked to 0 on read to simulate a smaller register file, however these bits are still writable.

102.2.19Power-down

The processor can be configured to include a power-down feature to minimize power consumption during idle periods. The power-down mode is entered by performing a WRASR instruction to %asr19:

```
wr %g0, %asr19
```

During power-down, the pipeline is halted until the next interrupt occurs. Signals inside the processor pipeline and caches are then static, reducing power consumption from dynamic switching.

Note: %asr19 must always be written with the data value zero to ensure compatiblity with future extensions.

Note: This instruction must be performed in supervisor mode with interrupts enabled.

When resuming from power-down, the pipeline will be re-filled from the point of power-down and the first instruction following the WRASR instruction will be executed prior to taking the interrupt trap. Up to six instructions after the WRASR instruction will be fetched (possibly with cache miss if they are not in cache) prior to fetching the trap handler.

102.2.20Processor reset operation

The processor is reset by asserting the RESET input for at least 4 clock cycles. The following table indicates the reset values of a subset of the registers which are affected by the reset..

*Table 1956.*Processor reset values

Register	Reset value
Trap Base Register	Trap Base Address field reset (value given by *rstaddr* VHDL generic)
PC (program counter)	0x0 (*rstaddr* VHDL generic)
nPC (next program counter)	0x4 (*rstaddr* VHDL generic + 4)
PSR (processor status register)	ET=0, S=1

By default, the execution will start from address 0. This can be overridden by setting the *rstaddr* VHDL generic in the model to a non-zero value. The reset address is always aligned on a 4 KiB boundary. If *rstaddr* is set to 16#FFFFF#, then the reset address is taken from the signal IRQI.RSTVEC. This allows the reset address to be changed dynamically.

102.2.21Multi-processor systems

In multiprocessor systems, the ID of the processor on which the code is executing can be read out by reading the index field of the LEON3 configuration register. Only processor 0 starts executing after reset, the others are in power-down mode and are activated by a signal from the interrupt controller.

102.2.22LEON-REX extension

The processor can be built with support for the LEON-REX addition to the SPARC instruction set, allowing a more compact code representation than the regular SPARC machine code. The details of the extension are given in a separate document. The extension is implemented when the *rex* VHDL generic is set.

Detection of whether support is present can be done by checking the REXV field in the asr17 register (see section 102.11.2). The REX support can be set to enabled, illegal or transparent mode via the REXEN/REXILL bits in the asr17 register, after reset the default setting is illegal so any LEON-REX code will cause an illegal instruction trap.

The extension is implemented as a decompressor internally inside the pipeline and does not affect the behavior of the caches, MMU or AHB bus interfaces.

When the *rex* generic is set, the instruction trace buffer entries are changed so the two most significant bits of the time tag are instead used to represent REX mode enabled status, and bit 1 of the program counter. The instructions opcodes logged into the trace buffer are the regular SPARC opcodes that are generated internally in the pipeline, not the LEON-REX opcodes that are in memory and cache.

102.3 Cache system

102.3.1 Overview

The LEON3 processor pipeline implements a Harvard architecture with separate instruction and data buses, connected to two separate cache controllers. As long as the execution does not cause a cache miss, the cache controllers can serve one 32-bit instruction fetch and one 32-bit data load/store per cycle, keeping the pipeline running at full speed. Each cache controller can be configured with different sizes and replacement policy, and it is also possible to attach local RAM to each cache controller.

On cache miss, the cache controller will assert a hold signal freezing the IU pipeline, and after delivering the data the hold signal is again lifted so execution continues. For accessing the bus, the cache controllers share the same AHB connection to the on-chip bus. Certain parts of the MMU (table walk logic, and depending on configuration also TLB buffer) are also shared between the two caches.

Another important component included in the data cache is the write buffer, allowing stores to proceed in parallel to executing instructions.

Cachability (memory areas that are cachable) for both caches is controlled either through the AHB plug&play address information or using a VHDL generic, see section 102.7.2.

102.3.2 Cache operation

Each cache controller has two main memory blocks, the tag memory and the data memory. At each address in the tag memory, a number of cache entries, ways, are stored for a certain set of possible memory addresses. The data memory stores the data for the corresponding ways.

For each way, the tag memory contains the following information:

- Valid bits saying if the entry contains valid data or is free. The I-cache has one valid bit per word (instruction) while the D-cache has a single valid bit for the whole cache line.
- The tag, all bits of the cached memory address that are not implied by the set
- If MMU is enabled, the context ID of the cache entry
- If MMU is enabled with *mmuen* generic set to 2, SO bit (supervisor only access)
- If locking support is enabled, a lock bit for that entry
- If LRR is used, a bit specifying the replacement order
- If FT is enabled, check bits for detecting errors

When a read from cache is performed, the tags and data for all cache ways of the corresponding set are read out in parallel, the tags and valid bits are compared to the desired address and the matching way is selected. In the hit case, this is all done in the same cycle to support the full execution rate of the processor.

In the miss case, the cache will at first deliver incorrect data. However on the following cycle, a hold signal will be asserted to prevent the processor from proceeding with that data. After the miss has been processed, the correct data is injected into the pipeline using a memory data strobe (mds) signal, and afterwards the hold signal can be released. If the missed address is cacheable, then the data read in from the cache miss will be stored into the cache, possibly replacing one of the existing ways.

The miss case is handled slightly differently for the I and D caches. Depending on the instruction burst fetch configuration bit, the instruction cache will either read the single missed instruction, or stream in data from the missed address until to the end of that cache line. The valid bits in the cache will reflect which words in the line were actually read in. The D-cache will always fetch a whole cache line on miss and therefore only has one valid bit.

In the instruction streaming case, the processor pipeline is stepped one step for every received instruction. If the processor needs extra pipeline cycles to stretch a multi-cycle instruction, or due to an interlock condition (see section 102.2), or if the processor jumps/branches away, then the instruction cache will hold the pipe, fetch the remainder of the cache line, and the pipeline will then proceed normally.

102.3.3 Cache configuration options

Each cache controller can be configured to implement a single-way (direct-mapped) cache or a multi-way cache with set associativity of 2 - 4. The way size is configurable to 1 - 256 KiB, divided into cache lines with 16 or 32 bytes of data.

In multi-way configurations, one of three replacement policies can be selected:

- Least-recently-used (LRU) - This maintains the order of usage for each set in the cache and replaces the one that has was used last. The LRU information needs to be updated on every cache hit and is therefore not stored with the tags but in separate flip flops.
- Least-recently-replaced (LRR) - This stores the index of the oldest replaced way along with the tags and uses this to select which way to replace. This policy can only be implemented when the number of ways is set to 2.
- Pseudo-random - This method samples a free-running counter to select which way to replace. System jitter (AMBA bus delay variations) will help to randomize the selected value.

Locking support can also be added to each cache if desired. This option adds a lock bit to each tag that allows you to lock a number of ways in each set, preventing those cache entries from being replaced. Note that when using locking together with LRU and more than two ways, this will add extra lookup tables to determine which way to replace and this might become a critical path in the core.

102.3.4 Address mapping

The addresses seen by the CPU are divided into tag, index and offset bits. The index is used to select the set in the cache, therefore only a limited number of cache lines with the same index part can be stored at one time in the cache. The tag is stored in the cache and compared upon read.

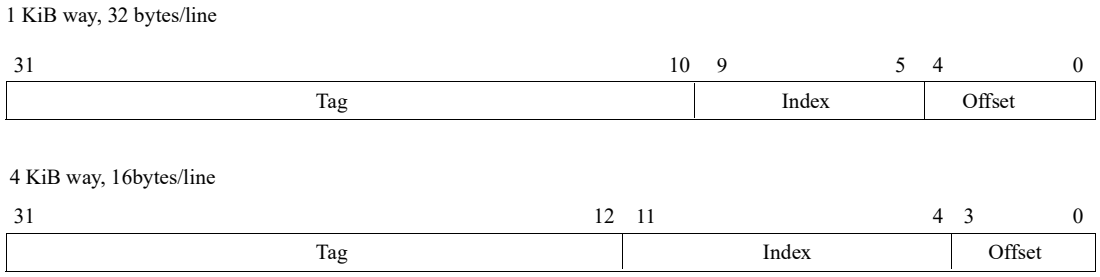


Figure 260. Cache address mapping examples

102.3.5 Data cache policy

The data cache employs a write-through policy, meaning that every store made on the CPU will propagate, via the write buffer, to the bus and there are no “dirty” lines in the cache that has not yet been written out apart from what is in the buffer. The store will also update the cache if the address is present, however a new line will not be allocated in that case

.

*Table 1957.*LEON3 Data caching behavior

Operation	In cache	Cacheable	Bus action	Cache action	Load data
Data load	No	No	Read	No change	Bus
	No	Yes	Read	Line allocated/replaced	Bus
	Yes	-	None	No change	Cache
Data load with forced cache miss (ASI 1)	No	No	Read	No change	Bus
	No	Yes	Read	Line allocated/replaced[1]	Bus
	Yes	-	Read	Data updated[1]	Bus
Data load with MMU bypass (ASI 0x1C)	-	-	Read (phys addr)	No change	Bus
Data store	No	No	Write (via buffer)	No change	(N/A)
	No	Yes	Write (via buffer)	No change	(N/A)
	Yes	-	Write (via buffer)	Data updated	(N/A)
Data store with MMU bypass (ASI 0x1C)	-	-	Write (via buffer, phys addr)	No change	(N/A)

[1] This behavior varies slightly between LEON3 versions, see section 102.13.3

102.3.6 Write buffer

The data cache contains a write buffer able to hold a single 8,16,32, or 64-bit write. For half-word or byte stores, the stored data replicated into proper byte alignment for writing to a word-addressed device. The write is processed in the background so the processor pipeline can keep executing while the write is being processed. However, any following instruction that requires bus access will block until the write buffer has been emptied. Loads served from cache will however not block, due to the cache policy used there can not be a mismatch between cache data and store buffer (the effect of this behavior on SMP systems is discussed in section 102.8).

Since the processor executes in parallel with the write buffer, a write error will not cause an exception to the store instruction. Depending on memory and cache activity, the write cycle may not occur until

several clock cycles after the store instructions has completed. If a write error occurs, the currently executing instruction will take trap 0x2b. This trap can be disabled using the DWT configuration (see section 102.11.2).

Note: a 0x2b trap handler should flush the data cache, since a write hit would update the cache while the memory would keep the old value due the write error.

102.3.7 Operating with MMU

When MMU is enabled, the virtual addresses seen by the running code no longer correspond directly to the physical addresses on the AHB bus. The cache uses tags based on the virtual addresses, as this avoids having to do any additional work to translate the address in the most timing-critical hit case. However, any time a bus access needs to be made, a translation request has to be sent to the MMU to convert the virtual address to a physical address. For the write buffer, this work is included in the background processing of the store. The translation request to the MMU may result in memory accesses from the MMU to perform table walk, depending on the state of the MMU.

The MMU context ID is included in the cache tags in order to allow switching between multiple MMU contexts mapping the same virtual address to different physical addresses. Note that the cache does not detect aliases to the same physical address so in that case the same physical address may be cached in multiple ways (also see snooping below).

102.3.8 Snooping

The data cache can be configured to support AHB bus snooping. The AHB bus the processor is connected to, is monitored for writes from other masters to an address that is in the cache. If a write is done to a cached address, that cache line is marked invalid and the processor will be forced to fetch the (new) data from memory the next time it is read.

For using snooping together with the MMU, an extra tag memory storing physical tags must be added to allow comparing with the physical address on the AHB bus (called separate snoop tags or physical tags).

The processor can snoop on itself and invalidate any other cache lines aliased to the same physical address in case there are multiple virtual mappings to the same physical address that is being written. However, note that this does not happen until the write occurs on the bus so the other virtual aliases will return the old data in the meantime. Note also that the behavior is not available in older LEON3 version.

Snooping requires the way size of the cache to be equal or smaller than the MMU page size, otherwise the index into the physical and virtual tag RAM:s may not match, resulting in aliasing problems.

102.3.9 Enabling and disabling cache

Both I and D caches are disabled after reset. They are enabled by writing to the cache control register (see 102.11.6). Before enabling the caches after a reset they must be flushed to ensure that all tags are marked invalid.

102.3.10Cache freeze

Each cache can be in one of three modes: disabled, enabled and frozen. If disabled, no cache operation is performed and load and store requests are passed directly to the memory controller. If enabled, the cache operates as described above. In the frozen state, the cache is accessed and kept in sync with the main memory as if it was enabled, but no new lines are allocated on read misses.

If the DF or IF bit is set, the corresponding cache will be frozen when an asynchronous interrupt is taken. This can be beneficial in real-time system to allow a more accurate calculation of worst-case execution time for a code segment. The execution of the interrupt handler will not evict any cache lines and when control is returned to the interrupted task, the cache state is identical to what it was

before the interrupt. If a cache has been frozen by an interrupt, it can only be enabled again by enabling it in the CCR. This is typically done at the end of the interrupt handler before control is returned to the interrupted task.

102.3.11Flushing

Both instruction and data cache are flushed either by executing the FLUSH instruction, setting the FI/FD bits in the cache control register, or by writing to certain ASI address spaces.

Cache flushing takes one clock cycle per cache set, during which the IU will not be halted, but during which the caches are disabled. When the flush operation is completed, the cache will resume the state (disabled, enabled or frozen) indicated in the cache control register. Diagnostic access to the cache is not possible during a flush operation and will cause a data exception (trap=0x09) if attempted.

Note that while the SPARC V8 specifies only that the instructions pointed to by the FLUSH argument will be flushed, the LEON3 will additionally flush the entire I and D cache (which is permitted by the manual as the additional flushing only affects performance and not operation). While the LEON3 currently ignores the address argument, it is recommended for future compatibility to only use the basic flush %g0 form if you want the full flush behavior.

102.3.12Locking

In a multi-way configuration the instruction and data cache controllers can be configured with optional lock bit in the cache tag. Setting the lock bit prevents the cache line to be replaced by the replacement algorithm. A cache line is locked by performing a diagnostic write to the instruction tag (ASI 0xC, see 102.10.5) on the cache offset of the line to be locked, setting the Address Tag field to the address tag of the line to be locked, enabling the lock bit and clearing the valid bits. The locked cache line will be updated on a read-miss and will remain in the cache until the line is unlocked.

For correct operation, the ways must always be locked in ascending order, for example if two ways are locked it must always be way 0 and 1. Note also that you must always keep one way unlocked in each set for the cache to function correctly in case of miss to that set.

For run-time detection of this feature, setting the lock bit in a cache tag and reading the same tag will show if the cache line locking was enabled during the LEON3 configuration: the lock bit will be set if the cache line locking was enabled otherwise it will be 0.

Since the lock bit is not involved in the parity calculation, the locking option is not compatible with the fault tolerant configurations of the cache and therefore it should not be used if cmft > 0.

102.3.13Diagnostic access

The cache tag and data contents can be directly accessed for diagnostics and for locking purposes via various ASI:s, see section 102.10.5.

102.3.14Local scratch pad RAM

Local instruction and data ram can optionally be attached to the cache controllers. The ram can be between 1 - 256 KiB, and mapped on any 16 MiB block in the address space. Accesses performed to the local RAMs are not cached, and will not appear on the AHB bus or in the write buffer. The default address for the instruction ram is 0x8e000000, and for the data ram 0x8f000000, but these can be reconfigured using VHDL generics.

The local instruction RAM is intended for executing instructions and will serve instructions without any wait states. It can be accessed (with a few wait states) through 32-bit load and store instructions only, and is therefore not suitable for general application data. Initializing the instruction local RAM is done from software via stores as that is the only way to write into it.

The local data RAM will serve data accesses of any size without adding wait states. It can however not serve instructions.

The local instruction/data RAM implementation is not compatible with the MMU functionality, as it will “shadow” the virtual address space regardless of page table setup.

Since the local RAM implementation does not support error protection, it is not applicable for FT configurations of the LEON3.

102.3.15Fault tolerance support

The cache memories (tags and data) can optionally be protected against soft errors using byte-parity-codes. Enabling of the data protection is done through the *cmft* VHDL generic and the functionality is only enabled for users that have licensed the fault-tolerant version of LEON3. Cache memory data protection is further described in section 102.9.6.

102.4 Memory management unit

102.4.1 Overview

A memory-management unit can optionally be enabled. This is compatible with the SPARC V8 reference MMU (SRMMU) architecture described inthe SPARC V8 manual, appendix H.

The MMU provides address translation of both instructions and data via page tables stored in memory.When needed, the MMU will automatically access the page tables to calculate the correct physical address. The latest translations are stored in a special cache called the translation lookaside buffer (TLB), also referred to as Page Descriptor Cache (PDC) in the SRMMU specification. The MMU also provides access control, making it possible to “sandbox” unpriviledged code from accessing the rest of the system.

102.4.2 MMU/Cache operation

When the MMU is disabled, the MMU is bypassed and the caches operate with physical address mapping. When the MMU is enabled, the cache tags store the virtual address and also include an 8-bit context field. Both the tag address and context field must match to generate a cache hit. When *mmuen* generic is set to 2 and MMU is enabled, the cache tags store a bit in addition to context called SO bit (supervisor only access). The SO bit is used to check the access permission of the data and instructions that resides in the level-1 caches when MMU is enabled. Without SO bit, access permissions of the load operations that hit in the data cache or the instruction accesses that hit in the instruction cache will not be checked properly.

If cache snooping is used, physical tags (separate snoop tags) must be enabled for it to work when address translation is used, see section 102.3.8.

Because the cache is virtually tagged, no extra clock cycles are needed in case of a cache load or instruction cache hit. In case of miss or write buffer processing, a translation is required that might add extra latency to the processing time, depending on TLB configuration and if there is a TLB miss.

The TLB can be configured in three different ways:

- Separate TLBs, slow access. TLB lookup adds 2 extra clock cycles.
- Shared TLB, slow access. TLB lookup adds 2 extra clock cycles, the TLB may be used by the other cache, leading to up to 4 extra cycles lookup time in the worst case.
- Separate TLBs, fast access. TLB lookup is done at the same time as tag lookup and therefore add no extra clock cycles.

If there is a TLB miss the page table must be traversed, resulting in up to four AMBA read accesses and one possible writeback operation. See the SRMMU specification for the exact format of the page table.

An MMU page fault will generate trap 0x09 for the D-cache and trap 0x01 for the I cache, and update the MMU status registers according to table 1958 and the SRMMU specification. In case of multiple

errors, they fault type values are prioritized as the SRMMU specification requires. The cache and memory will not be modified on an MMU page fault.

*Table 1958.*LEON3 MMU Fault Status Register, fault type values

Fault type	SPARC V8 ref	Priority	Condition
6	Internal error	1	Never issued by LEON SRMMU
4	Translation error	2	AHB error response while performing table walk. Translations errors as defined in SPARC V8 manual. A translation error caused by an AMBA ERROR response will overwrite all other errors. Other translation errors do no overwrite existing translation errors when FAV = 1.
1	Invalid address error	3	Page table entry for address was marked invalid
3	Privlege violation error	4	Access denied based on page table and su status (see SRMMU spec for how privilege and protection error are prioritized)
2	Protection error	5	
0	None	-	No error (inside trap this means the trap occurred when fetching the actual data)

102.4.3 Translation look-aside buffer (TLB)

The MMU can be configured to use a shared TLB, or separate TLB for instructions and data. The number of TLB entries (for each implemented TLB) can be set to 2 - 64 via VHDL generics. The organisation of the TLB and number of entries is not visible to the software and does thus not require any modification to the operating system. The TLB can be flushed using an STA instruction to ASI 0x18, see section 102.10.7.

102.4.4 Variable minimum page sizes

The standard minimum page size for the SRMMU is 4 KiB. The minimum page size can also be configured to 8, 16 or 32 KiB in order to allow for large data cache ways. The page size can either be configured hard at implementation time or made software-configurable via the MMU control register. The page sizes for level 1, 2 and 3 is seen in the table below:

*Table 1959.*MMU page size

Scheme	Level-1	Level-2	Level-3
4 KiB (default)	16 MiB	256 KiB	4 KiB
8 KiB	32 MiB	512 KiB	8 KiB
16 KiB	64 MiB	1 MiB	16 KiB
32 KiB	256 MiB	2 MiB	32 KiB

The layouts of the indexes are chosen so that PTE page tables can be joined together inside one MMU page without leaving holes.

Note that most operating systems are hard-coded for a specific page size and using one other than 4 KiB usually requires reconfiguration/recompilation of the operating system kernel.

102.5 Floating-point unit

The SPARC V8 architecture defines two (optional) co-processors: one floating-point unit (FPU) and one user-defined co-processor. Two different FPU's can be interfaced the LEON3 pipeline: Frontgrade Gaisler's GRFPU and GRFPU-Lite. Selection of which FPU to use is done through the fpu generic. The characteristics of the FPU's are described in the next sections.

102.5.1 Frontgrade Gaisler's floating-point unit (GRFPU)

The high-performance GRFPU operates on single- and double-precision operands, and implements all SPARC V8 FPU operations including square root and division. The FPU is interfaced to the LEON3 pipeline using a LEON3-specific FPU controller (GRFPC) that allows FPU instructions to be executed simultaneously with integer instructions. Only in case of a data or resource dependency is the integer pipeline held. The GRFPU is fully pipelined and allows the start of one instruction each clock cycle, with the exception is FDIV and FSQRT which can only be executed one at a time. The FDIV and FSQRT are however executed in a separate divide unit and do not block the FPU from performing all other operations in parallel.

All instructions except FDIV and FSQRT has a latency of three cycles, but to improve timing, the LEON3 FPU controller inserts an extra pipeline stage in the result forwarding path. This results in a latency of four clock cycles at instruction level. The table below shows the GRFPU instruction timing when used together with GRFPC:

*Table 1960.*GRFPU instruction timing with GRFPC

Instruction	Throughput	Latency
FADDS, FADDD, FSUBS, FSUBD,FMULS, FMULD, FSMULD, FITOS, FITOD, FSTOI, FDTOI, FSTOD, FDTOS, FCMPS, FCMPD, FCMPES. FCMPED	1	4
FDIVS	14	16
FDIVD	15	17
FSQRTS	22	24
FSQRTD	23	25

The GRFPC controller implements the SPARC deferred trap model, and the FPU trap queue (FQ) can contain up to 7 queued instructions when an FPU exception is taken. When the GRFPU is enabled in the model, the version field in %fsr has the value of 2.

The GRFPU does not handle denormalized numbers as inputs and will in that case cause an fp_exception with the FPU trap type set to unfinised_FPOP (tt=2). There is a non-standard mode in the FPU that will instead replace the denormalized inputs with zero and thus never create this condition.

102.5.2 GRFPU-Lite

GRFPU-Lite is a smaller version of GRFPU, suitable for FPGA implementations with limited logic resources. The GRFPU-Lite is not pipelined and executes thus only one instruction at a time. To improve performance, the FPU controller (GRLFPC) allows GRFPU-Lite to execute in parallel with the processor pipeline as long as no new FPU instructions are pending. Below is a table of worst-case throughput of the GRFPU-Lite:

*Table 1961.*GRFPU-Lite worst-case instruction timing with GRLFPC

Instruction	Throughput	Latency
FADDS, FADDD, FSUBS, FSUBD,FMULS, FMULD, FSMULD, FITOS, FITOD, FSTOI, FDTOI, FSTOD, FDTOS, FCMPS, FCMPD, FCMPES. FCMPED	8	8
FDIVS	31	31
FDIVD	57	57
FSQRTS	46	46
FSQRTD	65	65

The GRLFPC controller implements the SPARC deferred trap model, but the FPU trap queue (FQ) can contain only one queued instructions when an FPU exception is taken.When the GRFPU-Lite is enabled in the model, the version field in %fsr has the value of 3.

102.6 Co-processor interface

No implementation for the user-defined co-processor is currently provided.

102.7 AMBA interface

102.7.1 Overview

The LEON3 processor uses one AHB master interface for all data, instruction and MMU table-walk accesses. Instructions are fetched with incremental bursts if the IB bit is set in the cache control register, otherwise single READ cycles are used. Read data is accessed using bursts for cachable data, and byte, half-word and word accesses for uncacheable data. Store data is performed using single accesses or a two-beat incremental burst in case of 64-bit store.

The HPROT signals of the AHB bus are driven to indicate if the accesses is instruction or data, and if it is a user or supervisor access.

*Table 1962.*HPROT values

Type of access	User/Super	HPROT
Instruction	User	1100
Instruction	Super	1110
Data	User	1101
Data	Super	1111
MMU	Any	1101

In case of atomic accesses, a locked access will be made on the AMBA bus to guarantee atomicity as seen from other masters on the bus.

102.7.2 Cachability control

Cachability for both caches can be controlled through the AHB plug&play address information or set manually via the *cached* VHDL generic.

For plug'n'play based cachability, the memory mapping for each AHB slave indicates whether the area is cachable, and this information is used to (statically) determine which access will be treated as cacheable. This approach means that the cachability mapping is always coherent with the current AHB configuration.

When the *cached* VHDL generic is not zero, it is treated as a 16-bit field, defining the cachability of each 256 MiB address block on the AMBA bus. For example, a value of 16#00F3# will define cachable areas in 0 - 0x20000000 and 0x40000000 - 0x80000000.

In order to access the plug'n'play information, the processor takes the ahbso vector as input. Only the static hconfig signals are used so the use of this input will be eliminated through constant propagation during synthesis.

102.7.3 Error handling

An AHB ERROR response received while fetching instructions will normally cause an instruction access exception (tt=0x1). However if this occurs during streaming on an address which is not needed, the I cache controller will just not set the corresponding valid bit in the cache tag. If the IU later fetches an instruction from the failed address, a cache miss will occur, triggering a new access to the failed address.

An AHB ERROR response while fetching data into the data cache will normally trigger a data_access_exception trap (tt=0x9). If the error was for a part of the cache line other than what was currently being requested by the pipeline, a trap is not generated and the valid bit for that line is not set.

An ERROR response during an MMU table walk will lead the MMU to set the fault type to Internal error (1) and generate an instruction or data access exception, depending on which type of access that caused the table walk.

Error responses to writes will result in the data_store_error trap as described in section 102.3.6.

102.7.4 Snoop port

For the snooping logic, the LEON3 has an ahbsi input. For correct function, this must be tied to the same AHB bus that the master interface. It is not possible to snoop on another bus or to add extra pipeline registers to the snoop port, because the snoop logic must be in sync with the master interface.

102.8 Multi-processor system support

This section gives an overview of issues when using the LEON3 in multi-processor configuration.

Using the features described in earlier sections together with a multiprocessor capable IRQ controller (IRQMP), the LEON3 processor can support symmetric multiprocessing (SMP) configurations with shared memory, with up to 16 processors attached to the same AHB bus.

Enabling SMP features (sleeping on reset for CPU 1-N) is done by setting the *smp* VHDL generic to 1 or higher. Cache snooping should always be enabled in SMP systems to maintain data cache coherency between the processors.

102.8.1 Start-up

In multiprocessor systems, only the first processor will start after reset and all other processors will remain halted in power-down mode. After the system has been initialized, the remaining processors can be started by writing to the 'multiprocessor status register', located in the multiprocessor interrupt controller. The halted processors start executing from the reset address (0 or *rstaddr* VHDL generic, see section 102.2.20).

An application in a multiprocessor system can determine which processor it is executing on by checking the processor index field in the LEON3 configuration register (%asr17). As all processors typically have the same reset start address value, boot software must check the processor index and perform processor specific setup (e.g. initialization of stack pointer) based on the value of the processor index.

In recent versions of the LEON3, and if the IRQ controller is configured with the extended boot register extension, it is possible for one processor to monitor and reboot another processor.via the interrupt controller. This requires careful software design.

For earlier versions of the LEON3, this is not supported and if software detects that one processor is unresponsive and needs to restart the processor then the full system should be reset, for example by triggering the system's watchdog, if implemented. In order for software to monitor that all processors in a system are up and running it is recommended to implement a heartbeat mechanism in software.

While it is possible to have more fine-grained control over processor behaviour via the Debug Support Unit (if implemented) this is not recommended as the debug support unit is typically disabled in production mode.

102.8.2 Shared memory model

Each processor core has it's own separate AHB master interface and the AHB controller will arbitrate between them to share access to the on-chip bus.

If caches are not used, the processors will form a sequentially consistent (SC) system, where every processor will execute it's loads, stores and atomics to memory in program order on the AHB bus and the different processors operations will be interleaved in some order through the AHB arbitration. The

shared memory controller AHB slave is assumed to not reorder accesses so a read always returns the latest written value to that location on the bus.

When using caches with snooping (and with physical tags, also called separate snoop tags, if using the MMU), the shared memory will act according to the slightly weaker SPARC Total Store Order (TSO) model. The TSO model is close to SC, except that loads may be reordered before stores coming from the same CPU. The stores and atomics are conceptually placed in a FIFO (see the diagrams in the SPARC manual) and the loads are allowed to bypass the FIFO if they are not to the same address as the stores. Loaded data from other addresses may therefore be either older or newer, with respect to the global memory order, than the stores that have been performed by the same CPU.

In the LEON3 case this happens because cache hits are served without blocking even when there is data in the write buffer. The loaded data will always return the stored data in case of reading the same address, because if it is cached, the store updates the cache before being put in the write buffer, and if it was not in cache then the load will result in a miss which waits for the write buffer to complete. Loaded data from a different address can be older than the store if it is served by cache before the write has completed, or newer if it results in a cache miss or if there is a long enough delay for the store to propagate to memory before reading.

See relevant literature on shared memory systems for more information. These details are mainly of concern for complex applications using lock-free data structures such as the Linux kernel, the recommendation for applications is to instead avoid concurrent access to shared structures by using mutexes/semaphores based on atomic instructions, or to use message passing schemes with one-directional circular buffers.

102.8.3 Memory-mapped hardware

Hardware resources (IP cores) are normally memory mapped on uncacheable address spaces. They will be accessible from all the CPU:s in a sequentially consistent manner. Since software drivers usually expect to be “alone” accessing the IP core and the IP cores register interfaces are not designed for concurrent use by multiple masters, using a bare-C application designed for single-processor usage on multiple cores at the same time will generally not work. This can be solved by partitioning the applications so that each IP core is only accessed by one of the CPU:s. This partitioning also need to be done between the interrupts so the IP core’s interrupts will be received by the correct processor.

102.9 Fault tolerance

102.9.1 Overview

The LEON3 processor can be enhanced with fault-tolerance against SEU errors (referred to as LEON3FT). The fault-tolerance is focused on the protection of on-chip RAM blocks, which are used to implement IU/FPU register files and the cache memory.

The LEON3FT is licensed separately, and in the commercial LEON3 releases setting the FT VHDL generics will not have any effect.

102.9.2 Integer register file protection

The SEU protection for the integer unit register file can be implemented in four different ways, depending on target technology and available RAM blocks. The SEU protection scheme is selected

during synthesis, using the *iuft* VHDL generic. Table 1963 below shows the implementation characteristics of the four possible SEU protection schemes.

*Table 1963.*Integer unit SEU protection schemes

ID	Implementation	Description	Usage
0	No protection (hardening at lower level)	No error checking, equivalent to non-FT version. Register file hardness must be ensured separately, for example by mapping the register file memories to SEU hardened flip-flops.	IU, GRFPU, GRFPU-Lite
1	4-bit parity with restart	4-bit checksum per 32-bit word. Detects and corrects 1 bit per byte (4 bits per word). Pipeline restart on correction.	IU, GRFPU-Lite
2	8-bit parity without restart	8-bit checksum per 32-bit word. Detects and corrects 1 bit per byte (4 bits per word). Correction on-the-fly without pipeline restart.	IU, GRFPU
3	7-bit BCH with restart	7-bit BCH checksum per 32-bit word. Detects 2 bits and corrects 1 bit per word. Pipeline restart on correction.	IU
4	Memory triplication	Memory blocks triplicated and bit by bit voted on outputs. Correction on-the-fly without pipeline restart, no error injection interface or error counters. Note that care must be taken by the implementer to ensure that the TMR is not collapsed in optimization by synthesis tools.	IU, GRFPU, GRFPU-Lite
5	7-bit BCH without restart	7-bit BCH checksum per 32-bit word. Correction on the fly without pipeline restart. Error correction logic performed on same cycle as memory read.	IU
6	Technology specific	Implement register files using native ECC capability of the technology (via syncram_2pft in the techmap library). Only valid for subset of (FPGA) technologies. Correction on-the-fly without pipeline restart. Error injection and error counters may be supported depending on technology. Technology specific protection is further documented in the GRLIB-FT User's Manual (grlib-ft.pdf).	IU

The SEU error detection has no impact on behavior, but a correction cycle (scheme 1 and 3) will delay the current instruction with 6 clock cycles. An uncorrectable error in the IU register file will cause trap 0x20 (*r_register_access_error*). A dedicated counter exists in ASR16 to count the number of register file corrections.

The register file is implemented using scheme 0 if the regfile_3p_infer array is set for the selected memory technology in the techmap library, or if bits 16-17 of memtech are set, regardless of iuft/fpft setting.

102.9.3 Integer register file scrubber

A register file scrubber function can be activated via the SCREN bit in asr16. The scrubber reads each register in the integer register file in turn as a flat memory regardless of which register window is currently active in the running program.

The scrubber works by accessing the register file whenever a branch instruction is executed and runs in parallel with the branch instruction. Since branch instructions do not read or write any registers in the register file, the scrubbing can be done without any performance overhead of the scrubber in the uncorrected case.

If a register file protection scheme with restart is used (iuft=1 or iuft=3), the scrubber will use the same correction mechanism and perform a restart of the branch instruction that the scrub iteration was

running in parallel with. For other protection schemes, the register value is simply written back with zero overhead also in the corrected case.

If the register file protection scheme used can detect uncorrectable errors and generate r_register_access_error traps (iuft=3, iuft=5 or iuft=6), such traps will be delivered on the associated branch instruction of the scrubber access in case of an uncorrectable error.

The scrubber also performs a data consistency check between the two read ports, if there is no error detected but the read data between the two read ports differ for the scrubbed address then the SCRDM bit in asr18 is set.

102.9.4 Floating-point register file protection

The FPU register file has similar SEU protection as the IU register file, but with less configuration options. The FPU register file protection scheme is enabled with the fpft generic and has the same encoding as iuft. The protection schemes that are supported for the GRFPU and the GRFPU-Lite are listed in table 1963. An uncorrectable error in the FPU register file will cause an (deferred) FPU exception with %fsr.ftt set to 5 (hardware_error).

Note that the restrictions on protection scheme is not enforced, so it is recommended to simulate the configuration with error injection to ensure that the scheme chosen is functioning correctly. It is also recommended to confirm in the netlist that the expected register file type (memory block or flip flops) was implemented.

102.9.5 Register file EDAC/parity bits diagnostic read-out and error injection

The register file parity bits can be read out via ASI 0xF as described in section 102.10.6.

For test purposes, the IU and FPU register file EDAC/parity checkbits can be modified by software. This is done by setting the ITE or FTE bits in asr16 to ‘1’. In this mode, the EDAC/parity bits are first XORed with the contents of %asr16.FTB before written to the register files. This is done for all writes into the register file while the feature is enabled, so diagnostic code must be carefully designed to avoid accidentally injecting errors in other registers than intended. Also note that due to result forwarding in the pipeline, an injected error will not affect the immediately following instructions, which might cause surprising behavior such as errors triggering only when single stepping through the code.

Bit 0 (RF EDAC disable) in %asr16 should be set to 1 during diagnostic accesses and fault injection to the register file, to avoid EDAC correction cycles or error traps.

102.9.6 Cache protection

Each word in the tag or data memories is normally protected by four check bits. Use of the SYNCRAM protection allows the processor to use technology specific protection and this can lead to savings in resource utilization on target technologies that have built-in protection of SRAM blocks. If separate physical tags for snooping are enabled, the physical tag memory is also protected. An error during cache access will cause an invalidation of that cache line, and a re-execution of the failing instruction. This will ensure that the complete cache line (tags and data) is refilled from external memory.

If snooping is enabled, an error detected in the tags while snooping a write to that set will lead to that cache data being invalidated (since the tag before the error might have matched the written address).

For every detected error, a counter in the cache control register is incremented. The counters saturate at their maximum value (3), and should be reset by software after read-out.

The cache memory check bits can be diagnostically read and modified by setting the PS bit in the cache control register and then perform a normal tag or data diagnostic access, see section 102.10.5 for details.

102.10 ASI assignments

102.10.1Summary

The table shows the ASI usage for LEON.

*Table 1964.*ASI usage

ASI	Usage
0x01	Forced cache miss.
0x02	System control registers (cache control register)
0x08, 0x09	Not supported
0x0A	Access level is determined by 'S' bit in %psr when MMU is enabled and *mmuen* generic is set to 1 User Access when MMU is enabled and *mmuen* generic is set to 2 Sets HPROT to user data regardless of MMU
0x0B	Access level is determined by 'S' bit in %psr when MMU is enabled and *mmuen* generic is set to 1 Supervisor Access when MMU is enabled and *mmuen* generic is set to 2, otherwise normal cache access Sets HPROT to supervisor data regardless of MMU
0x0C	Instruction cache tags
0x0D	Instruction cache data
0x0E	Data cache tags
0x0F	Data cache data
0x0F	Register file diagnostic parity read-out (FT only)
0x10	Flush instruction cache (and also data cache when system is implemented with MMU)
0x11	Flush data cache
0x13	MMU only: Flush instruction and data cache
0x14	MMU only: MMU diagnostic D context cache access (deprecated, do not use)
0x15	MMU only: MMU diagnostic I cache context access (deprecated, do not use)
0x18, 0x03	MMU only: Flush TLB and I/D cache
0x19, 0x04	MMU only: MMU registers
0x1C	MMU and cache bypass
0x1D	MMU only: MMU diagnostic access (deprecated, do not use)
0x1E	MMU only: MMU snoop tags diagnostic access

102.10.2ASI 0x1, Forced cache miss

ASI 1 is used for systems without cache coherency, to load data that may have changed in the background, for example by DMA units. It can also be used for other reasons, for example diagnostic purposes, to force a AHB load from memory regardless of cache state.

The address mapping of this ASI is matched with the regular address space, and if MMU is enabled then the address will be translated normally. Stores to this ASI will perform the same way as ordinary data stores.

For situations where you want to guarantee that the cache is not modified by the access, the MMU and cache bypass ASI, 0x1C, can be used instead.

102.10.3ASI 0x2, System control registers

ASI 2 contains a few control registers that have not been assigned as ancillary state registers. These should only be read and written using 32-bit LDA/STA instructions.

All cache registers are accessed through load/store operations to the alternate address space (LDA/STA), using ASI = 2. The table below shows the register addresses:

Table 1965. ASI 2 (system registers) address map

Address	Register
0x00	Cache control register
0x04	Reserved
0x08	Instruction cache configuration register
0x0C	Data cache configuration register

102.10.4 ASI 0x8-0xB, Data/Instruction

These ASIs are assigned by the SPARC manual for normal data and instruction fetches.

Accessing the instruction ASIs (0x8,0x9) explicitly via LDA/STA instructions is not supported in the LEON3 implementation.

Using LDA/STA with the user/supervisor data ASI (0xA,0xB) will behave as the affect the HPROT signal emitted by the processor according to section 102.7.1. If *mmuen* generic is set to 2, MMU access control will be done according to the indicated user or supervisor ASI inline with reference MMU description in Sparc V8 manual. If *mmuen* generic is set to 1, MMU access control will be done depending on the SU bit in the %psr register.

102.10.5 ASI 0xC-0xF, ICache tags/data, DCache tags/data

Note that on LEON3FT, ASI 0xF is double mapped and the description in this section is valid only if the RFT bit is not set.

ASI 0xC-0xF provide diagnostic access to the instruction cache memories. These ASIs should only be accessed by 32-bit LDA/STA instructions. These ASIs can not be used while a cache flush is in progress.

The same address bits used normally as index are used to index the cache also in the diagnostic access. For a multi-way cache, the lowest bits above the index part, the lowest bits that would normally be used as tag, are used to select which way to read/write. The remaining address bits are don’t cares, leading the address map to wrap around.

If fault tolerance is enabled, the tag parity, context and SO bits can also be read out through these ASIs by setting the PS bit in the cache configuration register. When this bit is set, the parity data is read instead of the ordinary data. When writing the tag bits, the context bits will always be written with the current context in the MMU control register. The SO bit in the tag will be written with the SO bit value in the MMU control register (SO bit in MMU control register only exists in the tag when *mmuen* generic is set to 2). The parity to be written is calculated based on the supply write-value, the context ID and optionally SO bit (if mmuen is set to 2) in the MMU control register. The parity bits can be modified via the TB field in the cache control register.

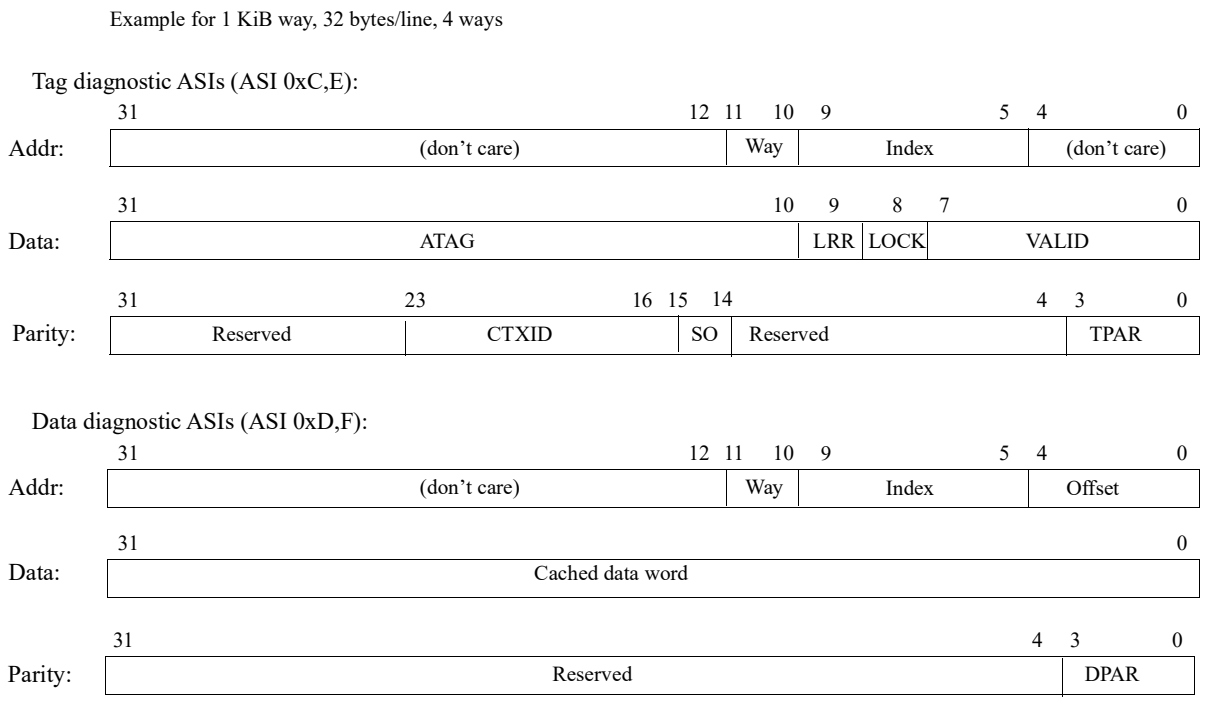


Figure 261. ASI 0xC-0xF address mapping and data layout

Field Definitions:

Address Tag (ATAG) - Contains the tag address of the cache line.

LRR - Used by LRR algorithm to store replacement history, otherwise 0.

LOCK - Locks a cache line when set. 0 if cache locking not implemented.

Valid (V) - When set, the corresponding 32-bit sub-block of the cache line contains valid data. V[0] corresponds to address 0 in the cache line, V[1] to address 1, V[2] to address 2 and so on. The D-Cache only has one valid bit per cache line which is replicated for the whole 8-bit diagnostic field.

CTXID - Context ID, used when MMU is enabled

SO - (Supervisor only access) It is used when MMU is enabled and *mmuen* (see Sec. 102.16) generic is set to 2. This bit is set to 1 when the associated page with the tag has access permission of 6 or 7 (supervisor access only) for other access permissions it is set to 0. (For detailed information about access permissions refer to the reference MMU part of the Sparc V8 manual)

TPAR - Byte-wise parity of tag bits, context ID parity is XOR:ed into bit 3.

DPAR - Byte-wise parity of data bits

NOTE: only the necessary bits will be implemented in the cache tag, depending on the cache configuration. As an example, a 4 KiB cache with 16 bytes per line would only have four valid bits and 20 tag bits. The cache rams are sized automatically by the RAM generators in GRLIB.

102.10.6ASI 0xF (alternate), FT register file parity read-out

In the FT version, an alternate function to ASI 0xF is selected by enabling the RFT bit in the cache control register. When enabled, this will override (shadow) the D-Cache data ASI function.

This ASI allows you to read out parity bits for the register file. The parity should be read out using a half-word LDUHA instruction. The parity bits are read out simultaneously for the two read ports. This is read only, for write access (fault injection) see section 102.9.5.

The ASI is not address-mapped, instead the registers used in the LDUHA instruction directly select which register's parity bits are returned. For example the instruction:

```
lduha   [%l0 + %l1] 0x0F, %g1
```

will read out the parity bit of %l0 on port 2 and %l1 on port 1, and store those values in register %g1.

The layout is shown in the figure below:

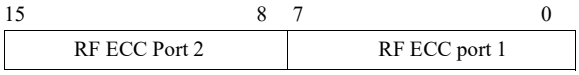


Figure 262. Register file ECC read-out layout

When the checkbits are read out using LDUHA, bit 29 (RFT) in the cache control register should be set to 1. The desired register should be used as address.

102.10.7ASI 0x10, 0x11, 0x13, 0x18 - Flush

For historical reasons there are multiple ASIs that flush the cache in different ways.

Writing to ASI 0x10 will flush the entire instruction cache. If MMU is implemented in the core, both instruction and data cache will be flushed.

Writing to ASI 0x11 will flush the data cache only.

Writing to ASI 0x13 will flush the data cache and the instruction cache, only available when MMU is implemented.

Writing to ASI 0x18 (and 0x03), which is available only if MMU is implemented, will flush both the MMU TLB, the I-cache, and the D-cache. This will block execution for a few cycles while the TLB is flushed and then continue asynchronously with the cache flushes continuing in the background.

102.10.8ASI 0x19 and 0x04 - MMU registers

This ASI provides access to the MMU:s control and status registers. The following MMU registers are implemented:

Table 1966. MMU registers (ASI = 0x19)

Address	Register
0x000	MMU control register
0x100	Context pointer register
0x200	Context register
0x300	Fault status register
0x400	Fault address register

102.10.9ASI 0x1C - MMU and cache bypass

Performing an access via ASI 0x1C will act as if MMU and cache were disabled. The address will not be translated and the cache will not be used or updated by the access.

102.10.10ASI 0x1E - MMU snoop/physical tags diagnostic access

If the MMU has been configured to use separate snoop (physical) tags, they can be accessed via ASI 0x1E. This is primarily useful for RAM testing, and should not be performed during normal operation. This ASI is addressed the same way as the regular diagnostic ASI:s 0xC, 0xE, and the read/written data has the layout as shown below (example for a 1 KiB/way data cache):

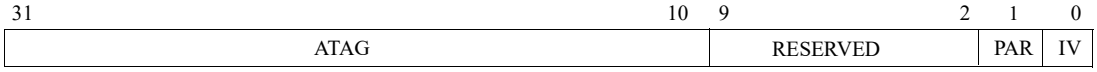


Figure 263. Snoop cache tag layout

[31:10] Address tag. The physical address tag of the cache line.
[1]: Parity. The odd parity over the data tag. LEON3FT only.
[0]: Invalid. When set, the cache line is not valid and will cause a cache miss if accessed by the processor. Only present if fast snooping is enabled.

102.11 Configuration registers

102.11.1PSR, WIM, TBR registers

The %psr, %wim, %tbr registers are implemented as required by the SPARC V8 manual.

*Table 1967.*LEON3 Processor state register (%psr)

31 28	27 24	23 20	19 16
IMPL	VER	ICC	RESERVED
0xF	0x3	0	0
r	r	r	r

15	14	13	12	11 8	7	6	5	4 0
AW	PAW	EC	EF	PIL	S	PS	ET	CWP
0	0	0	0	0	1	1	0	0
rw’	rw*	r	rw*	rw	rw	rw	rw	rw

31:28 Implementation ID (IMPL), read-only hardwired to “1111” (15)
27:24 Implementation version (VER), read-only hardwired to “0011” (3) for LEON3.
23:20 Integer condition codes (ICC), see sparcv8 for details
19:16 Reserved
15 Alternative window pointer select (AW), read-only if AWP not implemented
14 Previous alternative window pointer select (PAW), read-only if AWP not implemented
13 Enable coprocessor (EC), read-only if coprocessor not implemented
12 Enable floating-point (EF), read-only if FPU not implemented.
11:8 Processor interrupt level (PIL) - controls the lowest IRQ number that can generate a trap
7 Supervisor (S)
6 Previous supervisor (PS), see sparcv8 for details
5 Enable traps (ET)
4:0 Current window pointer

*Table 1968.*LEON3 Window invalid mask (%wim)

31 NWIN	NWIN-1 0
RESERVED	WIM
0	NR
r	rw

*Table 1969.*LEON3 Trap base address register (%tbr)

31 12	11 4	3 0
TBA	TT	R
*	0	0
rw	r	r

31:12 Trap base address (TBA) - Top 20 bits used for trap table address

11:4 Trap type (TT) - Last taken trap type. Read only.

3:0 Always zero, read only

102.11.2ASR17, LEON3 configuration register

The ancillary state register 17 (%asr17) provides information on how various configuration options were set during synthesis. This can be used to enhance the performance of software, or to support enumeration in multi-processor systems. There are also a few bits that are writable to configure certain aspects of the processor.

*Table 1970.*LEON3 configuration register (%asr17)

31 28	27	26	25	24 23	22 21	18	17	16
INDEX	DBP	NOTAG	DBPM	REXV	REXM	RESERVED	CS	CF[1]
*	0	1	1	*	*		*	*
r	rw*	r	rw*	r	rw*	r	r	r

15	14	13	12	11 10	9	8	7 5	4 0
CF[0]	DW	SV	LD	FPU	M	V8	NWP	NWIN
*	0	0	*	*	*	*	*	*
r	rw	rw*	r	r	r	r	r	r

31:28 Processor index (INDEX) - In multi-processor systems, each LEON core gets a unique index to support enumeration. The value in this field is identical to the *hindex* VHDL generic parameter in the VHDL model.

27 Disable Branch Prediction (DBP) - Disables branch prediction when set to '1'. Field is only available if the VHDL generic *bp* is set to the value 2.

26 Tagged arithmetic (NOTAG) - If this read-only field is '1' then the processor supports tagged arithmetic and the compare-and-swap (CASA) instruction. The current version if the LEON3 always supports tagged arithmetic and CASA.

25 Disable Branch Prediction on instruction cache misses (DBPM) - When set to '1' this avoids instruction cache fetches (and possible MMU table walk) for predicted instructions that may be annulled. This feature is on by default (reset value '1'), if branch prediction is programmable then this is also programmable.

24:23 REX version (REXV) - read-only field that is set to '00' if REX is not implemented, '01' if REX is implemented, '10' and '11' values are reserved for future implementations

22:21 REX mode (REXM) - set to '00' for REX enabled, '01' for REX illegal and '10' for REX transparent mode. Writable with reset value '01' when REX support has been enabled

20:18 Reserved for future implementations

17 Clock switching enabled (CS). If set, switching between AHB and CPU frequency is available.

16:15 CPU clock frequency (CF). CPU core runs at (CF+1) times AHB frequency.

14 Disable write error trap (DWT). When set, a write error trap (tt = 0x2b) will be ignored. Set to zero after reset.

13 Single-vector trapping (SVT) enable. If set, will enable single-vector trapping. Fixed to zero if SVT is not implemented. Set to zero after reset.

12 Load delay (LDDEL) - If set, the pipeline uses a 2-cycle load delay. Otherwise, a 1-cycle load delay i s used. Generated from the *lddel* VHDL generic parameter in the VHDL model.

11:10 FPU option. "00" = no FPU; "01" = GRFPU; "10" = Meiko FPU, "11" = GRFPU-Lite

9 If set, the optional multiply-accumulate (MAC) instruction is available

8 If set, the SPARC V8 multiply and divide instructions are available

7:5 Number of implemented watchpoints (NWP) (0 - 4)

4:0 Number of implemented registers windows corresponds to NWIN+1.

102.11.3ASR20, Alternative window register

This register allows access to the alternative window pointer. It is only implemented if the AWP support has been enabled in the processor.

*Table 1971.*LEON3 alternative window register (%asr20)

31	26	25	21	20	16
RESERVED		STWIN		CWPMAX	
0		0		*	
r		rw*		rw*	

15	5	4	0
RESERVED	WCWP	AWP	
0	-	*	
r	w	rw	

31:26	Reserved for future implementations
25:21	Starting window (STWIN) - Starting window of partition.
20:16	Maximum value of current window pointer (CWPMAX) - Partition size minus 1. Reset value is number of windows minus 1, which with STWIN=0 maps whole register file into partition. If this field is written with value 0, STWIN and CWPMAX fields are unmodified.
15:5	Reserved for future implementations
5	Write CWP - If written with 1, then the CWP field in PSR will simultaneously be written with the value written to AWP.
4:0	Alternative Window Pointer (AWP). Continuously updated with the value of CWP when the alternative window feature is disabled.

102.11.4ASR22-23 - Up-counter

The ancillary state registers 22 and 23 (%asr22-23) contain an internal up-counter that can be read by software without causing any access on the on-chip AMBA bus. The number of available bits in the counter is implementation dependent and is decided by the number of counter bits in the DSU time tag counter. %ASR23 contains the least significant part of the counter value and %ASR22 contains the most significant part. In case the implementation does not contain a debug support unit connected to the processor then the up-counter is not available (value is always zero).

The time tag value accessible in these registers is the same time tag value used for the system's trace buffers (if implemented) and for all processors connected to the same debug support unit. The time tag counter will increment when any of the trace buffers is enabled, or when the time tag counter is forced to be enabled via the DSU register interface, or when any processor has its %ASR22 Disable Up-counter (DUCNT) field set to zero.

The up-counter value will increment even if all processors have entered power-down mode.

Table 1972. LEON3 up-counter MSbs (%ASR22)

31	30 … 0
DUCNT	UPCNT(62:32)

31	Disable Up-counter (DUCNT) - Disable upcounter. When set to '1' the up-counter may be disabled. When cleared, the counter will increment each processor clock cycle. Default (reset) value is '1'.
30:0	Counter value (UPCNT(62:32)) - Most significant bits of internal up-counter. Read-only.

Table 1973. LEON3 up-counter LSbs (%ASR23)

31 … 0
UPCNT(31:0)

31:0	Counter value (UPCNT(31:0)) - Least significant bits of internal up-counter. Read-only.

102.11.5ASR24-31, Hardware watchpoint/breakpoint registers

Each breakpoint consists of a pair of ancillary state registers (%asr24/25, %asr26/27, %asr28/29 and %asr30/31) registers; one with the break address and one with a mask:

%asr24, %asr26
%asr28, %asr30

31 … 2	1	0
WADDR[31:2]		IF
NR	0	0
rw	r	rw

%asr25, %asr27
%asr29, %asr31

31 … 2		0
WMASK[31:2]	DL	DS
NR	0	0
rw	r	rw

Figure 264. Watch-point registers

WADDR - Address to compare against

WMASK - Bit mask controlling which bits to check (1) or ignore (0) for match

IF - break on instruction fetch from the specified address/mask combination

DL - break on data load from the specified address/mask combination

DS - break on data store to the specified address/mask combination

Note: Setting IF=DL=DS=0 disables the breakpoint

When there is a hardware watchpoint match and DL or DS is set then trap 0x0B will be generated. Hardware watchpoints can be used with or without the LEON3 debug support unit (DSU) enabled.

102.11.6Cache control register

The cache control register located at ASI 0x2, offset 0, contains control and status registers for the I and D cache.

*Table 1974.*LEON3 Cache Control Register (CCR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED		RFT	PS	TB				DS	FD	FI	FT		RES	ST	IB
0		0	0	0				0	0	0	*		0	*	0
r		rw*	rw*	rw*				rw	rw*	rw*	r		r	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IP	DP	ITE		IDE		DTE		DDE		DF	IF	DCS		ICS	
0	0	0		0		0		0		0	0	0		0	
r	r	r*		r*		r*		r*		rw	rw	rw		rw	

31:30	Reserved for future implementations
29	Register file test select (RFT). If set, will allow the read-out of IU register file checkbits via ASI 0x0F. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
28	Parity Select (PS) - if set diagnostic read will return 4 check bits in the lsb bits, otherwise tag or data word is returned. Only available if fault-tolerance is enabled (FT field in this register is non-zero). When the technology specific FT scheme is implemented (FT field is “10”) then PS selects if correctable or uncorrectable errors should generate cache misses. The recommended setting is to use PS=’0’ that treats uncorrectable RAM errors in the same way as parity errors would traditionally be handled.
27:24	Test Bits (TB) - if set, check bits will be xored with test bits TB during diagnostic write. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
23	Data cache snoop enable (DS) - if set, will enable data cache snooping.
22	Flush data cache (FD). If set, will flush the data cache. Always reads as zero.
21	Flush Instruction cache (FI). If set, will flush the instruction cache. Always reads as zero.
20:19	FT scheme (FT) - “00” = no FT, “01” = 4-bit checking implemented, “10” - Technology-specific protection implemented
18	Reserved for future implementations
17	Separate snoop tags (ST). This read-only bit is set if separate physical/snoop tags are implemented.
16	Instruction burst fetch (IB). This bit enables burst fill during instruction fetch.
15	Instruction cache flush pending (IP). This bit is set when an instruction cache flush operation is in progress
14	Data cache flush pending (DP). This bit is set when an data cache flush operation is in progress.
13:12	Instruction Tag Errors (ITE) - Number of detected parity errors in the instruction tag cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
11:10	Instruction Data Errors (IDE) - Number of detected parity errors in the instruction data cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
9:8	Data Tag Errors (DTE) - Number of detected parity errors in the data tag cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
7:6	Data Data Errors (DDE) - Number of detected parity errors in the data data cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
5	Data Cache Freeze on Interrupt (DF) - If set, the data cache will automatically be frozen when an asynchronous interrupt is taken.
4	Instruction Cache Freeze on Interrupt (IF) - If set, the instruction cache will automatically be frozen when an asynchronous interrupt is taken.
3:2	Data Cache state (DCS) - Indicates the current data cache state according to the following: X0= disabled, 01 = frozen, 11 = enabled.
1:0	Instruction Cache state (ICS) - Indicates the current data cache state according to the following: X0= disabled, 01 = frozen, 11 = enabled.

102.11.7I-cache and D-cache configuration registers

The configuration of the two caches if defined in two registers: the instruction and data configuration registers. These registers are read-only and indicate the size and configuration of the caches. They are located under ASI 2 at offset 8 and 12.

*Table 1975.*LEON3 Cache configuration register

31	30 28	27	26 24	23 20	19	18 16
CL	REPL	SN	WAYS	WSIZE	LR	LSIZE
*	*	*	*	*	*	*
r	r	r	r	r	r	r

15 12	11 4	3	2	1 0
LRSIZE	LRSTART	M	SO	RESERVED
*	*	*	*	0
r	r	r	r	r

31	Cache locking (CL). Set if cache locking is implemented.
30:28	Cache replacement policy (REPL). 00 - no replacement policy (direct-mapped cache), 01 - least recently used (LRU), 10 - least recently replaced (LRR), 11 - random
27	Cache snooping (SN). Set if snooping is implemented.
26:24	Cache associativity (WAYS). Number of ways in the cache: 000 - direct mapped, 001 - 2-way associative, 010 - 3-way associative, 011 - 4-way associative
23:20	Way size (WSIZE). Indicates the size (Kibs) of each cache way. Size = 2^{SIZE}
19	Local ram (LR). Set if local scratch pad ram is implemented.
18:16	Line size (LSIZE). Indicated the size (words) of each cache line. Line size = 2^{LSZ}
15:12	Local ram size (LRSZ). Indicates the size (Kibs) of the implemented local scratch pad ram. Local ram size = 2^{LRSZ}
11:4	Local ram start address (LRSTART). Indicates the 8 most significant bits of the local ram start address.
3	MMU present (M). This bit is set to ‘1’ if an MMU is present.
2	SO (supervisor only access) bit is present. This bit is set if *mmuen* generic is set to 2.
1:0	Reserved for future implementations

102.11.8ASR16, Register protection control register (FT only)

ASR register 16 (%asr16) is used to control the IU/FPU register file SEU protection. It is possible to disable the SEU protection by setting the IDI/FDI bits, and to inject errors using the ITE/FTE bits. Corrected errors in the register file are counted, and available in ICNT and FCNT fields. The counters saturate at their maximum value (7), and should be reset by software after read-out.

*Table 1976.*LEON3FT Register protection control register (%asr16)

31 30	29 27	26	25	24 22	21 20	19 18	17	16
FPFT	FCNT	SCREN	SCRDM	RESERVED	EFPFT	EIUFT	FTE	FDI
*	0	0	0	0	*	*	0	0
r	rw	rw	rw	r	r	r	rw	r

15 14	13 11	10 3	2	1	0
IUFT	ICNT	RFTB[7:0]	DP	ITE	IDI
*	0	0	0	0	0
r	rw	rw	rw	rw	rw

31:30	FP FT ID - Defines which SEU protection is implemented in the FPU (see table 1963)
29:27	FP RF error counter - Number of detected parity errors in the FP register file.
26	Integer register file scrubber enable
25	Integer register file scrubber data mismatch
24:22	Reserved for future implementations
21:20	Extended IU FT ID - Top bits of IUFT field to indicate FT values higher than 3
19:18	Extended IU FT ID - Top bits of IUFT field to indicate FT values higher than 3
17	FPU RF Test Enable - Enables FPU register file test mode. Parity bits are xored with TB before written to the FPU register file.
16	FP RF protection disable (FDI) - Disables FP RF parity protection when set.
15:14	IU FT ID - Defines which SEU protection is implemented in the IU (see table 1963)
13:11	IU RF error counter - Number of detected parity errors in the IU register file.
10:3	RF Test bits (RFTB) - In test mode, these bits are xored with correct parity bits before written to the register file.
2	DP ram select (DP) - Only applicable if the IU or FPU register files consists of two dual-port rams. See table 1977 below.
1	IU RF Test Enable - Enables register file test mode. Parity bits are xored with TB before written to the register file.
0	IU RF protection disable (IDI) - Disables IU RF parity protection when set.

*Table 1977.*DP ram select usage

ITE/FTE	DP	Function
1	0	Write to IU register (%i, %l, %o, %g) will only write location of %rs2 Write to FPU register (%f) will only write location of %rs2
1	1	Write to IU register (%i, %l, %o, %g) will only write location of %rs1 Write to FPU register (%f) will only write location of %rs1
0	X	IU and FPU registers written nominally

102.11.9MMU control register

The MMU control register is located in ASI 0x19 offset 0, and the layout can be seen in table 1978.

*Table 1978.*LEON3 MMU control register

31 28	27 24	23 21	20 18	17 16
IMPL	VER	ITLB	DTLB	PSZ
0	1	*	*	0
r	r	r	r	rw*

15	14	13	2	1	0
TD	ST	SO	RESERVED	NF	E
NR	0	0	0	0	0
rw*	r	rw	r	rw	rw

31:28	MMU Implementation ID. Hardcoded to “0000”
27:24	MMU Version ID. Hardcoded to “0001”.
23:21	Number of ITLB entries. The number of ITLB entries is calculated as 2^{ITLB}. If the TLB is shared between instructions and data, this field indicates to total number of TLBs.
20:18	Number of DTLB entries. The number of DTLB entries is calculated as 2^{DTLB}. If the TLB is shared between instructions and data, this field is zero.
17:16	Page size. The size of the smallest MMU page. 0 = 4 Kib; 1 = 8 Kib; 2 = 16 Kib; 3 = 32 Kib. If the page size is programmable, this field is writable, otherwise it is read-only.
15	TLB disable. When set to 1, the TLB will be disabled and each data access will generate an MMU page table walk. See Section 102.12.3 for detailed information.
14	Separate TLB. This bit is set to 1 if separate instruction and data TLBs are implemented
13	This bit only exists if *mmuen* generic is set to 2. This bit is written to the SO (supervisor only access) part of the TAG during diagnostic writes.
12:2	Reserved for future implementations
1	No Fault. When NF= 0, any fault detected by the MMU causes FSR and FAR to be updated and causes a fault to be generated to the processor. When NF= 1, a fault on an access to ASI 9 is handled as when NF= 0; a fault on an access to any other ASI causes FSR and FAR to be updated but no fault is generated to the processor.
0	Enable MMU. 0 = MMU disabled, 1 = MMU enabled.

102.11.10MMU context pointer and context registers

The MMU context pointer register is located in ASI 0x19 offset 0x100 and the MMU context register is located in ASI 0x19 offset 0x200. They together determine the location of the root page table descriptor for the current context. Their definition follows the SRMMU specification in the SPARC V8 manual with layouts shown below..

Table 1979. LEON3 MMU context pointer register

31 2	1 0
CONTEXT TABLE POINTER	R
NR	0
rw	r

31:2	Context table pointer, physical address bits 35:6 (note address is shifted 4 bits)
1:0	Reserved, always 0

Table 1980. LEON3 MMU context register

31 8	7 0
RESERVED	CONTEXT
0	0
r	rw

Table 1980. LEON3 MMU context register

31:8	Reserved
7:0	Current context ID

In the LEON3, the context bits are OR:ed with the lower MMU context pointer bits when calculating the address, so one can use less context bits to reduce the size/alignment requirements for the context table.

102.11.11MMU fault status register

The MMU fault status register is located in ASI 0x19 offset 0x300, and the definition follows the SRMMU specification in the SPARC V8 manual. The SPARC V8 specifies that the fault status register should be cleared on read, on the LEON3 only the FAV bit is cleared on read. The FAV bit is always set on error in the LEON3 implementation, so it can be used as a valid bit for the other fields..

*Table 1981.*LEON3 MMU fault status register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED														EBE	
0														0	
r														r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EBE						L		AT			FT			FAV	OW
0						0		0			0			0	0
r						r		r			r			r	r

31:18	Reserved
17:10	External bus error (EBE) - Never set on the LEON3
9:8	Level (L) - Level of page table entry causing the fault
7:5	Access type (AT) - See V8 manual
4:2	Fault type (FT) - See table 1958
1	Fault address valid (FAV) - Cleared on read, always written to 1 on fault
0	Overwrite (W) - Multiple faults of the same priority encountered

102.11.12MMU fault address register

The MMU fault address register is located in ASI 0x19 offset 0x400, and the definition follows the SRMMU specification in the SPARC V8 manual...

*Table 1982.*LEON3 MMU fault address register

31	12	11	0
FAULT ADDRESS		RESERVED	
NR		0	
r		r	

31:12	Top bits of virtual address causing translation fault
11:0	Reserved, always 0

102.12 Software considerations

102.12.1Register file initialization on power up (for LEON3FT)

After power-on, the check bits in the IU and FPU register files are not initialized. This means that access to an un-initialized (un-written) general-purpose register could cause a register access trap (tt = 0x20). Such behavior is considered as a software error, as the software should not read a register before it has been written. It is recommended that the boot code for the processor writes all registers in

the IU and FPU register files before launching the main application. This can be done also for the regular LEON3 to allow software to be portable to both FT and non-FT versions.

102.12.2Start-up

After reset, the caches are disabled and the cache control register (CCR) is 0. Before the caches may be enabled, a flush operation must be performed to initialized (clear) the tags and valid bits. A suitable assembly sequence could be:

```
flush
set 0x81000f, %g1
sta %g1, [%g0] 2
```

102.12.3MMU & TLB

After reset, the MMU is disabled and TLB is configured to be on (will not have any effect until MMU is enabled). Hence with default reset values the TLB has to be flushed before the MMU is being activated to initialize the valid bits in TLB. If the TLB is disabled while the MMU is active, the TLB must be flushed before enabled again.

102.12.4Data scrubbing (LEON3FT)

There is generally no need to perform data scrubbing on either IU/FPU register files or the cache memory. During normal operation, the active part of the IU/FPU register files will be flushed to memory on each task switch. This will cause all registers to be checked and corrected if necessary. Since most real-time operating systems performs several task switches per second, the data in the register files will be frequently refreshed.

The similar situation arises for the cache memory. In most applications, the cache memory is significantly smaller than the full application image, and the cache contents is gradually replaced as part of normal operation. For very small programs, the only risk of error build-up is if a part of the application is resident in the cache but not executed for a long period of time. In such cases, executing a cache flush instruction periodically (e.g. once per minute) is sufficient to refresh the cache contents.

102.13 LEON3 versions

102.13.1Overview

The primary way to identify the version of a implemented LEON3 processor is to look at the GRLIB build ID, plug&play device identifier and plug&play core revision (part of plug&play information, see GRLIB User's Manual for additional information). This documentation applies to version 3 of the LEON3 processor. Figure 265 shows the relationship between the different earlier LEON3 versions and the current LEON3v3.

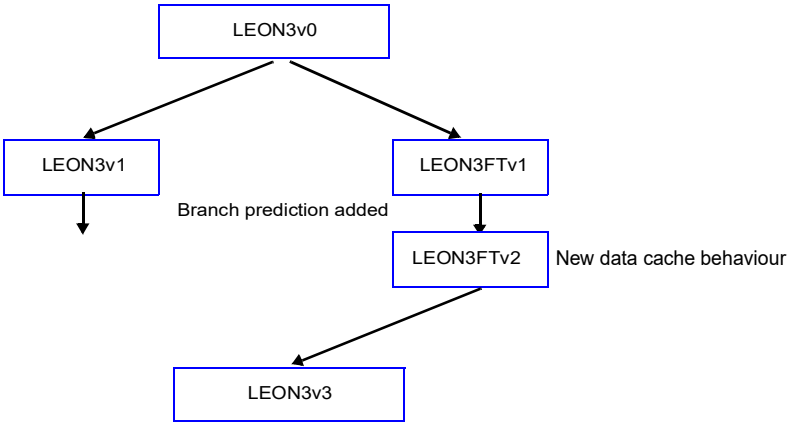


Figure 265. LEON3 processor evolution

102.13.2Data cache and AMBA behavior changes

Earlier versions of the LEON3 processor made use of one separate word access for each accessed word, even with cache enabled. This manual describes LEON3v3 which is the only LEON3 version included by default in the GRLIB IP Library. The table below also show the behaviour of earlier versions of the processor.

Processor operation	**Area not cacheable[1]**	**LEON3FTv1/LEON3v1**		**LEON3FTv2/LEON3v3**	
		Area is cacheable[1]		**Area is cacheable[1]**	
		Cache enabled[2]	**Cache disabled**	**Cache enabled[2]**	**Cache disabled**
Data load <= 32-bit	*Read access with size specified by load instruction*	*Word access*	*Read access with size specified by load instruction*	Burst of 32-bit accesses to fetch full cache line.	Read access with size specified by load instruction
Data load 64-bit (LDD)	*Burst of two 32-bit accesses*				Burst of two 32-bit accesses
Data store <= 32-bit	Store access with size specified by store instruction.				
Data store 64-bit (STD)	Burst of two 32-bit store accesses				

[1] Cachability is determined by the *cached* VHDL generic, if *cached* is zero then cachability is determined via AMBA PnP.

[2] Bus accesses for reads will only be made on L1 cache miss or forced cache miss.

102.13.3ASI 0x1 side effects

Reading from ASI 0x1 forces a real AMBA bus read and returns that data to the pipeline, and this behavior is implemented in all LEON3 and LEON3FT versions. However the side effects on the cache state (whether the cache is updated, and whether a line might be allocated) differs between different versions of the LEON3. Customers needing more information on this issue may contact Frontgrade Gaisler.

102.13.4ASR writes from user mode

Ancillary state registers ASR17 and ASR19-31 were made privileged on write, starting from LEON3v3. To preserve backward compatibility, these registers are still readable from user mode.

102.13.5MMU alias handling

LEON3v3 and LEON3FTv2 can handle double-mapped virtual addresses via self-snooping as described in section 102.3.8. This behavior is not available in earlier editions of LEON3, there the aliases can remain with incorrect data indefinitely.

102.13.6CASA and load delay 2

With the current LEON3 (starting with GRLIB build 4178) CASA is always supported. LEON3 versions before GRLIB build 4161 only supported the compare-and-swap (CASA) instruction when the processor was implemented with load delay 1 (*lddel* VHDL generic set to 1) while the current version supports CASA for all load delays.

For GRLB versions more recent than build 4161, software can check if CASA is supported by reading %asr17 and checking the NOTAG field. For earlier GRLIB versions (prior to GRLIB build 4161) the %asr17 LDDEL field must be set to 0 for CASA to be supported and software can then probe for CASA support by trying to execute the instruction and handle the illegal instruction trap that is generated if CASA is unsupported by the implementation.

102.14 Vendor and device identifiers

The core will have one of two device identifiers depending on if the processor has been implemented with or without fault-tolerance features.

The standard core has vendor identifiers 0x01 (Frontgrade Gaisler) and device identifier 0x003.

If the core has been implemented with fault-tolerance features then the core will be identified with vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x053.

For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

102.15 Implementation

102.15.1Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

102.15.2Technology mapping

LEON3 has two technology mapping VHDL generics, *fabtech* and *memtech*. The *fabtech* generic controls the implementation of some pipeline features, while *memtech* selects which memory blocks will be used to implement cache memories and the IU/FPU register file. *fabtech* can be set to any of the provided technologies (0 - NTECH) as defined in the TECHMAP.GENCOMP package. See the GRLIB Users's Manual for available settings for *memtech*.

102.15.3RAM usage

The LEON3 core maps all usage of RAM memory on the *syncram*, *syncram_2p* and *syncram_dp* components from the technology mapping library (TECHMAP). The type, configuration and number of RAM blocks is described below.

Register file

The register file is implemented with two *synram_2p* blocks for all technologies where the *regfile_3p_infer* constant in TECHMAP.GENCOMP is set to 0. The organization of the syncram_2p is shown in the following table:

*Table 1983.*syncram_2p sizes for LEON3 register file

Register windows	Syncram_2p organization
2 - 3	64x32
4 - 7	128x32
8 - 15	256x32
16-31	512x31
32	1024x32

If *regfile_3p_infer* is set to 1, the synthesis tool will automatically infer the register. On FPGA technologies, it can be in either flip-flops or RAM cells, depending on the tool and technology. On ASIC technologies, it will be flip-flops. The amount of flip-flops inferred is equal to the number of registers:

Number of flip-flops = ((NWINDOWS *16) + 8) * 32

FP register file

If FPU support is enabled, the FP register file is implemented with four *synram_2p* blocks when the *regfile_3p_infer* constant in TECHMAP.GENCOMP is set to 0. The organization of the syncram_2p blocks is 16x32.

If *regfile_3p_infer* is set to 1, the synthesis tool will automatically infer the FP register file. For ASIC technologies the number of inferred flip-flops is equal to number of bits in the FP register file which is 32 * 32 = 1024.

Cache memories

RAM blocks are used to implement the cache tags and data memories. Depending on cache configuration, different types and sizes of RAM blocks are used.

The tag memory is implemented with one *syncram* per cache way when no snooping is enabled. The tag memory depth and width is calculated as follows:

Depth = (cache way size in bytes) / (cache line size in bytes)

Width = 32 - log2(cache way size in bytes) + (cache line size in bytes)/4 + lrr + lock

For a 2 Kib cache way with lrr replacement and 32 bytes/line, the tag RAM depth will be (2048/32) = 64. The width will be: 32 - log2(2048) + 32/4 + 1 = 32 - 11 + 8 + 1 = 28. The tag RAM organization will thus be 64x28 for the configuration. If the MMU is enabled, the tag memory width will increase with 8 to store the process context ID, and the above configuration will us a 64x36 RAM.

If simple (MMU-less) snooping is enabled, the data cache tag memory will instead of single-port RAM blocks be implemented with a dual-port RAMs (syncram_dp) of the same size.

If physical (MMU-compatible) snooping is enabled, the data cache tag memories will be implemented using two *syncram_2p* components (with one read-only and one write-only port) per way , one memory for virtual and one for physical tags. The size of the virtual tag block will be the same as when snooping is disabled. The physical tag block will have the same depth as above and the data width corresponds to the width of the tag: 32 - log2(way size). A 4 KiB data cache way will thus require a 32 - 12 = 20 bit wide RAM block for the physical tags.

Physical snooping can also be implemented with valid bits placed in flip flops, the tag memories are then implemented using single-port memories and with one bit less in width for the virtual tag.

The data part of the caches (storing instructions or data) is always 32 bit wide. The depth is equal to the way size in bytes, divided by 4. A cache way of 2 KiB will thus use *syncram* component with and organization of 512x32.

Instruction Trace buffer

The instruction trace buffer will use four identical RAM blocks (*syncram*) to implement the buffer memory. The syncrams will always be 32-bit wide. The depth will depend on the *tbuf* VHDL generic, which indicates the total size of trace buffer in KiBs. If *tbuf* = 1 (1 KiB), then four RAM blocks of 64x32 will be used. If *tbuf* = 2, then the RAM blocks will be 128x32 and so on.

Scratch pad RAM

If the instruction scratch pad RAM is enabled, a *syncram* block will be instantiated with a 32-bit data width. The depth of the RAM will correspond to the configured scratch pad size. An 8 KiB scratch pad will use a *syncram* with 2048x32 organization. The RAM block for the data scratch pad will be configured in the same way as the instruction scratch pad.

102.15.4Double clocking

LEON3 double clocking is described in the LEON/GRLIB Configuration and Development Guide.

102.15.5Clock gating

LEON3 clock gating is described in the LEON/GRLIB Configuration and Development Guide.

102.15.6Scan support

Scan test support using signals distributed via the AMBA records is included in the LEON3 as described in the GRLIB User's Manual. It can be enabled using the *scantest* VHDL generic.

102.16 Configuration options

Table 1984 shows the configuration options of the core (VHDL generics).

*Table 1984.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
fabtech	Target technology	0 - NTECH	0 (inferred)
memtech	Vendor library for regfile and cache RAMs. Bits 16, 17 and 18 of this generic can be used to for the MMU TLB data RAM, IU register file and FP register file to inferred technology: + 2**16: Force inferred technology for MMU TLB data RAM + 2**17: Force inferred technology for IU register file + 2**18: Force inferred technology for FP register file Adding the value (2**17 + 2**18) is equivalent to setting the grlib.gencomp.regfile_3p_infer(memtech) to 1 (used for some technologies to force the register file implementations to inferred).	0 - 16#FFFFFFFF#	0 (inferred)
nwindows	Number of SPARC register windows. Choose 8 windows to be compatible with Bare-C and RTEMS cross-compilers.	2 - 32	8
dsu	Enable Debug Support Unit interface	0 - 1	0

*Table 1984.*Configuration options

Generic	Function	Allowed range	Default
fpu	Floating-point Unit 0 : no FPU 1 - 7: GRFPU 1 - inferred multiplier, 2 - DW multiplier, 3 - Module Generator multiplier, 4 - Technology specific multiplier 8 - 14: GRFPU-Lite 8 - simple FPC, 9 - data forwarding FPC, 10 - non-blocking FPC 15: Obsolete 16 - 31: as above (modulo 16) but use netlist 32 - 63: as above (modulo 32) but uses shared GRFPU interface Netlist or technology specific multiplier is recommended when using Synplify to target Virtex-5.	0 - 63	0
v8	Generate SPARC V8 MUL and DIV instructions This generic is assigned with the value: *mult + 4*struct* Where *mult* selects between the following implementation options for the multiplier and divider: 0 : No multiplier or divider 1 : 16x16 multiplier 2 : 16x16 pipelined multiplier 16#32# : 32x32 pipelined multiplier Where *struct* selects the structure option for the integer multiplier. The following structures can be selected: 0: Inferred by synthesis tool 1: Generated using Module Generators from NTNU 2: Using technology specific netlists (techspec). Only supported for RTAX-D FPGAs. Other technologies will assert a simulation error. 3: Using Synopsys DesignWare (DW02_mult and DW_mult_pipe)	0 - 16#3F#	0
cp	Generate co-processor interface	0 -1	0
mac	Generate SPARC V8e SMAC/UMAC instruction. Can only be used together with a 16x16 multiplier.	0 - 1	0
pclow	Least significant bit of PC (Program Counter) that is actually generated. PC[1:0] are always zero and are normally not generated. Generating PC[1:0] is convenient for VHDL-debugging and these bits should be optimized out by the synthesis tool.	0, 2	2
notag	Unused. Was previously used to disable tagged arithmetic and CASA instructions	0 - 1	0
nwp	Number of hardware watchpoints	0 - 4	0
icen	Enable instruction cache	0 - 1	1
irepl	Instruction cache replacement policy. 0 - least recently used (LRU), 1 - least recently replaced (LRR), 2 - random	0 - 1	0
isets	Number of instruction cache ways. Note: Generic named isets due to historical reasons.	1 - 4	1
ilinesize	Instruction cache line size in number of words	4, 8	4
isetsize	Size of each instruction cache way in KiB	1 - 256	1
isetlock	Enable instruction cache line locking. Do not use if cmft > 0	0 - 1	0
dcen	Data cache enable	0 - 1	1

*Table 1984.*Configuration options

Generic	Function	Allowed range	Default
drepl	Data cache replacement policy. 0 - least recently used (LRU), 1 - least recently replaced (LRR), 2 - random	0 - 1	0
dsets	Number of data cache ways Note: Generic named dsets due to historical reasons.	1 - 4	1
dlinesize	Data cache line size in number of words	4, 8	4
dsetsize	Size of each data cache way in KiB Note: If the processor is implemented with the MMU then the cache way size needs to be equal or less then the MMU page size for hardware cache coherency.	1 - 256	1
dsetlock	Enable data cache line locking. Do not use if cmft > 0	0 - 1	0
dsnoop	Enable data cache snooping Bit 0-1: 0: disable, 1: obsolete, 2: enabled, 3: single-port physical Bit 2: 0: simple MMU-less snooping, 1: save extra physical tags (MMU snooping)	0 - 7	0
ilram	Enable local instruction RAM	0 - 1	0
ilramsize	Local instruction RAM size in kB	1 - 512	1
ilramstart	8 MSB bits used to decode local instruction RAM area	0 - 255	16#8E#
dlram	Enable local data RAM (scratch-pad RAM)	0 - 1	0
dlramsize	Local data RAM size in kB	1 - 512	1
dlramstart	8 MSB bits used to decode local data RAM area	0 - 255	16#8F#
mmuen	Enable memory management unit (MMU) Note: Bus snooping is required to avoid cache aliasing effects when the MMU is enabled if the cache has more than one way. 0 : MMU does not exist. 1 : MMU exists. 2 : MMU exists and the cache tags include an additional bit called SO (supervisor only access). See sec. 102.3.2 for more details. If MMU is going to be instantiated in the processor, it is recommended to set the *mmuen* generic to 2.	0 - 2	0
itlbnum	Number of instruction TLB entries	2 - 64	8
dtlbnum	Number of data TLB entries	2 - 64	8
tlb_type	0 : separate TLB with slow write 1: shared TLB with slow write 2: separate TLB with fast write	0 - 2	1
tlb_rep	LRU (0) or Random (1) TLB replacement	0 - 1	0
lddel	Load delay. One cycle gives best performance, but might create a critical path on targets with slow (data) cache memories. A 2-cycle delay can improve timing but will reduce performance with about 5%.	1 - 2	2
disas	Print instruction disassembly in VHDL simulator console. Has no effect on synthesis (code removed via pragma translate_off)	0 - 1	0
tbuf	Size of instruction trace buffer in kB (0 - instruction trace disabled). For values 1-64 a single-port trace buffer of size *tbuf* is used. For values 65-128 a two-port trace buffer of size *tbuf*-64 is used.	0 - 128	0

*Table 1984.*Configuration options

Generic	Function	Allowed range	Default
pwd	Power-down. 0 - disabled, 1 - area efficient, 2 - timing efficient.	0 - 2	1
svt	Enable single-vector trapping	0 - 1	0
rstaddr	Default reset start address. This generic sets the 20 most significant bits of the reset address. The reset address must always be aligned on a 4 KiB address boundary. If this generic is set to 16#fffff# the processor will read its start address from the interrupt controller interface signal IRQI.RSTVEC (dynamic reset start address). See section 102.2.20 for more information.	0 - (2**20-1)	0
smp	Enable multi-processor support in pipeline (usually set in template designs to number of CPUs minus 1)	0 - 15	0
iuft, fpft	Register file SEU protection. (0: no protection; 1 : 4-bit parity, 2 : 8-bit parity; 3 : 7-bit BCH, 4: TMR, 5: BCH on-the-fly, 6: Tech-specific) See section 102.9.4 for fpft value restrictions.	0 - 6	0
cmft	Enable cache memory SEU protection., bitfield Bit 7:4: Selects technology specific protection cache protection. 0 is parity protection, 5 is technology specific protection Bit 3:1: Unused Bit 0: If set, enable cache memory protection Technology specific protection is further documented in the GRLIB-FT User's Manual (grlib-ft.pdf).	0 - 255	0
iuinj ceinj	Enable random error injection during simulation. Used for simulation only, removed on synthesis using pragma translate_off. Only supported for FT version of LEON3FT.	0 - 3	0
cached	Fixed cacheability mask. Setting to nonzero overrides plug'n'play cacheability information.	0 - 16#FFFF#	0
clk2x	Double-clocking, frequency factor		0
netlist	Use netlist rather than RTL code (currently unused)	0 - 1	0
scantest	Enable scan test support	0 - 1	0
mmupgsz	MMU Page size. 0 = 4K, 1 = 8K, 2 = 16K, 3 = 32K, 4 = programmable.	0 - 4	0
bp	Enable branch prediction support, 0 - disabled, 1 - always enabled, 2 - programmable	0 - 2	1
npasi	Enable SPARC V8E nonprivileged ASI access. 0 - All accesses to alternate address space are privileged. 1 - LOAD and STORE from alternate space instructions accessing ASIs 0x00-0x7F are privileged, ASIs 0x80-0xFF are nonprivileged.	0 - 1	0
pwrpsr	Enable SPARC V8E partial write PSR (WRPSR).	0 - 1	0
rex	Enable LEON-REX extension	0 - 1	0
altwin	Enable alternative window pointer extension and register file partitioning	0 - 1	0

102.17 Signal descriptions

Table 1985 shows the interface signals of the core (VHDL ports). There are several top-level entities available for the LEON3 processor. The *leon3x* entity contains all signals and settings. The other entities are wrappers around *leon3x*. The available entities are:

- leon3cg - Top-level with support for clock gating. Deprecated, do not use for new designs.
- leon3ft2x - Top-level with support for FT, double clocking and clock gating.
- leon3ftsh - Entity with support for FT and shared FPU.
- leon3ft - Entity with support for FT and clock gating, no separate FPU clock.
- leon3s2x - Top-level with support for clock gating and double clocking, no separate FPU clock.
- leon3sh - Top-level with support for shared FPU.
- leon3s - Simplest top-level, no FT, clock gating or shared FPU.
- leon3x - Entity with support for all features (double clocking, FT, clock gating, shared FPU)

Table 1985. Signal descriptions

Signal name	Field	Type	Function	Active
leon3x:CLK leon3s2x:CLK leon3ftsh:CLK	N/A	Input	AMBA clock, used in 2x mode. Note that this only applies to the processor entities listed.	-
leon3x:GCLK2 leon3s:CLK leon3sh:CLK leon3s2x:GCLK2 leon3ft:GCLK leon3ftsh:GCLK leon3cg:GCLK leon3ft2x:GCLK2	N/A	Input	Processor clock, can be gated when using an entity where the clock name starts with “G”. Note that this clock has different names depending on which top-level entity that is used to instantiate the processor. For example, LEON3S only has one clock input which covers three of the rows in this table (Processor clock, FPU-clock, free running processor clock).	-
leon3x:GFCLK2 leon3s:CLK leon3sh:CLK leon3s2x:GCLK2 leon3ft:CLK leon3ftsh:CLK leon3cg:CLK leon3ft2x:GCLK2	N/A	Input	FPU clock, can be gated Note that this clock has different names depending on which top-level entity that is used to instantiate the processor. For example, LEON3S only has one clock input which covers three of the rows in this table (Processor clock, FPU-clock, free running processor clock).	-
leon3x:CLK2 leon3s:CLK leon3sh:CLK leon3s2x:GCLK2 leon3ft:CLK leon3ftsh:CLK leon3cg:CLK leon3ft2x:GCLK2	N/A	Input	Free running processor clock Note that this clock has different names depending on which top-level entity that is used to instantiate the processor. For example, LEON3S only has one clock input which covers three of the rows in this table (Processor clock, FPU-clock, free running processor clock).	-
RSTN	N/A	Input	Reset	Low
AHBI	*	Input	AHB master input signals	-

*Table 1985.*Signal descriptions

Signal name	Field	Type	Function	Active
AHBO	*	Output	AHB master output signals	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO[]	*	Input	AHB slave output signals from all slaves on same bus. The processor makes use of the plug&play sideband signals to decode cacheability information of the bus. This can be overridden by the cached VHDL generic.	-
IRQI	IRL[3:0]	Input	Interrupt level	High
	RESUME	Input	Clear power-down and error mode	High
	RSTRUN	Input	Start after reset (SMP system only)	High
	RSTVEC[31:12]	Input	Reset start addr. (SMP and dynamic reset addr.)	-
	INDEX[3:0]	Input	Unused	-
	PWDSETADDR	Input	In power-down/error mode, shift PC to nPC and set PWDNEWADDR to PC.	High
	PWDNEWADDR [31:2]	Input	New PC value used with PWDSETADDR	-
	FORCEERR	Input	Force CPU into error mode	High
IRQO	INTACK	Output	Interrupt acknowledge	High
	IRL[3:0]	Output	Processor interrupt level	High
	PWD	Output	Processor in power-down mode	High
	FPEN	Output	Floating-point unit enabled	High
	ERR	Output	Processor in error mode	High
DBGI	-	Input	Debug inputs from DSU	-
DBGO	-	Output	Debug outputs to DSU	-
	ERROR		Processor in error mode, execution halted	Low
CLKEN		Input	Clock enable/qualifier used in 2x mode	High

* see GRLIB IP Library User's Manual

102.18 Signal definitions and reset values

When the processor enters error mode, the *errorn* output is driven active.

The signals and their reset values are described in table 1986.

*Table 1986.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
errorn	Tri-state output	Processor error mode indicator	Low	Tri-state

102.19 Timing

The timing waveforms and timing parameters are shown in figure 266 and are defined in table 1987.

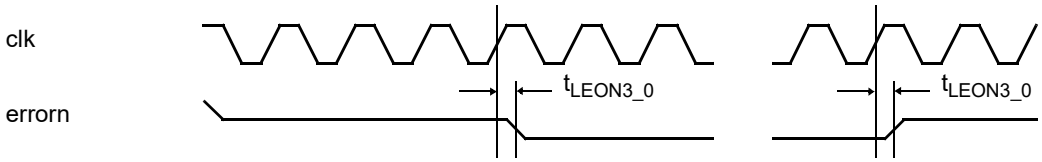


Figure 266. Timing waveforms

*Table 1987.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{LEON3_0}	clock to output delay	rising *clk* edge	TBD	TBD	ns

102.20 Library dependencies

Table 1988 shows the libraries used when instantiating the core (VHDL libraries).

*Table 1988.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	LEON3	Component, signals	LEON3 component declaration, interrupt and debug signals declaration

102.21 Component declaration

The core has the following component declaration.

```
entity leon3s is
  generic (
    hindex    : integer                := 0;
    fabtech   : integer range 0 to NTECH  := 0;
    memtech   : integer range 0 to NTECH  := 0;
    nwindows  : integer range 2 to 32 := 8;
    dsu       : integer range 0 to 1  := 0;
    fpu       : integer range 0 to 3  := 0;
    v8        : integer range 0 to 2  := 0;
    cp        : integer range 0 to 1  := 0;
    mac       : integer range 0 to 1  := 0;
    pclow     : integer range 0 to 2  := 2;
    notag     : integer range 0 to 1  := 0;
    nwp       : integer range 0 to 4  := 0;
    icen      : integer range 0 to 1  := 0;
    irepl     : integer range 0 to 2  := 2;
    isets     : integer range 1 to 4  := 1;
    ilinesize : integer range 4 to 8  := 4;
    isetsize  : integer range 1 to 256 := 1;
    isetlock  : integer range 0 to 1  := 0;
    dcen      : integer range 0 to 1  := 0;
    drepl     : integer range 0 to 2  := 2;
    dsets     : integer range 1 to 4  := 1;
    dlinesize : integer range 4 to 8  := 4;
    dsetsize  : integer range 1 to 256 := 1;
    dsetlock  : integer range 0 to 1  := 0;
    dsnoop    : integer range 0 to 6:= 0;
    ilram      : integer range 0 to 1 := 0;
    ilramsize  : integer range 1 to 512 := 1;
```

```
    ilramstart : integer range 0 to 255 := 16#8e#;
    dlram      : integer range 0 to 1 := 0;
    dlramsize  : integer range 1 to 512 := 1;
    dlramstart : integer range 0 to 255 := 16#8f#;
    mmuen      : integer range 0 to 2 := 0;
    itlbnum    : integer range 2 to 64 := 8;
    dtlbnum    : integer range 2 to 64 := 8;
    tlb_type   : integer range 0 to 1 := 1;
    tlb_rep    : integer range 0 to 1 := 0;
    lddel      : integer range 1 to 2  := 2;
    disas      : integer range 0 to 1  := 0;
    tbuf       : integer range 0 to 64 := 0;
    pwd        : integer range 0 to 2  := 2;     -- power-down
    svt        : integer range 0 to 1  := 1;     -- single vector trapping
    rstaddr    : integer               := 0;
    smp        : integer range 0 to 15 := 0;    -- support SMP systems
    cached     : integer               := 0;     -- cacheability table
    scantest   : integer               := 0
    mmupgsz    : integer range 0 to 5  := 0;
    bp         : integer               := 1
);

port (
    clk    : in  std_ulogic;
    rstn   : in  std_ulogic;
    ahbi   : in  ahb_mst_in_type;
    ahbo   : out ahb_mst_out_type;
    ahbsi  : in  ahb_slv_in_type;
    ahbso  : in  ahb_slv_out_vector;
    irqi   : in  l3_irq_in_type;
    irqo   : out l3_irq_out_type;
    dbgi   : in  l3_debug_in_type;
    dbgo   : out l3_debug_out_type
  );
end;
```

103 LEON4 - High-performance SPARC V8 32-bit Processor

103.1 Overview

LEON4 is a 32-bit processor core conforming to the IEEE-1754 (SPARC V8) architecture. It is designed for embedded applications, combining high performance with low complexity and low power consumption.

The LEON4 core has the following main features: 7-stage pipeline with Harvard architecture, separate instruction and data caches, hardware multiplier and divider, on-chip debug support and multiprocessor extensions.

The LEON4 processor can be enhanced with fault-tolerance against SEU errors. The fault-tolerance is focused on the protection of on-chip RAM blocks, which are used to implement IU/FPU register files and the cache memory. Configuring the processor to implement fault-tolerance enables additional internal register and register fields.

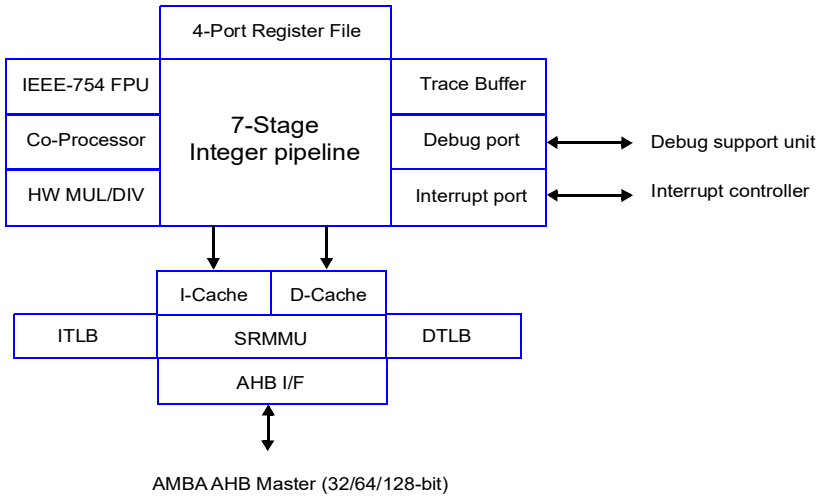


Figure 267. LEON4 processor core block diagram

Note: This manual describes the full functionality of the LEON4 core. Through the use of VHDL generics, parts of the described functionality can be suppressed or modified to generate a smaller or faster implementation.

Please also refer to the *LEON/GRLIB Configuration and Development Guide* for recommendations on system design and LEON4 configuration.

103.1.1 Integer unit

The LEON4 integer unit implements the full SPARC V8 manual, including hardware multiply and divide instructions. The number of register windows is configurable within the limit of the SPARC manual (2 - 32), with a default setting of 8. The pipeline consists of 7 stages with a separate instruction and data cache interface (Harvard architecture).

103.1.2 Cache sub-system

LEON4 has a highly configurable cache system, consisting of a separate instruction and data cache. Both caches can be configured with 1 - 4 ways, 1 - 256 KiB/way, 16 or 32 bytes per line. The instruction cache maintains one valid bit per cache line and uses streaming during line-refill to minimize refill latency. The data cache has one valid bit per cache line, uses write-through policy and imple-

ments a double-word write-buffer. Bus-snooping on the AHB bus can be used to maintain cache coherency for the data cache.

103.1.3 Floating-point unit and co-processor

The LEON4 integer unit provides interfaces for a floating-point unit (FPU), and a custom co-processor. Two FPU controllers are available, one for the high-performance GRFPU and one for the GRFPU-Lite core. The floating-point processors and co-processor execute in parallel with the integer unit, and does not block the operation unless a data or resource dependency exists. Note that the FPUs are provided separately.

103.1.4 Memory management unit

A SPARC V8 Reference Memory Management Unit (SRMMU) can optionally be enabled. The SRMMU implements the full SPARC V8 MMU specification, and provides mapping between multiple 32-bit virtual address spaces and physical memory. A three-level hardware table-walk is implemented, and the MMU can be configured to up to 64 fully associative TLB entries per implemented TLB.

103.1.5 On-chip debug support

The LEON4 pipeline includes functionality to allow non-intrusive debugging on target hardware. To aid software debugging, up to four watchpoint registers can be enabled. Each register can cause a breakpoint trap on an arbitrary instruction or data address range. When the (optional) debug support unit is attached, the watchpoints can be used to enter debug mode. Through a debug support interface, full access to all processor registers and caches is provided. The debug interfaces also allows single stepping, instruction tracing and hardware breakpoint/watchpoint control. An internal trace buffer can monitor and store executed instructions, which can later be read out via the debug interface.

103.1.6 Interrupt interface

LEON4 supports the SPARC V8 interrupt model with a total of 15 asynchronous interrupts. The interrupt interface provides functionality to both generate and acknowledge interrupts.

103.1.7 AMBA interface

The cache system implements an AMBA AHB master to load and store data to/from the caches. The interface is compliant with the AMBA-2.0 standard. During line refill, incremental burst are generated to optimise the data transfer. The AMBA interface can be configured to use a 64 or 128-bit bus on cache line fills. The processor also has a snoop AHB slave input port which is used to monitor the accesses made by other masters, if snooping has been enabled.

103.1.8 Power-down mode

The LEON4 processor core implements a power-down mode, which halts the pipeline and caches until the next interrupt. The processor supports optional clock gating during the power down period by providing a clock-enable signal that can be tied to an external clock gate cell, and by providing a separate clock input for the small part of the CPU that needs to run during power-down to check for wake-up conditions and maintain cache coherency.

103.1.9 Multi-processor support

LEON4 is designed to be used in multi-processor systems. Each processor has a unique index to allow processor enumeration. The write-through caches and snooping mechanism guarantees memory coherency in shared-memory systems.

103.2 LEON4 integer unit

103.2.1 Overview

The LEON4 integer unit implements the integer part of the SPARC V8 instruction set. The implementation is focused on high performance and low complexity. The LEON4 integer unit has the following main features:

- 7-stage instruction pipeline
- Separate instruction and data cache interface
- Support for 2 - 32 register windows
- Hardware multiplier with optional 16x16 bit MAC and 40-bit accumulator
- Radix-2 divider (non-restoring)
- Static branch prediction
- Single-vector trapping for reduced code size

Figure 268 shows a block diagram of the integer unit.

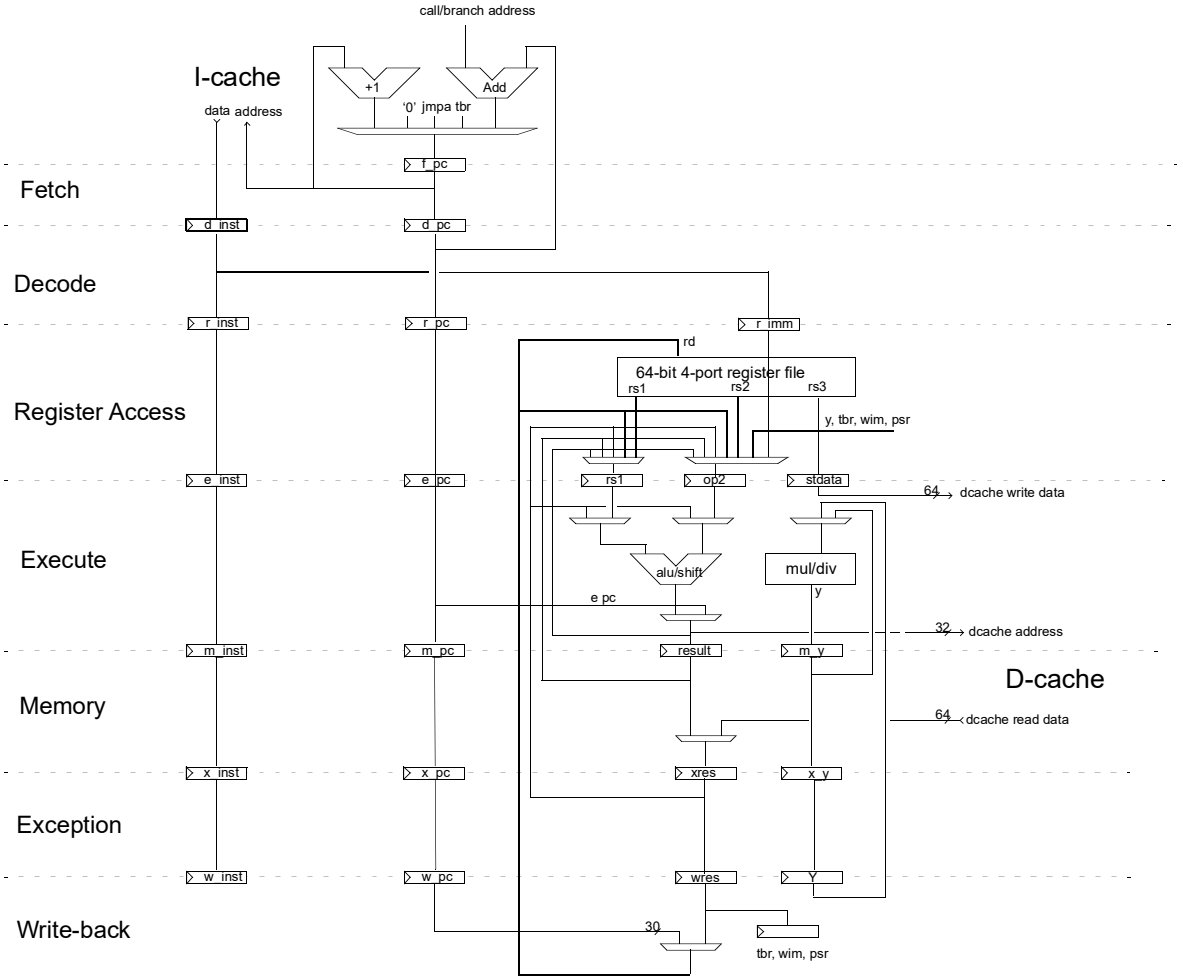


Figure 268. LEON4 integer unit datapath diagram

103.2.2 Instruction pipeline

The LEON4 integer unit uses a single instruction issue pipeline with 7 stages:

1. FE (Instruction Fetch): If the instruction cache is enabled, the instruction is fetched from the instruction cache. Otherwise, the fetch is forwarded to the memory controller. The instruction is valid at the end of this stage and is latched inside the IU.
2. DE (Decode): The instruction is decoded and the CALL/Branch target addresses are generated.
3. RA (Register access): Operands are read from the register file or from internal data bypasses.
4. EX (Execute): ALU, logical, and shift operations are performed. For memory operations (e.g., LD) and for JMPL/RETT, the address is generated.
5. ME (Memory): Data cache is accessed. Store data read out in the execution stage is written to the data cache at this time.
6. XC (Exception) Traps and interrupts are resolved. For cache reads, the data is aligned.
7. WR (Write): The result of ALU and cache operations are written back to the register file.

Table 1989 lists the cycles per instruction (assuming cache hit and no icc or load interlock):

*Table 1989.*Instruction timing

Instruction	Cycles (MMU disabled)
JMPL, RETT	3
SMUL/UMUL	1/4*
SDIV/UDIV	35
Taken Trap	5
Atomic load/store	5
All other instructions	**1**

* Multiplication cycle count is 1 clock (1 clock issue rate, 2 clock data latency), for the 32x32 multiplier and 4 clocks (issue rate, 4/5 clocks data latency for standard/pipelined version) for the 16x16 version.

Additional conditions that can extend an instructions duration in the pipeline are listed in the table and text below.

Branch interlock: When a conditional branch or trap is performed 1-2 cycles after an instruction which modifies the condition codes, 1-2 cycles of delay is added to allow the condition to be computed. If static branch prediction is enabled, this extra delay is incurred only if the branch is not taken.

Load delay: When using data resulting on a load shortly after the load, the instruction will be delayed to satisfy the pipeline's load delay. The processor pipeline has one cycle load delay.

Mul latency: For pipelined multiplier implementations there is 1 cycle extra data latency, accessing the result immediately after a MUL or MAC will then add one cycle pipeline delay.

Hold cycles: During cache miss processing or when blocking on the store buffer, the pipeline will be held still until the data is ready, effectively extending the execution time of the instruction causing the miss by the corresponding number of cycles. Note that since the whole pipeline is held still, hold cycles will not mask load delay or interlock delays. For instance on a load cache miss followed by a data-dependent instruction, both hold cycles and load delay will be incurred.

FPU/Coprocessor: The floating-point unit or coprocessor may need to hold the pipeline or extend a specific instruction. When this is done is specific to the FP/CP unit.

Certain specific events that cause these types of locks and their timing are listed in table 1990 below.

*Table 1990.*Event timing

Event	Cycles
Instruction cache miss processing, MMU disabled	3 + mem latency
Instruction cache miss processing, MMU enabled	5 + mem latency
Data cache miss processing, MMU disabled (read), L2 hit	3 + mem latency
Data cache miss processing, MMU disabled (write), write-buffer empty	0
Data cache miss processing, MMU enabled (read)	5 + mem latency
Data cache miss processing, MMU enabled (write), write-buffer empty	0
MMU page table walk	10 + 3 * mem latency
Branch prediction miss, branch follows ICC setting	2
Branch prediction miss, one instruction between branch and ICC setting	1
Pipeline restart due to register file or cache error correction	7

103.2.3 SPARC Implementor's ID

Frontgrade Gaisler is assigned number 15 (0xF) as SPARC implementor's identification. This value is hard-coded into bits 31:28 in the %psr register. The version number for LEON4 is 3 (same as for LEON3 to provide software compatibility), which is hard-coded in to bits 27:24 of the %psr.

103.2.4 Divide instructions

Full support for SPARC V8 divide instructions is provided (SDIV, UDIV, SDIVCC & UDIVCC). The divide instructions perform a 64-by-32 bit divide and produce a 32-bit result. Rounding and overflow detection is performed as defined in the SPARC V8 manual.

The divide instruction is required for full SPARC V8 compliance but can be configured out to save area using the *v8* VHDL generic.

103.2.5 Multiply instructions

The LEON processor supports the SPARC integer multiply instructions UMUL, SMUL UMULCC and SMULCC. These instructions perform a 32x32-bit integer multiply, producing a 64-bit result. SMUL and SMULCC performs signed multiply while UMUL and UMULCC performs unsigned multiply. UMULCC and SMULCC also set the condition codes to reflect the result. The multiply instructions are performed using a 32x32 pipelined hardware multiplier, or a 16x16 hardware multiplier which is iterated four times. To improve the timing, the 16x16 multiplier can optionally be provided with a pipeline stage.

The multiply instruction is required for full SPARC V8 compliance, but can be configured out to save area using the *v8* VHDL generic.

103.2.6 Multiply and accumulate instructions

To accelerate DSP algorithms, two multiply&accumulate instructions are implemented: UMAC and SMAC. The UMAC performs an unsigned 16-bit multiply, producing a 32-bit result, and adds the result to a 40-bit accumulator made up by the 8 lsb bits from the %y register and the %asr18 register. The least significant 32 bits are also written to the destination register. SMAC works similarly but performs signed multiply and accumulate. The MAC instructions execute in one clock but have two clocks latency, meaning that one pipeline stall cycle will be inserted if the following instruction uses the destination register of the MAC as a source operand.

The UMAC and SMAC instructions occupy the unused opcodes op=10,op3=1111110 for UMAC and op=10,op3=1111111 for SMAC.

Assembler syntax:

```
umac  rs1, reg_imm, rd
smac  rs1, reg_imm, rd
```

Operation:

```
prod[31:0] = rs1[15:0] * reg_imm[15:0]
result[39:0] = (Y[7:0] & %asr18[31:0]) + prod[31:0]
(Y[7:0] & %asr18[31:0]) = result[39:0]
rd = result[31:0]
```

%asr18 can be read and written using the RDASR and WRASR instructions.

The MAC instructions are an optional extension to SPARC V8, and enabled using the *mac* VHDL generic. The multiply and accumulate support also requires MUL/DIV support enabled by the *v8* VHDL generic and can only be used together with a 16x16 multiplier.

103.2.7 Compare and Swap instruction (CASA)

LEON4 implements the SPARC V9 Compare and Swap Alternative (CASA) instruction. The CASA operates as described in the SPARC V9 manual. The instruction is privileged, except when setting ASI = 0xA (user data).

103.2.8 Branch prediction

Static branch prediction can be optionally be enabled, and reduces the penalty for branches preceded by an instruction that modifies the integer condition codes. The predictor uses a branch-always strategy, and starts fetching instruction from the branch address. On a prediction hit, 1 or 2 clock cycles are saved, and there is no extra penalty incurred for misprediction as long as the branch target can be fetched from cache.

103.2.9 Register file data protection

The integer register file can optionally be protected against soft errors using triple modular redundancy (TMR) or technology specific protection (Technology specific protection is further documented in the GRLIB-FT User's Manual). Data errors will then be transparently corrected without impact at application level. The protection is enabled through the *ft* VHDL generic. The floating-point register file should be implemented with registers for applications that need FP register file protection.

103.2.10Hardware breakpoints

The integer unit can be configured to include up to four hardware breakpoints. Each breakpoint consists of a pair of ancillary state registers (see section 103.11.4). Any binary aligned address range can be watched for instruction or data access, and on a breakpoint hit, trap 0x0B is generated.

103.2.11Instruction trace buffer

The (optional) instruction trace buffer consists of a circular buffer that stores executed instructions. This is enabled and accessed only through the processor's debug port via the Debug Support Unit. When enabled, the following information is stored in real time, without affecting performance:

- Instruction address and opcode
- Instruction result
- Load/store data and address
- Trap information
- 30-bit time tag

The operation and control of the trace buffer is further described in section 33.4. Note that in multi-processor systems, each processor has its own trace buffer allowing simultaneous tracing of all instruction streams.

The size of the trace buffer is configurable from 1 to 64 KiB through the *tbuf* VHDL generic. If the value of tbuf is in the 65-128 range, a two-port instruction trace buffer of size *tbuf*-64 KiB will be used, allowing contextual reading of instructions while tracing is ongoing.

103.2.12Processor configuration register

The ancillary state register 17 (%asr17) provides information on how various configuration options were set during synthesis. This can be used to enhance the performance of software, or to support enumeration in multi-processor systems. See section 103.11.4 for layout.

103.2.13Exceptions

LEON4 adheres to the general SPARC trap model. The table below shows the implemented traps and their individual priority. When PSR (processor status register) bit ET=0, an exception trap causes the processor to halt execution and enter error mode, and the external error signal will then be asserted.

*Table 1991.*Trap allocation and priority

Trap	TT	Pri	Description	Class
reset	0x00	1	Power-on reset	Interrupting
data_store_error	0x2b	2	write buffer error during data store	Interrupting
instruction_access_exception	0x01	3	Error or MMU page fault during instruction fetch	Precise
privileged_instruction	0x03	4	Execution of privileged instruction in user mode	Precise
illegal_instruction	0x02	5	UNIMP or other un-implemented instruction	Precise
fp_disabled	0x04	6	FP instruction while FPU disabled	Precise
cp_disabled	0x24	6	CP instruction while Co-processor disabled	Precise
watchpoint_detected	0x0B	7	Hardware breakpoint match	Precise
window_overflow	0x05	8	SAVE into invalid window	Precise
window_underflow	0x06	8	RESTORE into invalid window	Precise
mem_address_not_aligned	0x07	10	Memory access to un-aligned address	Precise
fp_exception	0x08	11	FPU exception	Deferred
cp_exception	0x28	11	Co-processor exception	Deferred
data_access_exception	0x09	13	Access error during data load, MMU page fault	Precise
tag_overflow	0x0A	14	Tagged arithmetic overflow	Precise
division_by_zero	0x2A	15	Divide by zero	Precise
trap_instruction	0x80 - 0xFF	16	Software trap instruction (TA)	Precise
interrupt_level_15	0x1F	17	Asynchronous interrupt 15	Interrupting
interrupt_level_14	0x1E	18	Asynchronous interrupt 14	Interrupting
interrupt_level_13	0x1D	19	Asynchronous interrupt 13	Interrupting
interrupt_level_12	0x1C	20	Asynchronous interrupt 12	Interrupting
interrupt_level_11	0x1B	21	Asynchronous interrupt 11	Interrupting
interrupt_level_10	0x1A	22	Asynchronous interrupt 10	Interrupting
interrupt_level_9	0x19	23	Asynchronous interrupt 9	Interrupting
interrupt_level_8	0x18	24	Asynchronous interrupt 8	Interrupting
interrupt_level_7	0x17	25	Asynchronous interrupt 7	Interrupting
interrupt_level_6	0x16	26	Asynchronous interrupt 6	Interrupting
interrupt_level_5	0x15	27	Asynchronous interrupt 5	Interrupting
interrupt_level_4	0x14	28	Asynchronous interrupt 4	Interrupting
interrupt_level_3	0x13	29	Asynchronous interrupt 3	Interrupting
interrupt_level_2	0x12	30	Asynchronous interrupt 2	Interrupting
interrupt_level_1	0x11	31	Asynchronous interrupt 1	Interrupting

The prioritization follows the SPARC V8 manual.

The fp/cp_exception traps may be either deferred or precise depending on implementation, for the GRFPU and GRFPU-Lite implementations they are deferred. The data_store_error is delivered as a deferred exception but is non-resumable and therefore classed as interrupting in above table.

103.2.14Single vector trapping (SVT)

Single-vector trapping (SVT) is an SPARC V8e option to reduce code size for embedded applications. When enabled, any taken trap will always jump to the reset trap handler (%tbr.tba + 0). The trap type will be indicated in %tbr.tt, and must be decoded by the shared trap handler. SVT is enabled by setting bit 13 in %asr17. The model must also be configured with the VHDL generic *svt* = 1.

103.2.15Address space identifiers (ASI)

In addition to the address, a SPARC processor also generates an 8-bit address space identifier (ASI), providing up to 256 separate, 32-bit address spaces. During normal operation, the LEON4 processor accesses instructions and data using ASI 0x8 - 0xB as defined in the SPARC manual. Using the LDA/STA instructions, alternative address spaces can be accessed. The different available ASIs are described in section 103.10.

103.2.16Partial WRPSR

Partial write %PSR (WRPSR) is a SPARC V8e option that allows WRPSR instructions to only affect the %PSR.ET field. If the processor has been implemented with support for partial WRPSR and the WRPSR instruction's rd field is non-zero, then the WRPSR write will only update ET.

The model must be configured with the VHDL generic pwrpsr = 1 for partial WRPSR to be supported. Implementations that do not support partial WRPSR will write the full %PSR register regardless of the value of the WRPSR instruction's rd field.

103.2.17Power-down

The processor can be configured to include a power-down feature to minimize power consumption during idle periods. The power-down mode is entered by performing a WRASR instruction to %asr19:

```
wr %g0, %asr19
```

During power-down, the pipeline is halted until the next interrupt occurs. Signals inside the processor pipeline and caches are then static, reducing power consumption from dynamic switching.

Note: %asr19 must always be written with the data value zero to ensure compatiblity with future extensions.

Note: This instruction must be performed in supervisor mode with interrupts enabled.

When resuming from power-down, the pipeline will be re-filled from the point of power-down and the first instruction following the WRASR instruction will be executed prior to taking the interrupt trap. Up to six instructions after the WRASR instruction will be fetched (possibly with cache miss if they are not in cache) prior to fetching the trap handler.

103.2.18Processor reset operation

The processor is reset by asserting the RESET input for at least 4 clock cycles. The following table indicates the reset values of a subset of the registers which are affected by the reset..

*Table 1992.*Processor reset values

Register	Reset value
Trap Base Register	Trap Base Address field reset (value given by *rstaddr* VHDL generic)
PC (program counter)	0x0 (*rstaddr* VHDL generic)
nPC (next program counter)	0x4 (*rstaddr* VHDL generic + 4)
PSR (processor status register)	ET=0, S=1

By default, the execution will start from address 0. This can be overridden by setting the *rstaddr* VHDL generic in the model to a non-zero value. The reset address is always aligned on a 4 KiB boundary. If *rstaddr* is set to 16#FFFFF#, then the reset address is taken from the signal IRQI.RST-VEC. This allows the reset address to be changed dynamically.

103.2.19Multi-processor systems

In multiprocessor systems, the ID of the processor on which the code is executing can be read out by reading the index field of the LEON4 configuration register. Only processor 0 starts executing after reset, the others are in power-down mode and are activated by a signal from the interrupt controller.

103.2.20LEON-REX extension

The processor can be built with support for the LEON-REX addition to the SPARC instruction set, allowing a more compact code representation than the regular SPARC machine code. The details of the extension are given in a separate document. The extension is implemented when the *rex* VHDL generic is set.

Detection of whether support is present can be done by checking the REXV field in the asr17 register (see section 103.11.2). The REX support can be set to enabled, illegal or transparent mode via the REXEN/REXILL bits in the asr17 register, after reset the default setting is illegal so any LEON-REX code will cause an illegal instruction trap.

The extension is implemented as a decompressor internally inside the pipeline and does not affect the behavior of the caches, MMU or AHB bus interfaces.

When the *rex* generic is set, the instruction trace buffer entries are changed so the two most significant bits of the time tag are instead used to represent REX mode enabled status, and bit 1 of the program counter. The instructions opcodes logged into the trace buffer are the regular SPARC opcodes that are generated internally in the pipeline, not the LEON-REX opcodes that are in memory and cache.

103.3 Cache system

103.3.1 Overview

The LEON4 processor pipeline implements a Harvard architecture with separate instruction and data buses, connected to two separate cache controllers. As long as the execution does not cause a cache miss, the cache controllers can serve one beat of an instruction fetch and one data load/store per cycle, keeping the pipeline running at full speed. Each cache controller can be configured with different sizes and replacement policy.

On cache miss, the cache controller will assert a hold signal freezing the IU pipeline, and after delivering the data the hold signal is again lifted so execution continues. For accessing the bus, the cache controllers share the same AHB connection to the on-chip bus. Certain parts of the MMU (table walk logic, and depending on configuration also TLB buffer) are also shared between the two caches.

Another important component included in the data cache is the write buffer, allowing stores to proceed in parallel to executing instructions.

Cachability (memory areas that are cachable) for both caches is controlled through the AHB plug&play address information or using a VHDL generic, see section 103.7.2.

103.3.2 Cache operation

Each cache controller has two main memory blocks, the tag memory and the data memory. At each address in the tag memory, a number of cache entries, ways, are stored for a certain set of possible memory addresses. The data memory stores the data for the corresponding ways.

For each way, the tag memory contains the following information:

- Valid bits saying if the entry contains valid data or is free. Both caches have a single valid bit for each cache line.
- The tag, all bits of the cached memory address that are not implied by the set
- If MMU is enabled, the context ID of the cache entry
- If MMU is enabled with *mmuen* generic set to 2, SO bit (supervisor only access)
- If LRR is used, a bit specifying the replacement order
- If FT is enabled, check bits for detecting errors (depending on the fault-tolerance implementation selected the check bits may or may not be visible to the user)

When a read from cache is performed, the tags and data for all cache ways of the corresponding set are read out in parallel, the tags and valid bits are compared to the desired address and the matching way is selected. In the hit case, this is all done in the same cycle to support the full execution rate of the processor.

In the miss case, the cache will at first deliver incorrect data. However on the following cycle, a hold signal will be asserted to prevent the processor from proceeding with that data. After the miss has been processed, the correct data is injected into the pipeline using a memory data strobe (mds) signal, and afterwards the hold signal can be released. If the missed address is cacheable, then the data read in from the cache miss will be stored into the cache, possibly replacing one of the existing ways.

In the instruction streaming case, the processor pipeline is stepped one step for every received instruction. If the processor needs extra pipeline cycles to stretch a multi-cycle instruction or due to an interlock condition (see section 103.2), or if the processor jumps/branches away, then the instruction cache will hold the pipe, fetch the remainder of the cache line, and the pipeline will then proceed normally.

103.3.3 Cache configuration options

Each cache controller can be configured to implement a single-way (direct-mapped) cache or a multi-way cache with set associativity of 2 - 4. The way size is configurable to 1 - 256 KiB, divided into cache lines with 16 or 32 bytes of data.

In multi-way configurations, one of three replacement policies can be selected:

- Least-recently-used (LRU) - This maintains the order of usage for each set in the cache and replaces the one which has was used last. The LRU information needs to be updated on every cache hit and is therefore not stored with the tags but in separate flip flops. When the LRU option is enabled the replacement policy becomes dynamically configurable between LRU, Least-recently-replaced, pseudo-random and a direct-mapped mode.
- Least-recently-replaced (LRR) - This stores the index of the oldest replaced way along with the tags and uses this to select which way to replace. This policy can only be implemented when the number of ways is set to 2.
- Pseudo-random - This method samples a free-running counter to select which way to replace. System jitter (AMBA bus delay variations) will help to randomize the selected value.

Note that when using locking together with LRU and more than two ways, this will add extra lookup tables to determine which way to replace and this might become a critical path in the core.

103.3.4 Address mapping

The addresses seen by the CPU are divided into tag, index and offset bits. The index is used to select the set in the cache, therefore only a limited number of cache lines with the same index part can be stored at one time in the cache. The tag is stored in the cache and compared upon read.

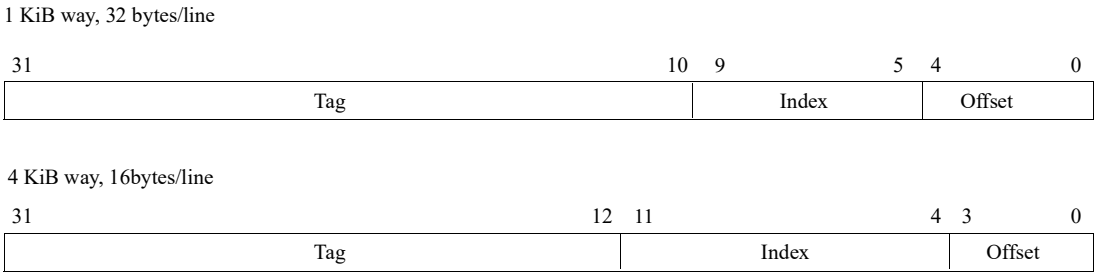


Figure 269. Cache address mapping examples

103.3.5 Data cache policy

The data cache employs a write-through policy, meaning that every store made on the CPU will propagate, via the write buffer, to the bus and there are no “dirty” lines in the cache that has not yet been written out apart from what is in the buffer. The store will also update the cache if the address is present, however a new line will not be allocated in that case.

*Table 1993.*LEON4 Data caching behavior

Operation	In cache	Cacheable	Bus action	Cache action	Load data
Data load	No	No	Read	No change	Bus
	No	Yes	Read	Line allocated/replaced	Bus
	Yes	-	None	No change	Cache
Data load with forced cache miss (ASI 1)	No	No	Read	No change	Bus
	No	Yes	Read	Line allocated/replaced	Bus
	Yes	-	Read	Data updated	Bus
Data load with MMU bypass (ASI 0x1C)	-	-	Read (phys addr)	No change	Bus
Data store	No	No	Write (via buffer)	No change	(N/A)
	No	Yes	Write (via buffer)	No change	(N/A)
	Yes	-	Write (via buffer)	Data updated	(N/A)
Data store with MMU bypass (ASI 0x1C)	-	-	Write (via buffer, phys addr)	No change	(N/A)

103.3.6 Write buffer

The data cache contains a write buffer able to hold a single 8,16,32, or 64-bit write. For half-word or byte stores, the stored data replicated into proper byte alignment for writing to a word-addressed device. The write is processed in the background so the system can keep executing while the write is being processed. However, any following instruction that requires bus access will block until the write buffer has been emptied. Loads served from cache will however not block, due to the cache policy used there can not be a mismatch between cache data and store buffer (the effect of this behavior on SMP systems is discussed in section 103.7).

Since the processor executes in parallel with the write buffer, a write error will not cause an exception to the store instruction. Depending on memory and cache activity, the write cycle may not occur until several clock cycles after the store instructions has completed. If a write error occurs, the currently

executing instruction will take trap 0x2b. This trap can be disabled using the DWT configuration (see section 103.11.2).

Note: a 0x2b trap handler should flush the data cache, since a write hit would update the cache while the memory would keep the old value due the write error

103.3.7 Operating with MMU

When MMU is enabled, the virtual addresses seen by the running code no longer correspond directly to the physical addresses on the AHB bus. The cache uses tags based on the virtual addresses, as this avoids having to do any additional work to translate the address in the most timing-critical hit case. However, any time a bus access needs to be made, a translation request has to be sent to the MMU to convert the virtual address to a physical address. For the write buffer, this work is included in the background processing of the store. The translation request to the MMU may result in memory accesses from the MMU to perform table walk, depending on the state of the MMU.

The MMU context ID is included in the cache tags in order to allow switching between multiple MMU contexts mapping the same virtual address to different physical addresses. Note that the cache does not detect aliases to the same physical address so in that case the same physical address may be cached in multiple ways (also see snooping below).

103.3.8 Snooping

The data cache can be configured to support AHB bus snooping. The AHB bus the processor is connected to, is monitored for writes from other masters to an address which is in the cache. If a write is done to a cached address, that cache line is marked invalid and the processor will be forced to fetch the (new) data from memory the next time it is read.

For using snooping together with the MMU, an extra tag memory storing physical tags must be added to allow comparing with the physical address on the AHB bus (called separate snoop tags or physical tags).

The processor can snoop on itself and invalidate any other cache lines aliased to the same physical address in case there are multiple virtual mappings to the same physical address that is being written. However, note that this does not happen until the write occurs on the bus so the other virtual aliases will return the old data in the meantime.

Snooping requires the way size of the cache to be equal or smaller than the MMU page size, otherwise the index into the physical and virtual tag RAM:s may not match, resulting in aliasing problems.

103.3.9 Enabling and disabling cache

Both I and D caches are disabled after reset. They are enabled by writing to the cache control register (see 103.11.5). Before enabling the caches after a reset they must be flushed to ensure that all tags are marked invalid.

103.3.10Cache freeze

Each cache can be in one of three modes: disabled, enabled and frozen. If disabled, no cache operation is performed and load and store requests are passed directly to the memory controller. If enabled, the cache operates as described above. In the frozen state, the cache is accessed and kept in sync with the main memory as if it was enabled, but no new lines are allocated on read misses.

If the DF or IF bit is set, the corresponding cache will be frozen when an asynchronous interrupt is taken. This can be beneficial in real-time system to allow a more accurate calculation of worst-case execution time for a code segment. The execution of the interrupt handler will not evict any cache lines and when control is returned to the interrupted task, the cache state is identical to what it was before the interrupt. If a cache has been frozen by an interrupt, it can only be enabled again by

enabling it in the CCR. This is typically done at the end of the interrupt handler before control is returned to the interrupted task.

103.3.11Flushing

Both instruction and data cache are flushed either by executing the FLUSH instruction, setting the FI/FD bits in the cache control register, or by writing to certain ASI address spaces.

Cache flushing takes one clock cycle per cache set, during which the IU will not be halted, but during which the caches are disabled. When the flush operation is completed, the cache will resume the state (disabled, enabled or frozen) indicated in the cache control register. Diagnostic access to the cache is not possible during a flush operation and will cause a data exception (trap=0x09) if attempted.

Note that while the SPARC V8 specifies only that the instructions pointed to by the FLUSH argument will be flushed, the LEON4 will additionally flush the entire I and D cache (which is permitted by the manual as the additional flushing only affects performance and not operation). While the LEON4 currently ignores the address argument, it is recommended for future compatibility to only use the basic flush %g0 form if you want the full flush behavior.

103.3.12Locking

Cache line locking is not supported by LEON4. The VHDL generics to enable this feature are present in the LEON4 component declaration, but are unused.

103.3.13Diagnostic access

The cache tag and data contents can be directly accessed for diagnostics and for locking purposes via various ASI:s, see section 103.10.5.

103.3.14Local scratch pad RAM

Local scratch pad RAM is not supported by LEON4. The VHDL generics to enable this feature are present in the LEON4 component declaration, but are unused.

103.3.15Fault tolerance support

The instruction and data cache can be protected a using mechanism implemented in the processor core (byte-parity codes) or by using functionality from SYNCRAMFT and SYNCRAM_2PFT. The most common option is to use parity protection that is provided by the processor. Use of the SYNCRAM protection allows the processor to use technology specific protection and this can lead to savings in resource utilization on target technologies that have built-in protection of SRAM blocks

On a detected error, the corresponding cache (I or D) will be flushed and the data will be refetched from external memory. This is done transparently to execution, and incur the same timing penalty as a regular cache miss. Enabling of the data protection is done through the *ft* VHDL generic and can only be implemented in GRLIB-FT releases.

103.4 Memory management unit

103.4.1 Overview

A memory-management unit can optionally be enabled. This is compatible with the SPARC V8 reference MMU (SRMMU) architecture described inthe SPARC V8 manual, appendix H.

The MMU provides address translation of both instructions and data via page tables stored in memory.When needed, the MMU will automatically access the page tables to calculate the correct physical address. The latest translations are stored in a special cache called the translation lookaside buffer (TLB), also referred to as Page Descriptor Cache (PDC) in the SRMMU specification. The MMU also

provides access control, making it possible to “sandbox” unpriviledged code from accessing the rest of the system.

103.4.2 MMU/Cache operation

When the MMU is disabled, the MMU is bypassed and the caches operate with physical address mapping. When the MMU is enabled, the cache tags store the virtual address and also include an 8-bit context field. Both the tag address and context field must match to generate a cache hit. When *mmuen* generic is set to 2 and MMU is enabled, the cache tags store a bit in addition to context called SO bit (supervisor only access). The SO bit is used to check the access permission of the data and instructions that resides in the level-1 caches when MMU is enabled. Without SO bit, access permissions of the load operations that hit in the data cache or the instruction accesses that hit in the instruction cache will not be checked properly.

If cache snooping is used, physical tags (separate snoop tags) must be enabled for it to work when address translation is used, see section 103.3.8.

Because the cache is virtually tagged, no extra clock cycles are needed in case of a cache load or instruction cache hit. In case of miss or write buffer processing, a translation is required which might add extra latency to the processing time, depending on TLB configuration and if there is a TLB miss.

The TLB can be configured in three different ways:

- Separate TLBs, slow access. TLB lookup adds 2 extra clock cycles.
- Shared TLB, slow access. TLB lookup adds 2 extra clock cycles, the TLB may be used by the other cache, leading to up to 4 extra cycles lookup time in the worst case.
- Separate TLBs, fast access. TLB lookup is done at the same time as tag lookup and therefore add no extra clock cycles.

If there is a TLB miss the page table must be traversed, resulting in up to four AMBA read accesses and one possible writeback operation. See the SRMMU specification for the exact format of the page table.

An MMU page fault will generate trap 0x09 for the D-cache and trap 0x01 for the I cache, and update the MMU status registers according to table 1994 and the SRMMU specification. In case of multiple errors, they fault type values are prioritized as the SRMMU specification requires. The cache and memory will not be modified on an MMU page fault.

*Table 1994.*LEON4 MMU Fault Status Register, fault type values

Fault type	SPARC V8 ref	Priority	Condition
6	Internal error	1	Never issued by LEON SRMMU
4	Translation error	2	AHB error response while performing table walk. Translations errors as defined in SPARC V8 manual. A translation error caused by an AMBA ERROR response will overwrite all other errors. Other translation errors do no overwrite existing translation errors when FAV = 1.
1	Invalid address error	3	Page table entry for address was marked invalid
3	Privilege violation error	4	Access denied based on page table and su status (see SRMMU spec for how privilege and protection error are prioritized)
2	Protection error	5	
0	None	-	No error (inside trap this means the trap occurred when fetching the actual data)

103.4.3 Translation look-aside buffer (TLB)

The MMU can be configured to use a shared TLB, or separate TLB for instructions and data. The number of TLB entries (for each implemented TLB) can be set to 2 - 64 via VHDL generics. The

organisation of the TLB and number of entries is not visible to the software and does thus not require any modification to the operating system. The TLB can be flushed using an STA instruction to ASI 0x18, see section 103.10.6.

103.4.4 Variable minimum page sizes

The standard minimum page size for the SRMMU is 4 KiB. The minimum page size can also be configured to 8, 16 or 32 KiB in order to allow for large data cache ways. The page size can either be configured hard at implementation time or made software-configurable via the MMU control register. The page sizes for level 1, 2 and 3 is seen in the table below:

*Table 1995.*MMU page size

Scheme	Level-1	Level-2	Level-3
4 KiB (default)	16 MiB	256 KiB	4 KiB
8 KiB	32 MiB	512 KiB	8 KiB
16 KiB	64 MiB	1 MiB	16 KiB
32 KiB	256 MiB	2 MiB	32 KiB

The layouts of the indexes are chosen so that PTE page tables can be joined together inside one MMU page without leaving holes.

Note that most operating systems are hard-coded for a specific page size and using one other than 4 KiB usually requires reconfiguration/recompilation of the operating system kernel.

103.5 Floating-point unit

The SPARC V8 architecture defines two (optional) co-processors: one floating-point unit (FPU) and one user-defined co-processor. Two different FPU's can be interfaced the LEON4 pipeline: Frontgrade Gaisler's GRFPU and GRFPU-Lite. Selection of which FPU to use is done through the VHDL model's VHDL generic map. The characteristics of the FPU's are described in the next sections.

103.5.1 Frontgrade Gaisler's floating-point unit (GRFPU)

The high-performance GRFPU operates on single- and double-precision operands, and implements all SPARC V8 FPU operations including square root and division. The FPU is interfaced to the LEON4 pipeline using a LEON4-specific FPU controller (GRFPC) that allows FPU instructions to be executed simultaneously with integer instructions. Only in case of a data or resource dependency is the integer pipeline held. The GRFPU is fully pipelined and allows the start of one instruction each clock cycle, with the exception is FDIV and FSQRT which can only be executed one at a time. The FDIV and FSQRT are however executed in a separate divide unit and do not block the FPU from performing all other operations in parallel.

All instructions except FDIV and FSQRT has a latency of three cycles, but to improve timing, the LEON4 FPU controller inserts an extra pipeline stage in the result forwarding path. This results in a

latency of four clock cycles at instruction level. The table below shows the GRFPU instruction timing when used together with GRFPC:

*Table 1996.*GRFPU instruction timing with GRFPC

Instruction	Throughput	Latency
FADDS, FADDD, FSUBS, FSUBD,FMULS, FMULD, FSMULD, FITOS, FITOD, FSTOI, FDTOI, FSTOD, FDTOS, FCMPS, FCMPD, FCMPES. FCMPED	1	4
FDIVS	14	16
FDIVD	15	17
FSQRTS	22	24
FSQRTD	23	25

The GRFPC controller implements the SPARC deferred trap model, and the FPU trap queue (FQ) can contain up to 7 queued instructions when an FPU exception is taken. When the GRFPU is enabled in the model, the version field in %fsr has the value of 2.

The GRFPU does not handle denormalized numbers as inputs and will in that case cause an fp_exception with the FPU trap type set to unfinised_FPOP (tt=2). There is a non-standard mode in the FPU that will instead replace the denormalized inputs with zero and thus never create this condition.

103.5.2 GRFPU-Lite

GRFPU-Lite is a smaller version of GRFPU, suitable for FPGA implementations with limited logic resources. The GRFPU-Lite is not pipelined and executes thus only one instruction at a time. To improve performance, the FPU controller (GRLFPC) allows GRFPU-Lite to execute in parallel with the processor pipeline as long as no new FPU instructions are pending. Below is a table of worst-case throughput of the GRFPU-Lite:

*Table 1997.*GRFPU-Lite worst-case instruction timing with GRLFPC

Instruction	Throughput	Latency
FADDS, FADDD, FSUBS, FSUBD,FMULS, FMULD, FSMULD, FITOS, FITOD, FSTOI, FDTOI, FSTOD, FDTOS, FCMPS, FCMPD, FCMPES. FCMPED	8	8
FDIVS	31	31
FDIVD	57	57
FSQRTS	46	46
FSQRTD	65	65

The GRLFPC controller implements the SPARC deferred trap model, but the FPU trap queue (FQ) can contain only one queued instructions when an FPU exception is taken.When the GRFPU-Lite is enabled in the model, the version field in %fsr has the value of 3.

103.6 Co-processor interface

No implementation for the user-defined co-processor is currently provided.

103.7 AMBA interface

103.7.1 Overview

The LEON4 processor has one AHB master interface. The types of AMBA accesses supported and performed by the processor depend on the accessed memory area's cachability, the maximum bus width, if the corresponding cache is enabled, and if the accessed memory area has been marked as being on the wide bus.

Cacheable instructions are fetched with a burst of 32-bit accesses, or 64- or 128-bit accesses depending on the cache line size and the AHB bus width.

The HPROT signals of the AHB bus are driven to indicate if the accesses is instruction or data, and if it is a user or supervisor access.

Table 1998. HPROT values

Type of access	User/Super	HPROT
Instruction	User	1100
Instruction	Super	1110
Data	User	1101
Data	Super	1111
MMU	Any	1101

In case of atomic accesses, a locked access will be made on the AMBA bus to guarantee atomicity as seen from other masters on the bus.

103.7.2 Cachability

Cachability for both caches can be controlled through the AHB plug&play address information or set manually via the *cached* VHDL generic.

For plug'n'play based cachability, the memory mapping for each AHB slave indicates whether the area is cachable, and this information is used to (statically) determine which access will be treated as cacheable. This approach means that the cachability mapping is always coherent with the current AHB configuration.

When the *cached* VHDL generic is not zero, it is treated as a 16-bit field, defining the cachability of each 256 MiB address block on the AMBA bus. For example, a value of 16#00F3# will define cachable areas in 0 - 0x20000000 and 0x40000000 - 0x80000000.

In order to access the plug'n'play information, the processor takes the ahbso vector as input. Only the static hconfig signals are used so the use of this input will be eliminated through constant propagation during synthesis.

103.7.3 AMBA access size

Cacheable data is fetched in a burst of 64- or 128-bit accesses, depending on the cache line size and AHB bus width. Data access to uncacheable areas may only be done with 8-, 16- and 32-bit accesses, i.e. the LDD and STD instructions may not be used.

If an area is marked as cacheable then the data cache will automatically try to use 64- or 128-bit accesses. This means that if 64- or 128-bit accesses need to be avoided, for example when performing Flash programming or if a slave does not support 64- or 128-bit accesses and is mapped as cacheable (this is a system design error), then software should only perform data accesses with using the cache bypass ASI and no 64-bit loads (LDD) when accessing the slave. One example of how to use cache bypass for loads is given by the following function:

```
static inline int load(int addr)
{
    int tmp;
    asm volatile(" lda [%1]0x1c, %0 "
        : "=r"(tmp)
        : "r"(addr)
        );
    return tmp;
}
```

The type of AMBA accesses used, and supported by the processor, for a memory area depends on the area's cachability and the values of the *wbmask* and *busw* VHDL generics.

The area which supports 64- or 128-bit access is indicated in the *wbmask* VHDL generic. This VHDL generic is treated as a 16-bit field, defining the 64/128-bit capability of each 256 MiB address block on the AMBA bus. A value of 16#00F3# will thus define areas in 0 - 0x20000000 and 0x40000000 - 0x80000000 to be 64/128-bit capable. The maximum access size to be used in the area(s) marked with WBMASK is determined by the *busw* VHDL generic.

Store instructions result in a AMBA access with size corresponding to the executed instruction, 64-bit store instructions (STD) are always translated to 64-bit accesses (never converted into two 32-bit stores as is done for LEON3). The table below indicates the access types used for instruction and data accesses depending on cachability, wide bus mask (wbmask), and cache configuration.

<table>
<tr><th rowspan="3">Processor operation</th><th colspan="3">Accessed memory area is 32-bit only, wbmask(address) = 0</th><th colspan="3">Accessed memory area is on wide bus wbmask(address) = 1</th></tr>
<tr><th rowspan="2">Area not cacheable[1]</th><th colspan="2">Area is cacheable[1]</th><th rowspan="2">Area not cacheable[1]</th><th colspan="2">Area is cacheable[1]</th></tr>
<tr><th>Cache enabled[2]</th><th>Cache disabled</th><th>Cache enabled[2]</th><th>Cache disabled</th></tr>
<tr><td>Instruction fetch</td><td>Burst of 32-bit read accesses</td><td colspan="2">Burst of 32-bit read accesses</td><td colspan="3">Burst of 64- or 128-bit accesses[5]</td></tr>
<tr><td>Data load <= 32-bit</td><td rowspan="3">Read access with size specified by load instruction</td><td rowspan="2">Illegal[3,6] Burst of 32-bit accesses, software may get incorrect data</td><td rowspan="2">Read access with size specified by load instruction</td><td rowspan="3">Read access with size specified by load instruction</td><td rowspan="2">Burst of 64- or 128-bit accesses[5]</td><td rowspan="2">Read access with size specified by load instruction</td></tr>
<tr><td>Data load <= 32-bit with ASI 0x01</td></tr>
<tr><td>Data load <= 32-bit with ASI 0x1C</td><td colspan="2">Read access with size specified by load instruction[6]</td><td colspan="2">Read access with size specified by load instruction[6]</td></tr>
<tr><td>Data load 64-bit (LDD)</td><td>Illegal[4] Single 64-bit access will be performed</td><td>Illegal[3] Burst of 64- or 128-bit accesses[5]</td><td>Illegal[3] Single 64-bit access will be performed</td><td>Illegal[4] Single 64-bit access will be performed</td><td>Burst of 64- or 128-bit accesses[5]</td><td>Single 64-bit read access</td></tr>
<tr><td>Data store <= 32-bit</td><td colspan="6">Store access with size specified by store instruction.</td></tr>
<tr><td>Data store 64-bit (STD)</td><td colspan="3">Illegal (64-bit store performed to 32-bit area) 64-bit store access will be performed.</td><td colspan="3">64-bit store access</td></tr>
</table>

Cells with red text show unsupported combinations of settings for wide bus and cacheability. The LEON4 core requires that all cacheable areas are can handle wide bus accesses (64- or 128-bit). Implementing the core with cacheable areas where wbmask is 0 is unsupported. The table lists the behaviour in these cases for completeness.

[1] Cachability is determined by the *cached* VHDL generic, if *cached* is zero then cachability is determined via AMBA PnP.

[2] Bus accesses for reads will only be made on L1 cache miss, load with forced cache miss or loads with cache bypass.

[3] LEON4 is designed to always make use of wide bus accesses for cacheable data. Cacheable data can only be handled with 64- or 128 bit accesses.

[4] Data accesses to uncachable areas may only be done with 8-, 16- and 32-bit accesses.

[5] 64- or 128-bit accesses depending on *busw* VHDL generic.

[6] Loads with cache bypass (ASI 0x1C) can be used to perform single accesses to cachable slaves.

103.7.4 Error handling

An AHB ERROR response received while fetching instructions will normally case an instruction access exception (tt=0x1). However if this occurs during streaming on an address that is not needed, the I cache controller will just not set the corresponding valid bit in the cache tag. If the IU later fetches an instruction from the failed address, a cache miss will occur, triggering a new access to the failed address.

An AHB ERROR response while fetching data into the data cache will normally trigger a data_access_exception trap (tt=0x9). If the error was for a part of the cache line other than what was currently being requested by the pipeline, a trap is not generated and the valid bit for that line is not set.

An ERROR response during an MMU table walk will lead the MMU to set the fault type to Internal error (1) and generate an instruction or data access exception, depending on which type of access that caused the table walk.

103.7.5 Snoop port

For the snooping logic, the LEON4 has an ahbsi input. For correct function, this must be tied to the same AHB bus that the master interface. It is not possible to snoop on another bus or to add extra pipeline registers to the snoop port, because the snoop logic must be in sync with the master interface.

103.8 Multi-processor system support

This section gives an overview of issues when using the LEON4 in multi-processor configuration.

Using the features described in earlier sections together with a multiprocessor capable IRQ controller (IRQMP, IRQ(A)MP), the LEON4 processor can support symmetric multiprocessing (SMP) configurations with shared memory, with up to 16 processors attached to the same AHB bus.

Enabling SMP features (sleeping on reset for CPU 1-N) is done by setting the *smp* VHDL generic to 1 or higher. Cache snooping should always be enabled in SMP systems to maintain data cache coherency between the processors.

103.8.1 Start-up

In multiprocessor systems, only the first processor will start after reset and all other processors will remain halted in power-down mode. After the system has been initialized, the remaining processors can be started by writing to the ‘multiprocessor status register’, located in the multiprocessor interrupt controller. The halted processors start executing from the reset address (0 or *rstaddr* VHDL generic, see section 103.2.18).

An application in a multiprocessor system can determine which processor it is executing on by checking the processor index field in the LEON4 configuration register (%asr17). As all processors typically have the same reset start address value, boot software must check the processor index and perform processor specific setup (e.g. initialization of stack pointer) based on the value of the processor index.

In recent versions of the LEON4, and if the IRQ controller is configured with the extended boot register extension, it is possible for one processor to monitor and reboot another processor.via the interrupt controller. This requires careful software design.

For earlier versions of the LEON4, this is not supported and if software detects that one processor is unresponsive and needs to restart the processor then the full system should be reset, for example by triggering the system’s watchdog, if implemented. In order for software to monitor that all processors in a system are up and running it is recommended to implement a heartbeat mechanism in software.

While it is possible to have more fine-grained control over processor behaviour via the Debug Support Unit (if implemented) this is not recommended as the debug support unit is typically disabled in production mode.

103.8.2 Shared memory model

Each processor core has it’s own separate AHB master interface and the AHB controller will arbitrate between them to share access to the on-chip bus.

If caches are not used, the processors will form a sequentially consistent (SC) system, where every processor will execute it’s loads, stores and atomics to memory in program order on the AHB bus and

the different processors operations will be interleaved in some order through the AHB arbitration. The shared memory controller AHB slave is assumed to not reorder accesses so a read always returns the latest written value to that location on the bus.

When using caches with snooping (and with separate physical tags, also called separate snoop tags, if using the MMU), the shared memory will act according to the slightly weaker SPARC Total Store Order (TSO) model. The TSO model is close to SC, except that loads may be reordered before stores coming from the same CPU. The stores and atomics are conceptually placed in a FIFO (see the diagrams in the SPARC manual) and the loads are allowed to bypass the FIFO if they are not to the same address as the stores. Loaded data from other addresses may therefore be either older or newer, with respect to the global memory order, than the stores that have been performed by the same CPU.

In the LEON4 case this happens because cache hits are served without blocking even when there is data in the write buffer. The loaded data will always return the stored data in case of reading the same address, because if it is cached, the store updates the cache before being put in the write buffer, and if it was not in cache then the load will result in a miss which waits for the write buffer to complete. Loaded data from a different address can be older than the store if it is served by cache before the write has completed, or newer if it results in a cache miss or if there is a long enough delay for the store to propagate to memory before reading.

See relevant literature on shared memory systems for more information. These details are mainly of concern for complex applications using lock-free data structures such as the Linux kernel, the recommendation for applications is to instead avoid concurrent access to shared structures by using mutexes/semaphores based on atomic instructions, or to use message passing schemes with one-directional circular buffers.

103.8.3 Memory-mapped hardware

Hardware resources (IP cores) are normally memory mapped on uncacheable address spaces. They will be accessible from all the CPU:s in a sequentially consistent manner. Since software drivers usually expect to be “alone” accessing the IP core and the IP cores register interfaces are not designed for concurrent use by multiple masters, using a bare-C application designed for single-processor usage on multiple cores at the same time will generally not work. This can be solved by partitioning the applications so that each IP core is only accessed by one of the CPU:s. This partitioning also need to be done between the interrupts so the IP core’s interrupts will be received by the correct processor.

103.9 Fault tolerance

103.9.1 Overview

The LEON4 processor can be enhanced with fault-tolerance against SEU errors (referred to as LEON4FT). The fault-tolerance is focused on the protection of on-chip RAM blocks, which are used to implement IU register file and the cache memory.

The LEON4 with FT features is licensed separately, and in the commercial LEON4 releases setting the FT VHDL generic will not have any effect.

103.9.2 Integer register file protection

The SEU protection for the integer unit register file can be implemented in two different ways, depending on target technology and available RAM blocks. The SEU protection scheme is selected

during synthesis, using the *ft* VHDL generic. Table 1999 below shows the implementation characteristics of the four possible SEU protection schemes.

*Table 1999.*Integer unit SEU protection schemes

ID	Implementation	Description	Usage
0	No protection (hardening at lower level)	No error checking, equivalent to non-FT version. Register file hardness must be ensured separately, for example by mapping the register file memories to SEU hardened flip-flops.	IU
4	Memory triplication	Memory blocks triplicated and bit by bit voted on outputs. Correction on-the-fly without pipeline restart, no error injection interface or error counters. Note that care must be taken by the implementer to ensure that the TMR is not collapsed in optimization by synthesis tools.	IU
6	Technology specific	Implement register files using native ECC capability of the technology (via syncram_2pft in the techmap library). Only valid for subset of (FPGA) technologies. Correction on-the-fly without pipeline restart. Error injection and error counters may be supported depending on technology. Technology specific protection is further documented in the GRLIB-FT User's Manual (grlib-ft.pdf).	IU

An uncorrectable error in the IU register file will cause trap 0x20 (*r_register_access_error*). A dedicated counter exists in ASR16 to count the number of register file corrections.

The register file is implemented using scheme 0 if the regfile_4p_infer array is set for the selected memory technology in the techmap library, or if bits 16-17 of the memtech VHDL generic are set.

103.9.3 Floating-point register file protection

The FPU register file is implemented with registers.

Note that the restrictions on protection scheme is not enforced, so it is recommended to simulate the configuration with error injection to ensure that the scheme chosen is functioning correctly. It is also recommended to confirm in the netlist that the expected register file type (memory block or flip flops) was implemented.

103.9.4 Cache protection

Each word in the tag or data memories is normally protected by four check bits. Use of the SYNCRAM protection allows the processor to use technology specific protection and this can lead to savings in resource utilization on target technologies that have built-in protection of SRAM blocks. If separate physical tags for snooping are enabled, the physical tag memory is also protected. An error during cache access will cause an invalidation of the cache, and a re-execution of the failing instruction. This will ensure that the complete cache line (tags and data) is refilled from external memory.

If snooping is enabled, an error detected in the tags while snooping a write to that set will lead to that cache data being invalidated (since the tag before the error might have matched the written address).

103.10 ASI assignments

103.10.1Summary

The table shows the ASI usage for LEON.

*Table 2000.*ASI usage

ASI	Usage
0x01	Forced cache miss.
0x02	System control registers (cache control register)
0x08, 0x09	Not supported
0x0A	Access level is determined by 'S' bit in %psr when MMU is enabled and *mmuen* generic is set to 1 User Access when MMU is enabled and *mmuen* generic is set to 2 Sets HPROT to user data regardless of MMU
0x0B	Access level is determined by 'S' bit in %psr when MMU is enabled and *mmuen* generic is set to 1 Supervisor Access when MMU is enabled and *mmuen* generic is set to 2, otherwise normal cache access Sets HPROT to supervisor data regardless of MMU
0x0C	Instruction cache tags
0x0D	Instruction cache data
0x0E	Data cache tags
0x0F	Data cache data
0x0F	Register file diagnostic parity read-out (FT only)
0x10	Flush instruction cache (and also data cache when system is implemented with MMU)
0x11	Flush data cache
0x13	MMU only: Flush instruction and data cache
0x14	MMU only: MMU diagnostic D context cache access (deprecated, do not use)
0x15	MMU only: MMU diagnostic I cache context access (deprecated, do not use)
0x18, 0x03	MMU only: Flush TLB and I/D cache
0x19, 0x04	MMU only: MMU registers
0x1C	MMU and cache bypass
0x1D	MMU only: MMU diagnostic access (deprecated, do not use)
0x1E	MMU only: MMU snoop tags diagnostic access

103.10.2ASI 0x1, Forced cache miss

ASI 1 is used for systems without cache coherency, to load data that may have changed in the background, for example by DMA units. It can also be used for other reasons, for example diagnostic purposes, to force a AHB load from memory regardless of cache state.

The address mapping of this ASI is matched with the regular address space, and if MMU is enabled then the address will be translated normally. Stores to this ASI will perform the same way as ordinary data stores.

For situations where you want to guarantee that the cache is not modified by the access, the MMU and cache bypass ASI, 0x1C, can be used instead.

103.10.3ASI 0x2, System control registers

ASI 2 contains a few control registers that have not been assigned as ancillary state registers. These should only be read and written using 32-bit LDA/STA instructions.

All cache registers are accessed through load/store operations to the alternate address space (LDA/STA), using ASI = 2. The table below shows the register addresses:

Table 2001. ASI 2 (system registers) address map

Address	Register
0x00	Cache control register
0x04	Reserved
0x08	Instruction cache configuration register
0x0C	Data cache configuration register

103.10.4 ASI 0x8-0xB, Data/Instruction

These ASIs are assigned by the SPARC manual for normal data and instruction fetches.

Accessing the instruction ASIs explicitly via LDA/STA instructions is not supported in the LEON4 implementation.

Using LDA/STA with the user/supervisor data ASI (0xA,0xB) will behave as the affect the HPROT signal emitted by the processor according to section 103.7.1. If *mmuen* generic is set to 2, MMU access control will be done according to the indicated user or supervisor ASI inline with reference MMU description in Sparc V8 manual. If *mmuen* generic is set to 1, MMU access control will be done depending on the SU bit in the %psr register.

103.10.5 ASI 0xC-0xF, ICache tags/data, DCache tags/data

ASI 0xC-0xF provide diagnostic access to the instruction cache memories. These ASIs should only be accessed by 32-bit LDA/STA instructions. These ASIs can not be used while a cache flush is in progress and will cause a data exception (trap=0x09) if attempted. Diagnostic accesses to the instruction cache can only be performed when the instruction cache is disabled and will cause a a data exception (trap=0x09) when the instruction cache is enabled.

The same address bits used normally as index are used to index the cache also in the diagnostic access. For a multi-way cache, the lowest bits above the index part, the lowest bits that would normally be used as tag, are used to select which way to read/write. The remaining address bits are don't cares, leading the address map to wrap around.

If fault tolerance is enabled, the tag parity, context and SO bits can also be read out through these ASIs by setting the PS bit in the cache configuration register. When this bit is set, the parity data is read instead of the ordinary data. When writing the tag bits, the context bits will always be written with the current context in the MMU control register. The SO bit in the tag will be written with the SO bit value in the MMU control register (SO bit in MMU control register only exists in the tag when *mmuen* generic is set to 2). The parity to be written is calculated based on the supply write-value, the context ID and optionally SO bit (if mmuen is set to 2) in the MMU control register. The parity bits can be modified via the TB field in the cache control register.

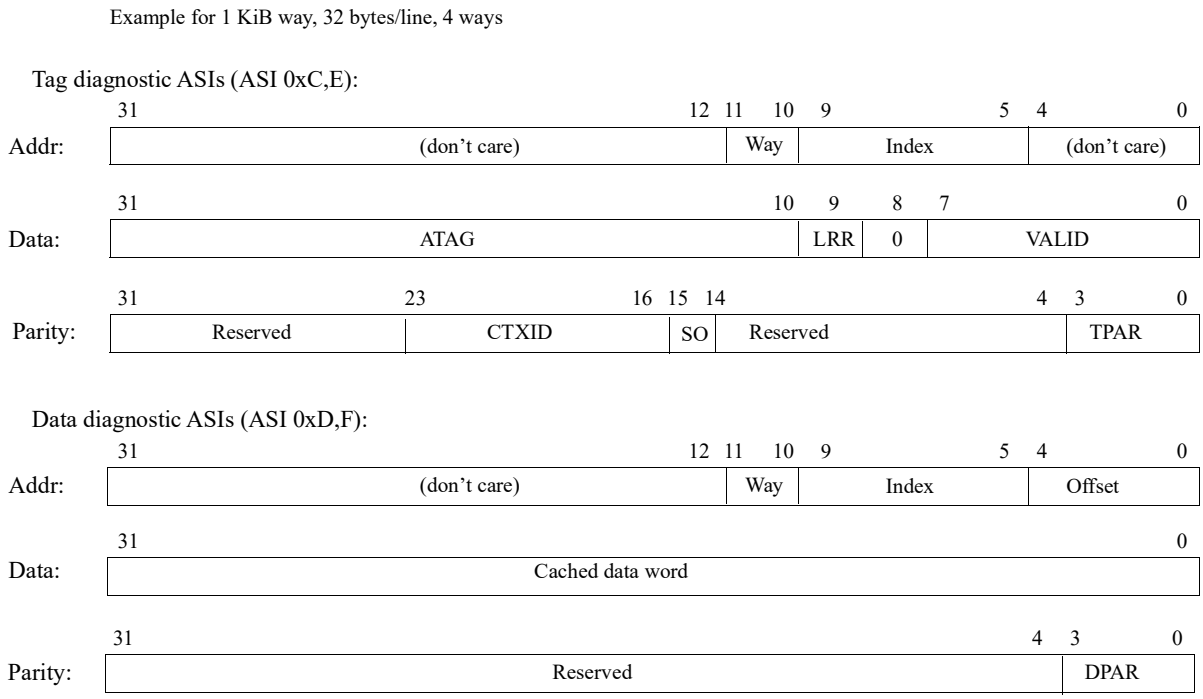


Figure 270. ASI 0xC-0xF address mapping and data layout

Field Definitions:

Address Tag (ATAG) - Contains the tag address of the cache line.

LRR - Used by LRR algorithm to store replacement history, otherwise 0.

Valid (V) - When set, the cache line contains valid data. The LEON4 caches only have one valid bit per cache line which is replicated for the whole 8-bit diagnostic field to keep software backward compatibility.

CTXID - Context ID, used when MMU is enabled.

SO - (Supervisor only access) It is used when MMU is enabled and *mmuen* (see Sec. 103.15) generic is set to 2. This bit is set to 1 when the associated page with the tag has access permission of 6 or 7 (supervisor access only) for other access permissions it is set to 0. (For detailed information about access permissions refer to the reference MMU part of the Sparc V8 manual)

TPAR - Byte-wise parity of tag bits, context ID parity is XOR:ed into bit 3.

DPAR - Byte-wise parity of data bits

NOTE: only the necessary bits will be implemented in the cache tag, depending on the cache configuration. As an example, a 4 KiB cache with 16 bytes per line would only have four valid bits and 20 tag bits. The cache rams are sized automatically by the RAM generators in GRLIB.

103.10.6ASI 0x10, 0x11, 0x13, 0x18 - Flush

For historical reasons there are multiple ASIs that flush the cache in different ways.

Writing to ASI 0x10 will flush the entire instruction cache. If MMU is implemented in the core, both instruction and data cache will be flushed.

Writing to ASI 0x11 will flush the data cache only.

Writing to ASI 0x13 will flush the instruction cache and data cache. Only available when MMU is implemented.

Writing to ASI 0x18 and 0x03, which is available only if MMU is implemented, will flush both the MMU TLB, the I-cache, and the D-cache. This will block execution for a few cycles while the TLB is flushed and then continue asynchronously with the cache flushes continuing in the background.

103.10.7ASI 0x19 and 0x04 - MMU registers

This ASI provides access to the MMU:s control and status registers. The following MMU registers are implemented:

*Table 2002.*MMU registers (ASI = 0x19)

Address	Register
0x000	MMU control register
0x100	Context pointer register
0x200	Context register
0x300	Fault status register
0x400	Fault address register

103.10.8ASI 0x1C - MMU and cache bypass

Performing an access via ASI 0x1C will act as if MMU and cache were disabled. The address will not be translated and the cache will not be used or updated by the access.

103.10.9ASI 0x1E - MMU physical/snoop tags diagnostic access

If the MMU has been configured to use separate snoop (physical) tags, they can be accessed via ASI 0x1E. This is primarily useful for RAM testing, and should not be performed during normal operation. This ASI is addressed the same way as the regular diagnostic ASI:s 0xC, 0xE, and the read/written data has the layout as shown below (example for a 1 KiB/way data cache):

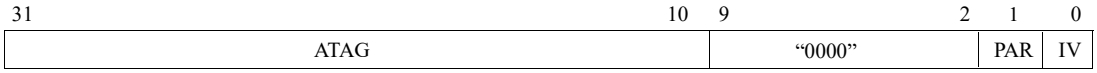


Figure 271. Snoop cache tag layout

[31:10] Address tag. The physical address tag of the cache line.
[1]: Parity. The odd parity over the data tag. Only used when processor is implemented with fault-tolerance features.
[0]: Invalid. When set, the cache line is not valid and will cause a cache miss if accessed by the processor. Only present if fast snooping is enabled.

103.11 Configuration registers

103.11.1PSR, WIM, TBR registers

The %psr, %wim, %tbr registers are implemented as required by the SPARC V8 manual.

*Table 2003.*LEON4 Processor state register (%psr)

31 28	27 24	23 20	19 16
IMPL	VER	ICC	RESERVED
0xF	0x3	0	0
r	r	r	r

15 14	13	12	11 8	7	6	5	4 0
RESERVED	EC	EF	PIL	S	PS	ET	CWP
0	0	0	0	1	1	0	0
r	r	rw*	rw	rw	rw	rw	rw

31:28	Implementation ID (IMPL), read-only hardwired to “1111” (15)
27:24	Implementation version (VER), read-only hardwired to “0011” (3) for LEON3/LEON4.
23:20	Integer condition codes (ICC), see sparcv8 for details
19:14	Reserved
13	Enable coprocessor (EC), read-only if coprocessor not implemented
12	Enable floating-point (EF), read-only if FPU not implemented.
11:8	Processor interrupt level (PIL) - controls the lowest IRQ number that can generate a trap
7	Supervisor (S)
6	Previous supervisor (PS), see sparcv8 for details
5	Enable traps (ET)
4:0	Current window pointer

*Table 2004.*LEON4 Window invalid mask (%wim)

31 NWIN	NWIN-1 0
RESERVED	WIM
0	NR
r	rw

*Table 2005.*LEON4 Trap base address regsiter (%tbr)

31 12	11 4	3 0
TBA	TT	R
*	0	0
rw	rw	r

31:12	Trap base address (TBA) - Top 20 bits used for trap table address
11:4	Trap type (TT) - Last taken trap type. Read only.
3:0	Always zero, read only

103.11.2ASR17, LEON4 configuration register

The ancillary state register 17 (%asr17) provides information on how various configuration options were set during synthesis. This can be used to enhance the performance of software, or to support enumeration in multi-processor systems. There are also a few bits that are writable to configure certain aspects of the processor.

*Table 2006.*LEON4 configuration register (%asr17)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
INDEX				DBP	RES	DBPM	REXV		REXM		RESERVED			CS	CF[1]
*				0	0	0	*		*		0			*	*
r				rw	r	rw	r		rw*		r			r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CF[0]	DWT	SVT	LD	FPU		M	V8	NWP			NWIN				
*	*	0	0	*		*	*	*			*				
r	rw	rw*	r	r		r	r	r			r				

31:28 Processor index (INDEX) - In multi-processor systems, each LEON core gets a unique index to support enumeration. The value in this field is identical to the *hindex* VHDL generic parameter in the VHDL model.

27 Disable Branch Prediction (DBP) - Disables branch prediction when set to ‘1’. Default value is ‘0’.

26 Reserved for future implementations

25 Disable Branch Prediction on instruction cache misses (DBPM) - When set to ‘1’ this avoids instruction cache fetches (and possible MMU table walk) for predicted instructions that may be annullated. This feature is on by default (reset value ‘1’).

24:23 REX version (REXV) - read-only field that is set to ‘00’ if REX is not implemented, ‘01’ if REX is implemented, ‘10’ and ‘11’ values are reserved for future implementations

22:21 REX mode (REXM) - set to ‘00’ for REX enabled, ‘01’ for REX illegal and ‘10’ for REX transparent mode. Writable with reset value ‘01’ when REX support has been enabled

20:18 Reserved for future implementations

17 Clock switching enabled (CS). If set, switching between AHB and CPU frequency is available.

16:15 CPU clock frequency (CF). CPU core runs at (CF+1) times AHB frequency.

14 Disable write error trap (DWT). When set, a write error trap (tt = 0x2b) will be ignored. Set to zero after reset.

13 Single-vector trapping (SVT) enable. If set, will enable single-vector trapping. Fixed to zero if SVT is not implemented. Set to zero after reset.

12 Load delay. 1-cycle load delay i s used.

11:10 FPU option. “00” = no FPU; “01” = GRFPU; “11” = GRFPU-Lite

9 If set, the optional multiply-accumulate (MAC) instruction is available

8 If set, the SPARC V8 multiply and divide instructions are available

7:5 Number of implemented watchpoints (NWP) (0 - 4)

4:0 Number of implemented registers windows corresponds to NWIN+1.

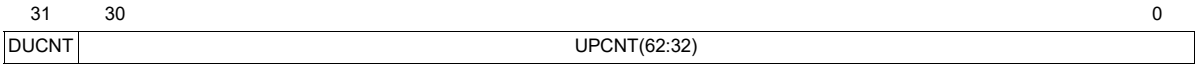
103.11.3ASR22-23 - Up-counter

The ancillary state registers 22 and 23 (%asr22-23) contain an internal up-counter that can be read by software without causing any access on the on-chip AMBA bus. The number of available bits in the counter is implementation dependent and is decided by the number of counter bits in the DSU time tag counter. %ASR23 contains the least significant part of the counter value and %ASR22 contains the most significant part. In case the implementation does not contain a debug support unit connected to the processor then the up-counter is not available (value is always zero).

The time tag value accessible in these registers is the same time tag value used for the system's trace buffers (if implemented) and for all processors connected to the same debug support unit. The time tag counter will increment when any of the trace buffers is enabled, or when the time tag counter is forced to be enabled via the DSU register interface, or when any processor has its %ASR22 Disable Up-counter (DUCNT) field set to zero.

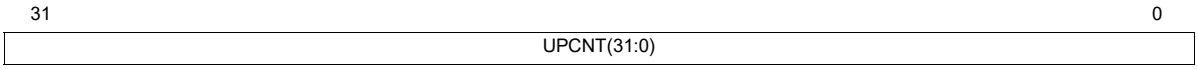
The up-counter value will increment even if all processors have entered power-down mode.

*Table 2007.*LEON4 up-counter MSbs (%ASR22)



31	Disable Up-counter (DUCNT) - Disable upcounter. When set to '1' the up-counter may be disabled. When cleared, the counter will increment each processor clock cycle. Default (reset) value is '1'.
30:0	Counter value (UPCNT(62:32)) - Most significant bits of internal up-counter. Read-only.

*Table 2008.*LEON4 up-counter LSbs (%ASR23)



31:0	Counter value (UPCNT(31:0)) - Least significant bits of internal up-counter. Read-only.

103.11.4ASR24-31, Hardware watchpoint/breakpoint registers

Each breakpoint consists of a pair of ancillary state registers (%asr24/25, %asr26/27, %asr28/29 and %asr30/31) registers; one with the break address and one with a mask:

%asr24, %asr26
%asr28, %asr30

31	2	1	0
WADDR[31:2]			IF
NR		0	0
rw		r	rw

%asr25, %asr27
%asr29, %asr31

31	2		0
WMASK[31:2]		DL	DS
NR		0	0
rw		rw	rw

Figure 272. Watch-point registers

WADDR - Address to compare against

WMASK - Bit mask controlling which bits to check (1) or ignore (0) for match

IF - break on instruction fetch from the specified address/mask combination

DL - break on data load from the specified address/mask combination

DS - break on data store to the specified address/mask combination

Note: Setting IF=DL=DS=0 disables the breakpoint

When there is a hardware watchpoint match and DL or DS is set then trap 0x0B will be generated. Hardware watchpoints can be used with or without the LEON4 debug support unit (DSU) enabled.

103.11.5Cache control register

The cache control register located at ASI 0x2, offset 0, contains control and status registers for the I and D cache.

*Table 2009.*LEON4 Cache Control Register (CCR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	STE	R	PS	TB				DS	FD	FI	FT		R	ST	R
0	*	0	*	0				0	0	0	*		0	*	0
r	rw*	r	rw*	rw*				rw	rw	rw	r		r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IP	DP	ITE		IDE		DTE		DDE		DF	IF	DCS		ICS	
0	0	0		0		0		0		0	0	0		0	
r	r	rw*		rw*		rw*		rw*		rw*	rw*	rw		rw	

31	Reserved
30	Snoop Tag Flag (STE) - Set when parity error is detected in the data physical (snoop) tags. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
29	Reserved
28	Parity Select (PS) - if set diagnostic read will return 4 check bits in the lsb bits, otherwise tag or data word is returned. Only available if fault-tolerance is enabled (FT field in this register is non-zero). When the technology specific FT scheme is implemented (FT field is “10”) then PS selects if correctable or uncorrectable errors should generate cache misses. The recommended setting is to use PS=’0’ that treats uncorrectable RAM errors in the same way as parity errors would traditionally be handled.
27:24	Test Bits (TB) - if set, check bits will be xored with test bits TB during diagnostic write. Only available if fault-tolerance is enabled (FT field in this register is “01”).
23	Data cache snoop enable (DS) - if set, will enable data cache snooping.
22	Flush data cache (FD). If set, will flush the data cache. Always reads as zero.
21	Flush Instruction cache (FI). If set, will flush the instruction cache. Always reads as zero.
20:19	FT scheme (FT) - “00” = no FT, “01” = 4-bit checking implemented, “10” - Technology-specific protection implemented.
18	Reserved for future implementations
17	Separate snoop tags (ST). This read-only bit is set if separate physical/snoop tags are implemented.
16	Reserved
15	Instruction cache flush pending (IP). This bit is set when an instruction cache flush operation is in progress
14	Data cache flush pending (DP). This bit is set when an data cache flush operation is in progress.
13:12	Instruction Tag Errors (ITE) - Number of detected parity errors in the instruction tag cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
11:10	Instruction Data Errors (IDE) - Number of detected parity errors in the instruction data cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
9:8	Data Tag Errors (DTE) - Number of detected parity errors in the data tag cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
7:6	Data Data Errors (DDE) - Number of detected parity errors in the data data cache. Only available if fault-tolerance is enabled (FT field in this register is non-zero).
5	Data Cache Freeze on Interrupt (DF) - If set, the data cache will automatically be frozen when an asynchronous interrupt is taken.
4	Instruction Cache Freeze on Interrupt (IF) - If set, the instruction cache will automatically be frozen when an asynchronous interrupt is taken.
3:2	Data Cache state (DCS) - Indicates the current data cache state according to the following: X0= disabled, 01 = frozen, 11 = enabled.
1:0	Instruction Cache state (ICS) - Indicates the current data cache state according to the following: X0= disabled, 01 = frozen, 11 = enabled.

103.11.6I-cache and D-cache configuration registers

The configuration of the two caches if defined in two registers: the instruction and data configuration registers. These registers are read-only, except for the REPL field that may be writable, and indicate the size and configuration of the caches. They are located under ASI 2 at offset 8 and 12.

*Table 2010.*LEON4 Cache configuration register

31	30	29 28	27	26 25 24	23 22 21 20	19	18 17 16
CL	R	REPL	SN	WAYS	WSIZE	LR	LSIZE
0	0	*	*	*	*	0	*
r	r	r	r	r	r	r	r

15 14 13 12 11 10 9 8 7 6 5 4	3	2	1 0
RESERVED	M	SO	RESERVED
0	*	*	0
r	r	r	r

31	Cache locking (CL). Set if cache locking is implemented (always zeo)
30:28	Cache replacement policy (REPL). 00 - no replacement policy (direct-mapped cache), 01 - least recently used (LRU), 10 - least recently replaced (LRR), 11 - random. This field is writable when LRU policy is implemented.
27	Cache snooping (SN). Set if snooping is implemented.
26:24	Cache associativity (WAYS). Number of ways in the cache: 000 - direct mapped, 001 - 2-way associative, 010 - 3-way associative, 011 - 4-way associative
23:20	Way size (WSIZE). Indicates the size (KiB) of each cache way. Size = 2^{SIZE}
19	Local ram (LR). Set if local scratch pad ram is implemented. (Always zero for LEON4)
18:16	Line size (LSIZE). Indicated the size (words) of each cache line. Line size = 2^{LSZ}
15:4	RESERVED
3	MMU present (M). This bit is set to ‘1’ if an MMU is present.
2	SO (supervisor only access) bit is present. This bit is set if *mmuen* generic is set to 2.
1:0	Reserved for future implementations

103.11.7MMU control register

The MMU control register is located in ASI 0x19 offset 0, and the layout can be seen in table 2011.

*Table 2011.*LEON4 MMU control register

31 28	27 24	23 21	20 18	17 16
IMPL	VER	ITLB	DTLB	PSZ
0	1	*	*	0
r	r	r	r	rw*

15	14	13 2	1	0
TD	ST	RESERVED	NF	E
NR	0	0	0	0
rw*	r	r	rw	rw

31:28	MMU Implementation ID. Hardcoded to “0000”

*Table 2011.*LEON4 MMU control register

27:24	MMU Version ID. Hardcoded to “0001”.
23:21	Number of ITLB entries. The number of ITLB entries is calculated as 2^{ITLB}. If the TLB is shared between instructions and data, this field indicates to total number of TLBs.
20:18	Number of DTLB entries. The number of DTLB entries is calculated as 2^{DTLB}. If the TLB is shared between instructions and data, this field is zero.
17:16	Page size. The size of the smallest MMU page. 0 = 4 Kib; 1 = 8 Kib; 2 = 16 Kib; 3 = 32 Kib. If the page size is programmable, this field is writable, otherwise it is read-only.
15	TLB disable. When set to 1, the TLB will be disabled and each data access will generate an MMU page table walk. See Section 103.12.3 for detailed information.
14	Separate TLB. This bit is set to 1 if separate instruction and data TLBs are implemented
13	This bit only exists if *mmuen* generic is set to 2. This bit is written to the SO (supervisor only access) part of the TAG during diagnostic writes.
13:2	Reserved for future implementations
1	No Fault. When NF= 0, any fault detected by the MMU causes FSR and FAR to be updated and causes a fault to be generated to the processor. When NF= 1, a fault on an access to ASI 9 is handled as when NF= 0; a fault on an access to any other ASI causes FSR and FAR to be updated but no fault is generated to the processor.
0	Enable MMU. 0 = MMU disabled, 1 = MMU enabled.

103.11.8MMU context pointer and context registers

The MMU context pointer register is located in ASI 0x19 offset 0x100 and the MMU context register is located in ASI 0x19 offset 0x200. They together determine the location of the root page table descriptor for the current context. Their definition follows the SRMMU specification in the SPARC V8 manual with layouts shown below..

*Table 2012.*LEON4 MMU context pointer register

31 2	1 0
CONTEXT TABLE POINTER	RES
NR	0
rw	r

31:2	Context table pointer, physical address bits 35:6 (note address is shifted 4 bits)
1:0	Reserved, always 0

*Table 2013.*LEON4 MMU context register

31 8	7 0
RESERVED	CONTEXT
0	0
r	rw

31:8	Reserved
7:0	Current context ID

In the LEON4, the context bits are OR:ed with the lower MMU context pointer bits when calculating the address, so one can use less context bits to reduce the size/alignment requirements for the context table.

103.11.9MMU fault status register

The MMU fault status register is located in ASI 0x19 offset 0x300, and the definition is based on the SRMMU specification in the SPARC V8 manual. The SPARC V8 specifies that the fault status regis-

ter should be cleared on read, on the LEON4 only the FAV bit is cleared on read. The FAV bit is always set on error in the LEON4 implementation, so it can be used as a valid bit for the other fields..

*Table 2014.*LEON4 MMU fault status register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED														EBE	
0														0	
r														r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EBE						L		AT			FT			FAV	OW
0						0		0			0			0	0
r						r		r			r			r	r

31:18	Reserved
17:10	External bus error (EBE) - Never set on the LEON3
9:8	Level (L) - Level of page table entry causing the fault
7:5	Access type (AT) - See V8 manual
4:2	Fault type (FT) - See table 1994
1	Fault address valid (FAV) - Cleared on read, always written to 1 on fault
0	Overwrite (W) - Multiple faults of the same priority encountered

103.11.10MMU fault address register

The MMU fault address register is located in ASI 0x19 offset 0x400, and the definition follows the SRMMU specification in the SPARC V8 manual..

*Table 2015.*LEON4 MMU fault address register

31 12	11 0
FAULT ADDRESS	RESERVED
NR	0
r	r

31:12	Top bits of virtual address causing translation fault
11:0	Reserved, always 0

103.12 Software considerations

103.12.1Register file initialization on power up

It is recommended that the boot code for the processor writes all registers in the IU and FPU register files before launching the main application. This allows software to be portable to both FT and non-FT versions of the LEON3 and LEON4 processors.

103.12.2Start-up

After reset, the caches are disabled and the cache control register (CCR) is 0. Before the caches may be enabled, a flush operation must be performed to initialized (clear) the tags and valid bits. A suitable assembly sequence could be:

```
flush
set 0x81000f, %g1
sta %g1, [%g0] 2
```

103.12.3MMU & TLB

After reset, the MMU is disabled and TLB is configured to be on (will not have any effect until MMU is enabled). Hence with default reset values the TLB has to be flushed before the MMU is being activated to initialize the valid bits in TLB. If the TLB is disabled while the MMU is active, the TLB must be flushed before enabled again.

103.12.4Data scrubbing (for fault-tolerant implementations)

There is generally no need to perform data scrubbing on either IU/FPU register files or the cache memory. During normal operation, the active part of the IU/FPU register files will be flushed to memory on each task switch. This will cause all registers to be checked and corrected if necessary. Since most real-time operating systems performs several task switches per second, the data in the register files will be frequently refreshed.

The similar situation arises for the cache memory. In most applications, the cache memory is significantly smaller than the full application image, and the cache contents is gradually replaced as part of normal operation. For very small programs, the only risk of error build-up is if a part of the application is resident in the cache but not executed for a long period of time. In such cases, executing a cache flush instruction periodically (e.g. once per minute) is sufficient to refresh the cache contents.

103.13 Vendor and device identifiers

The core has vendor identifiers 0x01 (Frontgrade Gaisler) and device identifiers 0x048. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

103.14 Implementation

103.14.1Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_async_reset_enable* is set.

103.14.2Technology mapping

LEON4 has two technology mapping VHDL generics, *fabtech* and *memtech*. The *fabtech* generic controls the implementation of some pipeline features, while *memtech* selects which memory blocks will be used to implement cache memories and the IU/FPU register file. *Fabtech* can be set to any of the provided technologies (0 - NTECH) as defined in the TECHMAP.GENCOMP package. See the GRLIB Users's Manual for available settings for *memtech*.

103.14.3RAM usage

The LEON4 core maps all usage of RAM memory on the *syncram*, *syncram_2p* and *syncram_dp* components from the technology mapping library (TECHMAP). The type, configuration and number of RAM blocks is described below.

Register file

The register file is implemented with six *synram_2p* blocks for all technologies where the *regfile_4p_infer* constant in TECHMAP.GENCOMP is set to 0. The organization of the syncram_2p is shown in the following table:

*Table 2016.*syncram_2p sizes for LEON4 register file

Register windows	Syncram_2p organization
2 - 3	32x32
4 - 7	64x32
8 - 15	128x32
16-31	256x31
32	512x32

If *regfile_4p_infer* is set to 1, the synthesis tool will automatically infer the register. On FPGA technologies, it can be in either flip-flops or RAM cells, depending on the tool and technology. On ASIC technologies, it will be flip-flops. The amount of flip-flops inferred is equal to the number of registers:

Number of flip-flops = ((NWINDOWS *16) + 8) * 32

FP register file

If FPU support is enabled, the FP register file is implemented with four *synram_2p* blocks when the *regfile_3p_infer* constant in TECHMAP.GENCOMP is set to 0. The organization of the syncram_2p blocks is 16x32.

If *regfile_3p_infer* is set to 1, the synthesis tool will automatically infer the FP register file. For ASIC technologies the number of inferred flip-flops is equal to number of bits in the FP register file which is 32 * 32 = 1024.

Cache memories

RAM blocks are used to implement the cache tags and data memories. Depending on cache configuration, different types and sizes of RAM blocks are used.

The tag memory is implemented with one *syncram* per cache way when no snooping is enabled. The tag memory depth and width is calculated as follows:

Depth = (cache way size in bytes) / (cache line size in bytes)

Width = 32 - log2(cache way size in bytes) + 1

For a 2 KiB cache way with 32 bytes/line, the tag RAM depth will be (2048/32) = 64. The width will be: 32 - log2(2048) + 1 = 32 - 11 + 1 = 22. The tag RAM organization will thus be 64x22 for the configuration. If the MMU is enabled, the tag memory width will increase with 8 to store the process context ID, and the above configuration will us a 64x30 RAM.

If simple (MMU-less) snooping is enabled, the data cache tag memory will instead of single-port RAM blocks be implemented with a dual-port RAMs (syncram_dp) of the same size.

If physical (MMU-compatible) snooping is enabled, the data cache tag memories will be implemented using two *syncram_2p* components (with one read-only and one write-only port) per way, one memory for virtual and one for separate physical tags. The size of the virtual tag block will be the same as when snooping is disabled. The physical tag block will have the same depth as above and the data width corresponds to the width of the tag: 32 - log2(way size). A 4 KiB data cache way will thus require a 32 - 12 = 20 bit wide RAM block for the physical tags.

The data part of the caches (storing instructions or data) is either 64 or 128 bit wide, depending on the setting of the *busw* VHDL generic. The depth is equal to the way size in bytes, divided by 8 (BUSW=64) or 16 (BUSW=128). A 64-bit cache way of 4 KiB will use two *syncram* components with and organization of 512x32. If the 128-bit AHB bus option is used, the data RAM will be divided on four 32-bit RAM blocks to allow loading of a 16-bit cache line in one clock. A 4 KiB data cache will then use four 256x32 RAM blocks.

Instruction Trace buffer

The instruction trace buffer will use four identical RAM blocks (*syncram*) to implement the buffer memory. The syncrams will always be 32-bit wide. The depth will depend on the *tbuf* VHDL generic, which indicates the total size of trace buffer in KiBs. If *tbuf* = 1 (1 KiB), then four RAM blocks of 64x32 will be used. If *tbuf* = 2, then the RAM blocks will be 128x32 and so on.

103.14.4Double clocking

LEON4 implements double clocking in the same way as LEON3. Please refer to the LEON3 Double-Clocking section in the LEON/GRLIB Configuration and Development Guide.

103.14.5Clock gating

LEON4 clock gating is described in the LEON/GRLIB Configuration and Development Guide.

103.14.6Scan support

If the *scantest* VHDL generic is set to 1, support for scan testing is enabled. This will make use of the AHB scan support signals in the following manner: when AHBI.testen and AHBI.scanen are both '1', the select signals to all RAM blocks (cache RAM, register file and DSU trace buffers) are disabled. This means that when the scan chain is shifted, no accidental write or read can occur in the RAM blocks. The scan signal AHBI.testrst is not used as there are no asynchronous resets in the LEON4 core.

103.15 Configuration options

Table 2017 shows the configuration options of the core (VHDL generics).

*Table 2017.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
fabtech	Target technology	0 - NTECH	0 (inferred)
memtech	Vendor library for regfile and cache RAMs. Bits 16, 17 and 18 of this generic can be used to for the MMU TLB data RAM, IU register file and FP register file to inferred technology: + 2**16: Force inferred technology for MMU TLB data RAM + 2**17: Force inferred technology for IU register file + 2**18: Force inferred technology for FP register file Adding the value (2**17) is equivalent to setting the grlib.gencomp.regfile_4p_infer(memtech) to 1 (used for some technologies to force the register file implementations to inferred). Adding the value (2**18) is equivalent to setting the grlib.gencomp.regfile_3p_infer(memtech) to 1 (used for some technologies to force the register file implementations to inferred).	0 - 16#FFFFFFFF#	0 (inferred)
nwindows	Number of SPARC register windows. Choose 8 windows to be compatible with Bare-C and RTEMS cross-compilers.	2 - 32	8
dsu	Enable Debug Support Unit interface	0 - 1	0
fpu	Floating-point Unit 0 : no FPU 1 - 7: GRFPU 1 - inferred multiplier, 2 - DW multiplier, 3 - Module Generator multiplier 8 - 14: GRFPU-Lite 8 - simple FPC, 9 - data forwarding FPC, 10 - non-blocking FPC 16 - 31: as above (modulo 16) but use netlist 32 - 63: as above (modulo 32) but uses shared GRFPU interface Netlist or technology specific multiplier is recommended when using Synplify to target Virtex-5.	0 - 63	0

*Table 2017.*Configuration options

Generic	Function	Allowed range	Default
v8	Generate SPARC V8 MUL and DIV instructions This generic is assigned with the value: *mult + 4*struct* Where mult selects between the following implementation options for the multiplier and divider: 0 : No multiplier or divider 1 : 16x16 multiplier 2 : 16x16 pipelined multiplier 16#32# : 32x32 pipelined multiplier Where *struct* selects the structure option for the integer multiplier. The following structures can be selected: 0: Inferred by synthesis tool 1: Generated using Module Generators from NTNU 2: Using technology specific netlists (techspec). Only supported for RTAX-D FPGAs. Other technologies will assert a simulation error. 3: Using Synopsys DesignWare (DW02_mult and DW_mult_pipe)	0 - 16#3F#	0
cp	Generate co-processor interface	0 -1	0
mac	Generate SPARC V8e SMAC/UMAC instruction. Can only be used together with a 16x16 multiplier.	0 - 1	0
pclow	Least significant bit of PC (Program Counter) that is actually generated. PC[1:0] are always zero and are normally not generated. Generating PC[1:0] makes VHDL-debugging easier.	0, 2	2
notag	Disable tagged instructions.	0 - 1	0
nwp	Number of watchpoints	0 - 4	0
icen	Enable instruction cache	0 - 1	1
irepl	Instruction cache replacement policy. 0 - least recently used (LRU)/LRR/random/direct, 2 - random only	0, 2	0
isets	Number of instruction cache ways	1 - 4	1
ilinesize	Instruction cache line size in number of words	4, 8	4
isetsize	Size of each instruction cache way in KiB	1 - 256	1
isetlock	Unused	0 - 1	0
dcen	Data cache enable	0 - 1	1
drepl	Data cache replacement policy. 0 - least recently used (LRU)/LRR/random/direct, 2 - random only	0, 2	0
dsets	Number of data cache ways	1 - 4	1
dlinesize	Data cache line size in number of words	4, 8	4
dsetsize	Size of each data cache way in KiB Note: If the processor is implemented with the MMU then the cache way size needs to be equal or less then the MMU page size for hardware cache coherency.	1 - 256	1
dsetlock	Unused	0 - 1	0
dsnoop	Enable data cache snooping Bit 0-1: 0: disable, 1: obsolete, 2: enable Bit 2: 0: simple (no-MMU) snooping, 1: separate physical tags	0 - 6	0
ilram	Enable local instruction RAM (not used at this point)	0 - 1	0
ilramsize	Local instruction RAM size in kB (not used at this point)	1 - 512	1

*Table 2017.*Configuration options

Generic	Function	Allowed range	Default
ilramstart	8 MSB bits used to decode local instruction RAM area (not used at this point)	0 - 255	16#8E#
dlram	Enable local data RAM (scratch-pad RAM) (not used at this point)	0 - 1	0
dlramsize	Local data RAM size in kB (not used at this point)	1 - 512	1
dlramstart	8 MSB bits used to decode local data RAM area (not used at this point)	0 - 255	16#8F#
mmuen	Enable memory management unit (MMU) Note: Bus snooping is required to avoid cache aliasing effects when the MMU is enabled if the cache has more than one way. 0 : MMU does not exist. 1 : MMU exists. 2 : MMU exists and the cache tags include an additional bit called SO (supervisor only access). See sec. 103.4.2 for more details. If MMU is going to be instantiated in the processor, it is recommended to set the *mmuen* generic to 2.	0 - 2	0
itlbnum	Number of instruction TLB entries	2 - 64	8
dtlbnum	Number of data TLB entries	2 - 64	8
tlb_type	0 : separate TLB with slow write 1: shared TLB with slow write 2: separate TLB with fast write	0 - 2	1
tlb_rep	LRU (0) or Random (1) TLB replacement	0 - 1	0
lddel	Unused. LEON4 is always implemented with load delay 1.	1 - 2	2
disas	Print instruction disassembly in VHDL simulator console.	0 - 1	0
tbuf	Size of instruction trace buffer in kB (0 - instruction trace disabled). For values 1-64 a single-port trace buffer of size *tbuf* is used. For values 65-128 a two-port trace buffer of size *tbuf*-64 is used.	0 - 128	0
pwd	Power-down. 0 - disabled, 1 - area efficient, 2 - timing efficient.	0 - 2	1
svt	Enable single-vector trapping	0 - 1	0
rstaddr	Default reset start address. This generic sets the 20 most significant bits of the reset address. The reset address must always be aligned on a 4 KiB address boundary. If this generic is set to 16#fffff# the processor will read its start address from the interrupt controller interface signal IRQI.RSTVEC (dynamic reset start address). See section 103.2.18 for more information.	0 - (2**^{20}-1)	0
smp	Enable multi-processor support 0: SMP support disabled 1- 15: SMP enabled, cpu index is taken from hindex generic 16-31: SMP enabled, cpu index is taken from irqi.index signal	0 - 31	0
cached	Fixed cacheability mask. See sections 103.3 and 103.7 for more information.	0 - 16#FFFF#	0
clk2x	Enables double-clocking. See section 103.14.4 and the LEON/GRLIB Design and Configuration Guide. Not present on all top-level entites.	0 - 15	0
scantest	Enable scan test support	0 - 1	0

*Table 2017.*Configuration options

Generic	Function	Allowed range	Default
wbmask	Wide-bus mask. Indicates which address ranges are 64/128 bit capable. Treated as a 16-bit vector with LSB bit (right-most) indicating address 0 - 0x10000000. See section 103.7 for more information.	0 - 16#FFFF#	0
busw	Bus width of the wide bus area (64 or 128). See section 103.7 for more information.	64, 128	64
netlist	Use technology specific netlist	0 - 1	0
ft	Register file and cache memory protection, bitfield Bit 10:7: Selects technology specific protection cache protection. 0 to disable, 5 to enable. Bit 6:4: Unused Bit 3: enable cache memory parity protection Bit 2:0 Register file SEU protection. (0: no protection; 4: TMR, 6: Tech-specific) Technology specific protection is further documented in the GRLIB-FT User's Manual (grlib-ft.pdf).	0 - 2047	0
npasi	Enable SPARC V8E nonprivileged ASI access. 0 - All accesses to alternate address space are privileged. 1 - LOAD and STORE from alternate space instructions accessing ASIs 0x00-0x7F are privileged, ASIs 0x80 - 0xFF are nonprivileged.	0 - 1	0
pwrpsr	Enable SPARC V8E partial write PSR (WRPSR).	0 - 1	0
ahbpipe	Add pipeline registers to AHB read data vectors. Only has effect for *busw* = 128. Setting only present on leon4x entity.	0 - 1	0
rex	Enable LEON-REX extension	0 - 1	0
mmupgsz	MMU Page size. 0 = 4K, 1 = 8K, 2 = 16K, 3 = 32K, 4 = programmable.	0 - 4	0

103.16 Signal descriptions

Table 2018 shows the interface signals of the core (VHDL ports). Note that there are several top-level entities available for the LEON4 processor: *leon4x, leon4s, leon4sh, leon4s2x, leon4ft* and *leon4cg*. The *leon4x* entity exposes all signals and settings. The other entities are wrapper around *leon4x*.

*Table 2018.*Signal descriptions

Signal name	Field	Type	Function	Active
leon4x:AHBCLK leon4s2x:CLK	N/A	Input	AMBA clock in 2x mode Note that this only applies to the processor entities listed.	-
leon4x:CPUCLK leon4s: CLK leon4sh: CLK leon4s2x: CLK2 leon4ft: CLK leon4cg: CLK	N/A	Input	Processor clock, can be gated. Note that this clock has different names depending on which top-level entity that is used to instantiate the processor. For example, LEON4S only has one clock input which covers three of the rows in this table (Processor clock, FPU-clock, free running processor clock).	-

*Table 2018.*Signal descriptions

Signal name	Field	Type	Function	Active
leon4x:GCPUCLK leon4s: CLK leon4sh: CLK leon4s2x: GCLK2 leon4ft: GCLK leon4cg: GCLK	N/A	Input	Free running processor clock. Note that this clock has different names depending on which top-level entity that is used to instantiate the processor. For example, LEON4S only has one clock input which covers three of the rows in this table (Processor clock, FPU-clock, free running processor clock).	-
leon4x:FPUCLK leon4s: CLK leon4sh: CLK leon4s2x: GCLK2 leon4ft: GCLK leon4cg: GCLK	N/A	Input	FPU clock, can be gated. Note that this clock has different names depending on which top-level entity that is used to instantiate the processor. For example, LEON4S only has one clock input which covers three of the rows in this table (Processor clock, FPU-clock, free running processor clock).	-
RSTN	N/A	Input	Reset	Low
AHBI	*	Input	AHB master input signals	-
AHBO	*	Output	AHB master output signals	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO[]	*	Input	AHB slave output signals from all slaves on same bus. The processor makes use of the plug&play sideband signals to decode cacheability information of the bus. This can be overridden by the cached VHDL generic.	-
IRQI	IRL[3:0]	Input	Interrupt level	High
	RESUME	Input	Reset power-down and error mode	High
	RSTRUN	Input	Start after reset (SMP system only)	High
	RSTVEC[31:12]	Input	Reset start addr. (SMP and dynamic reset addr.)	-
	INDEX[3:0]	Input	CPU index when SMP = 2	-
	PWDSETADDR	Input	In power-down/error mode, shift PC to nPC and set PWDNEWADDR to PC.	High
	PWDNEWADDR [31:2]	Input	New PC value used with PWDSETADDR	-
	FORCEERR	Input	Force CPU into error mode	High
IRQO	INTACK	Output	Interrupt acknowledge	High
	IRL[3:0]	Output	Processor interrupt level	High
	PWD	Output	Processor in power-down mode	High
	FPEN	Output	Floating-point unit enabled	High
	ERR	Output	Processor in error mode	High
DBGI	-	Input	Debug inputs from DSU	-
DBGO	-	Output	Debug outputs to DSU	-
	ERROR		Processor in error mode, execution halted	Low
GCLK		Input	Gated processor clock for LEON4cg	

* see GRLIB IP Library User's Manual

103.17 Signal definitions and reset values

When the processor enters error mode, the *errorn* output is driven active.

The signals and their reset values are described in table 2019.

*Table 2019.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
errorn	Tri-state output	Processor error mode indicator	Low	Tri-state

103.18 Timing

The timing waveforms and timing parameters are shown in figure 273 and are defined in table 2020.

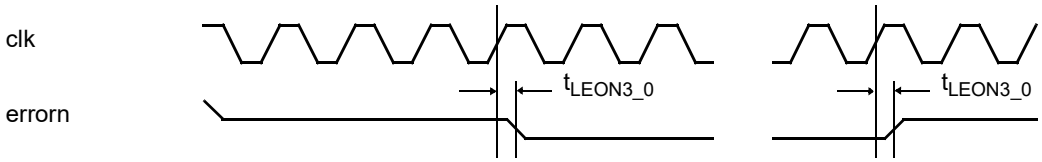


Figure 273. Timing waveforms

*Table 2020.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{LEON3_0}	clock to output delay	rising *clk* edge	TBD	TBD	ns

103.19 Library dependencies

Table 2021 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2021.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	LEON3, LEON4	Component, signals	LEON4 component declaration, interrupt and debug signals declaration

103.20 Component declaration

The LEON4 core has the following component declaration. There are also LEON4 top-levels that support clock gating (leon4cg), double-clocking (leon4s2x), shared FPU (leon4sh) and all available interfaces (leon4x). See the GRLIB template designs for instantiation examples.

```
entity leon4s is
  generic (
    hindex    : integer               := 0;
    fabtech   : integer range 0 to NTECH  := DEFFABTECH;
    memtech   : integer range 0 to NTECH  := DEFMEMTECH;
    nwindows  : integer range 2 to 32 := 8;
    dsu       : integer range 0 to 1  := 0;
    fpu       : integer range 0 to 31 := 0;
    v8        : integer range 0 to 63 := 0;
    cp        : integer range 0 to 1  := 0;
    mac       : integer range 0 to 1  := 0;
    pclow     : integer range 0 to 2  := 2;
    notag     : integer range 0 to 1  := 0;
    nwp       : integer range 0 to 4  := 0;
    icen      : integer range 0 to 1  := 0;
    irepl     : integer range 0 to 2  := 2;
    isets     : integer range 1 to 4  := 1;
    ilinesize : integer range 4 to 8  := 4;
    isetsize  : integer range 1 to 256 := 1;
    isetlock  : integer range 0 to 1  := 0;
    dcen      : integer range 0 to 1  := 0;
    drepl     : integer range 0 to 2  := 2;
    dsets     : integer range 1 to 4  := 1;
    dlinesize : integer range 4 to 8  := 4;
    dsetsize  : integer range 1 to 256 := 1;
    dsetlock  : integer range 0 to 1  := 0;
    dsnoop    : integer range 0 to 6  := 0;
    ilram      : integer range 0 to 1 := 0;
    ilramsize  : integer range 1 to 512 := 1;
    ilramstart : integer range 0 to 255 := 16#8e#;
    dlram      : integer range 0 to 1 := 0;
    dlramsize  : integer range 1 to 512 := 1;
    dlramstart : integer range 0 to 255 := 16#8f#;
    mmuen     : integer range 0 to 2 := 0;
    itlbnum   : integer range 2 to 64 := 8;
    dtlbnum   : integer range 2 to 64 := 8;
    tlb_type  : integer range 0 to 3  := 1;
    tlb_rep   : integer range 0 to 1  := 0;
    lddel     : integer range 1 to 2  := 2;
    disas     : integer range 0 to 2  := 0;
    tbuf      : integer range 0 to 64 := 0;
    pwd       : integer range 0 to 2  := 2;     -- power-down
    svt       : integer range 0 to 1  := 1;     -- single vector trapping
    rstaddr   : integer               := 0;
    smp       : integer range 0 to 15 := 0;     -- support SMP systems
    cached    : integer               := 0;     -- cacheability table
    scantest  : integer               := 0;
    wbmask    : integer               := 0;     -- Wide-bus mask
    busw      : integer               := 64;    -- AHB/Cache data width (64/128)
    ft        : integer               := 0
);
  port (
    clk    : in  std_ulogic;
    rstn   : in  std_ulogic;
    ahbi   : in  ahb_mst_in_type;
    ahbo   : out ahb_mst_out_type;
    ahbsi  : in  ahb_slv_in_type;
    ahbso  : in  ahb_slv_out_vector;
    irqi   : in  l3_irq_in_type;
    irqo   : out l3_irq_out_type;
    dbgi   : in  l3_debug_in_type;
    dbgo   : out l3_debug_out_type
  );
end;
```

104 LEON5SYS - High-performance SPARC V8 32-bit Processor Subsystem

104.1 Overview

LEON5 is a 32-bit processor core conforming to the IEEE-1754 (SPARC V8) architecture. It is designed for embedded applications, combining high performance with low complexity and low power consumption.

The LEON5SYS block combines the LEON5 processor with essential peripherals needed to create a LEON5-based system-on-chip.

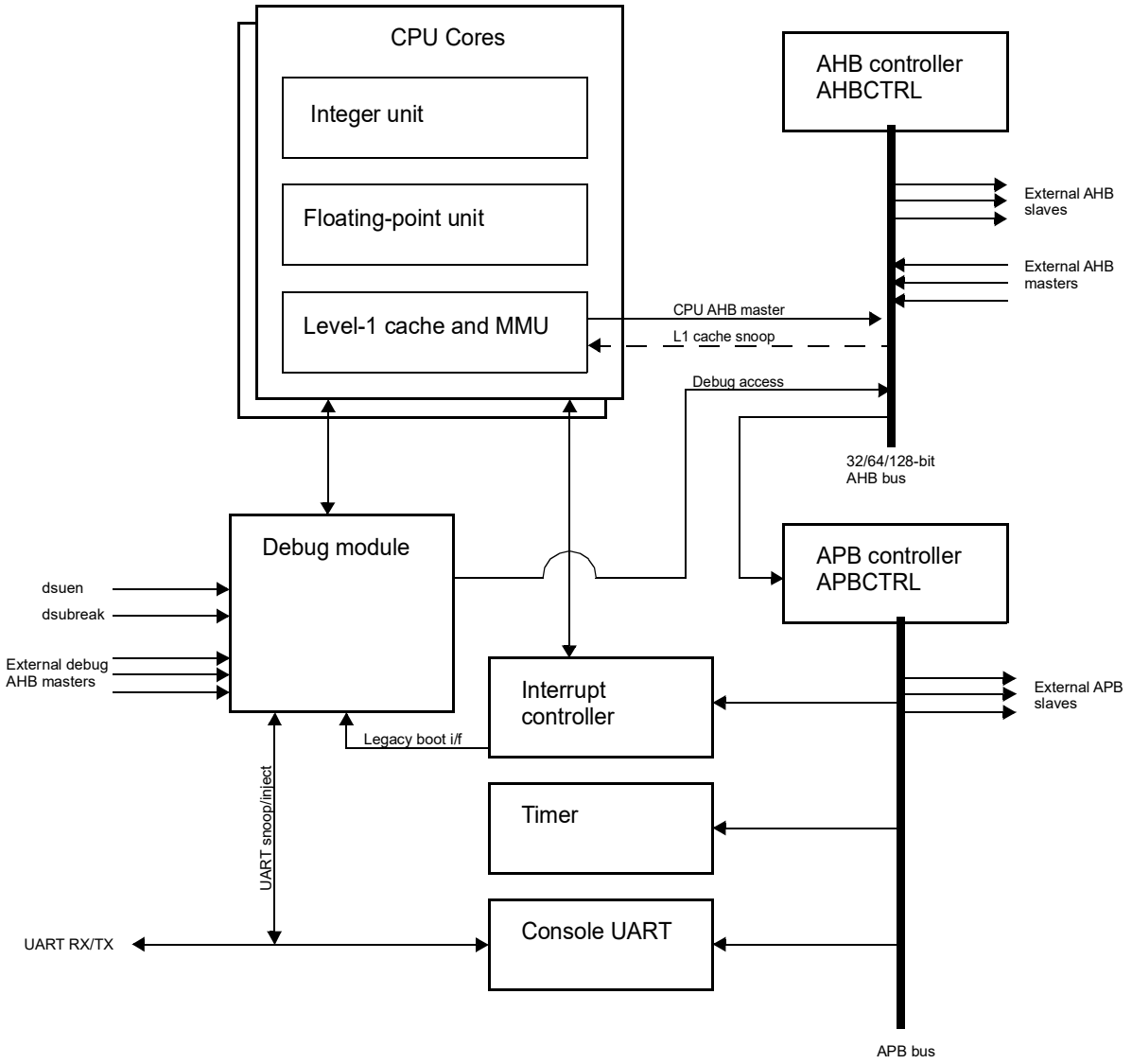


Figure 274. LEON5SYS subsystem block diagram

104.1.1 Integer unit

The LEON5 integer unit implements the full SPARC V8 specification, including hardware multiply and divide instructions. The processor has 8 register windows. The dual-issue pipeline consists of 8 stages with a separate instruction and data cache interface (Harvard architecture).

104.1.2 Level 1 cache

LEON5 has a separate instruction and data cache. The instruction cache maintains one valid bit per cache line and performs buffer forwarding during line-refill to minimize refill latency. The data cache has one valid bit per cache line, uses write-through policy and implements a four double-word write-buffer. Bus-snooping on the AHB bus is used to maintain cache coherency for the data cache. Optionally, tightly coupled memory can be added to the caches that can be accessed with the same speed as data in the cache.

104.1.3 Floating-point unit

The LEON5 integer interfaces a floating-point unit (FPU). Two FPU options are available, one for the high-performance GRFPU and one for the NanoFPU core. The floating-point processor executes in parallel with the integer unit, and does not block the operation unless a data or resource dependency exists. Note that the GRFPU is provided separately.

104.1.4 Memory management unit

A SPARC V8 Reference Memory Management Unit (SRMMU) is integrated into the LEON5 cache controller. The SRMMU implements the full SPARC V8 MMU specification, and provides mapping between multiple 32-bit virtual address spaces and physical memory. A three-level hardware table-walk is implemented, and the MMU has a separate fully associative TLB for instruction and data.

104.1.5 On-chip debug support

The LEON5 subsystem includes functionality to allow non-intrusive debugging on target hardware. Interfaces to access the debug functionality are connected using a dedicated AHB port that does not interfere with normal operation.

To aid software debugging, hardware watchpoint registers can be enabled in the integer pipeline. Each register can cause a breakpoint trap on an arbitrary instruction or data address range. The watchpoints can be used to put the processor in stopped state for debugging.

When accessed through the debug module, full access to all processor registers and caches is provided, as well as direct read/write access to the AHB bus. The debug interfaces also allows single stepping, instruction tracing and hardware breakpoint/watchpoint control. An internal trace buffer can monitor and store executed instructions, which can later be read out via the debug interface.

104.1.6 Interrupt interface

An interrupt controller is included in the subsystem, collecting interrupts from external and internal sources and presenting them to the integer units.

The LEON5 integer unit supports the SPARC V8 interrupt model with a total of 15 asynchronous interrupt levels, and the interrupt controller provides extended interrupt support used to get up to 31 interrupts. More interrupts can be handled via interrupt sharing in software.

104.1.7 Standard peripherals

The LEON5SYS block includes standard APBUART and GPTIMER peripherals as are typically expected by software.

104.1.8 AMBA interface

An AMBA 2.0 AHB bus is included in the subsystem. Each processor’s cache system implements an AMBA AHB master to load and store data to/from the caches that is connected to the AHB bus. During line refill, incremental burst are generated to optimise the data transfer. The AMBA interface can be configured to 32, 64 or 128-bit data width.

External slaves (memory controllers) and masters (peripheral DMA) are connected to the subsystem via ports. The external AHB slaves may either be narrow (support maximum 32-bit width) or wide (support full width of AMBA bus) and it is possible to have a mix of narrow and wide AHB slaves.

An APB controller is also included in the subsystem for the internal peripherals, and additional external peripherals may be connected via a subsystem port.

104.1.9 Power-down mode

The LEON5 processor core implements a power-down mode, which halts the pipeline and caches until the next interrupt. The processor supports optional clock gating during the power down period by providing a clock-enable signal that can be tied to an external clock gate cell, and by providing a separate clock input for the small part of the CPU that needs to run during power-down to check for wake-up conditions and maintain cache coherency.

104.1.10Multi-processor support

LEON5 is designed to be used in multi-processor systems both in SMP and AMP configurations. Each processor has a unique index to allow processor enumeration. The write-through caches combined with the snooping mechanism guarantees memory coherency in shared-memory systems consistent with the SPARC TSO (Total Store Order) memory model. A CPU to CPU interface is provided to allow the processors to start and stop each other.

104.2 LEON5 integer unit

104.2.1 Overview

The LEON5 integer unit contains the main processor pipeline and implements the integer part of the SPARC V8 instruction set. The implementation is focused on high performance. The LEON5 integer unit has the following main features:

- 8-stage dual issue instruction pipeline
- Separate instruction and data cache interface
- Hardware multiplier
- Radix-2 divider (non-restoring)
- Branch prediction and branch target buffer
- Late ALU for load and multiply results
- Debug support

Figure 275 shows a block diagram of the integer unit.

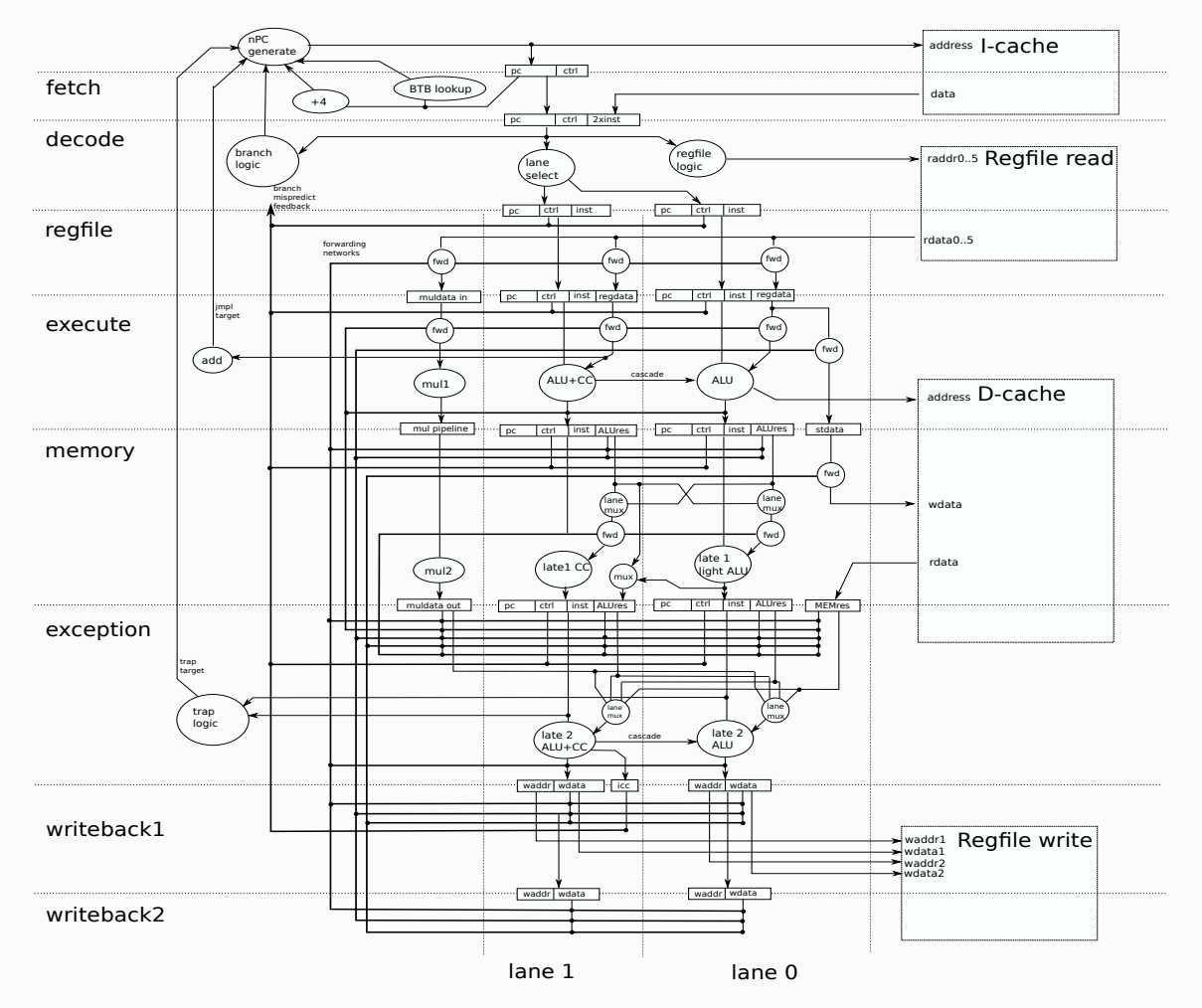


Figure 275. LEON5 integer unit datapath diagram

104.2.2 Instruction pipeline

The LEON5 integer unit uses a dual instruction issue pipeline with 8 stages:

1. FE (Instruction Fetch): A 64-bit word (holding two instructions) is fetched from the instruction cache. The instruction is valid at the end of this stage and is latched inside the IU. Branch target buffer and branch history is also accessed during this stage.
2. DE (Decode): The two instructions are decoded and the CALL/Branch target addresses are generated. It is determined if dual issue is possible, which instruction goes into which lane of the following stages. Floating point operations stalls in this stage if any data dependency exists.
3. RA (Register access): Operands are read from the register file or from internal data bypasses. Stalls due to data dependencies, or multi-cycle operations (like division, atomic) are generated in this stage.
4. EX (Execute): ALU, logical, and shift operations are performed in both lanes. Forwarding to the ALU inputs from XC stage is performed to reduce the LOAD/MUL latency to 1. The two lanes' ALU:s may be cascaded if lane 0 depends on lane 1s result. For memory operations (e.g., LD) and for JMPL/RETT, the address is generated.
5. ME (Memory): Read data is received from the data cache is accessed. Store data is generated and fed into the data cache inputs at this time. ALU:s with reduced functionality are included in this stage to allow delayed dependency resolution in some cases.
6. XC (Exception) Traps and interrupts are resolved. For cache reads, the data is aligned. For cache writes, the data update takes place inside the cache memory on this cycle. Late ALU:s with full functionality are included in this stage to allow delayed dependency resolution.
7. WR1 (Write): Pipeline stage to avoid long paths through the XC stage logic into register file data inputs.
8. WR2 (Write): The result of ALU and cache operations are written back to the register file.

Table 2022 lists the cycles per instruction, not considering dual issue (assuming cache hit and no icc or load interlock):

*Table 2022.*Instruction timing

Instruction	Cycles	Comment
JMPL/RETT	4	Including delay slot
CALL	2	Including delay slot
SMUL/UMUL	1	
Load	1	
Store, 32/64 bit	1 / 2	Two cycles if following instruction is load or 8/16 bit store
Store, 8/16 bit	1 / 2	Two cycles if following instruction is load or store
SDIV/UDIV	35	
Taken Trap	6	
Atomic load/store	5	
All other instructions	**1**	

104.2.3 Branch predictor and branch target buffer

The pipeline implements a branch predictor in the decode stage. The predictor uses a two level adaptive predictor scheme with 128 entries and 4 length history buffer. The pipeline can also be switched over to a static always-taken prediction mode to improve timing predictability.

A branch target buffer is implemented to predict branch buffer targets in the fetch stage to avoid a one-cycle gap between branches and the target instruction. The branch target buffer is only used for branches in the first part of a pair (on address multiple of 8) as the delay slot will take up the following cycle in the other case.

104.2.4 Late ALU

Late ALUs are implemented to allow ALU and ALUcc operations normally done in the EX stage to be deferred to the ME or XC states. This allows masking the load and multiply to ALU operation latency.

The late ALU in the ME stage is reduced in functionality to only handle the most common operations, other cases will be deferred to the late ALU in the XC stage.

Arithmetic operations that can use Late ALU are ADD, AND, OR, XOR, SUB, ANDN, ORN, XNOR, SLL, SRL, SRA, ADDCC, ANDCC, ORCC, XORCC, SUBCC, ANDNCC, ORNCC and XNORCC.

104.2.5 Interlocks

Additional conditions that can extend an instructions duration in the pipeline are listed in the table and text below.

Branch target interlock: When the least significant 3-bits of the program counter of a branch is 0x0 and the branch is predicted to be taken, if branch target buffer lookup is a miss or branch target buffer is disabled one cycle interlock is incurred in which branch target address is calculated. For branches with PC ending with 0x4, the delay slot is fetched while calculating the target address so no interlock is incurred.

Load delay: When using data resulting on a load shortly after the load, if late ALU is enabled and the instruction can use late alu (see) no delay cycle is incurred. Otherwise there can be up to two cycle delay.

Mul delay: Same as load delay.

Late alu delay: If an operation depends on the result of a late alu operation and it can not itself use the late alu (see) there can be up to three cycle delays. If a store operation needs to store the result of a late alu operation there can be at most 1 cycle delay.

Hold cycles: During cache miss processing or when blocking on the store buffer, the pipeline will be held still until the data is ready, effectively extending the execution time of the instruction causing the miss by the corresponding number of cycles. Note that since the whole pipeline is held still, hold cycles will not mask load delay or interlock delays. For instance on a load cache miss followed by a data-dependent instruction, both hold cycles and load delay may be incurred.

FPU/Coprocessor: The floating-point unit or coprocessor may need to hold the pipeline or extend a specific instruction. When this is done is specific to the FP/CP unit.

104.2.6 Branch Handling

When branch prediction is enabled, instructions following the branch are fetched speculatively until branch resolution happens. Branch resolution can happen on any stage between the decode stage and the exception stage, meaning that it can take 2-6 cycles until a branch is resolved. If the branch prediction is correct, then no extra delay is incurred while the branch is resolved regardless in which stage it is resolved, since instructions from predicted path are fetched. If there is a branch misprediction, mispredicted instructions are annulled and the true target of the branch is fetched, leading to up to 6 cycles of effective timing penalty depending on in which stage the branch is resolved.

If branch prediction is configured to be disabled, and condition codes are not resolved when a branch reaches the regfile access stage, the branch will stall there. In addition, only the delay slot instruction will be in the pipeline so there can be up to two bubbles inserted on fetch and decode stages. The number of stall cycles of a branch can be up to three cycles if late alu is enabled, an instruction updating the condition code is immediately before the conditional branch, and the instruction updating the condition codes gets calculated in late alu,

104.2.7 Dual issue pairing

The processor pipeline can dual issue pairs of instructions. In case the instructions do not have register dependencies, table 2023 below shows possible pairings. In cases where the registers do have register dependency, table 2024 lists the cases where pairing is possible.

Table 2023. Dual issue pairings in non-dependency case

	ALU	ALUccout	ALUccin	MUL	DIV	Load	Store	FPop1	FPop2	FP-Load	FP-Store	CALL	Ba,Bn	Bicc	JMPL
ALU	P	P	P	P	-	P	P	P	P	P	P	P	P	P	P
ALUccout	P	P	-	P	-	P	P	P	P	P	P	P	P	P	P
ALUccin	P	-	P	P	-	P	P	P	P	P	P	P	P	P	P
MUL	P	P	P	-	-	P	P	P	P	P	P	P	P	P	P
DIV	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Load	P	P	P	P	-	-	-	P	P	-	-	P	P	P	P
Store	P	P	P	P	-	-	-	P	P	-	-	P	P	P	P
FPop1	P	P	P	P	-	P	P	-	-	-	-	P	P	P	P
FPop2	P	P	P	P	-	P	P	-	-	-	-	P	P	P	P
FP-Load	P	P	P	P	-	-	-	-	-	-	-	P	P	P	P
FP-Store	P	P	P	P	-	-	-	-	-	-	-	P	P	P	P
CALL	P	P	P	P	-	P	P	P	P	P	P	-	-	-	-
Ba,Bn	P	P	P	P	-	P	P	P	P	P	P	-	-	-	-
Bicc	P	P	P	P	-	P	P	P	P	P	P	-	-	-	-
JMPL	P	P	P	P	-	P	P	P	P	P	P	-	-	-	-

Table 2024. Dual issue pairings when register dependencies present

First operation		Second operation		Dep. from	Dep. to	Pairability
ALU	Any	ALU	Any	rd	rs1 or rs2	Only pairable for specific cases shown below.
	Logic / SETHI		Logic / SETHI	rd	rs1 or rs2	Pairable
	ADD / SUB		Logic	rd	rs1 or rs2	Pairable
	Logic		Shift	rd	rs1 (shifted value)	Only shifted value is pairable, the shift count dependency is not pairable
ALU	Any	Load / FP-Load	Any	rd	rs1 or rs2	Only pairable for specific cases shown below.
	ADD		Integer load	rd	rs1 or rs2	Pairable
ALU		Store / FP-Store		rd	rs1 or rs2	No pairing.
ALU		Store		rd	rd (store data)	Pairable provided that the ALU operation is not delayed to the late ALU.
ALU		JMPL		rd	rs1 or rs2	No pairing
Load / MUL		ALU		rd	rs1 or rs2	Pairable, late ALU resolution
Others						No pairing

104.2.8 SPARC Implementor's ID

Frontgrade Gaisler is assigned number 15 (0xF) as SPARC implementor's identification. This value is hard-coded into bits 31:28 in the %psr register. The version number for LEON5 is 5, which is hard-coded in to bits 27:24 of the %psr.

104.2.9 Divide instructions

Full support for SPARC V8 divide instructions is provided (SDIV, UDIV, SDIVCC & UDIVCC). The divide instructions perform a 64-by-32 bit divide and produce a 32-bit result. Rounding and overflow detection is performed as defined in the SPARC V8 manual.

104.2.10 Multiply instructions

The LEON processor supports the SPARC integer multiply instructions UMUL, SMUL UMULCC and SMULCC. These instructions perform a 32x32-bit integer multiply, producing a 64-bit result. SMUL and SMULCC performs signed multiply while UMUL and UMULCC performs unsigned multiply. UMULCC and SMULCC also set the condition codes to reflect the result. The multiply instructions are performed using a 32x32 pipelined hardware multiplier.

104.2.11 Compare and Swap instruction (CASA)

LEON5 implements the SPARC V9 Compare and Swap Alternative (CASA) instruction. The CASA operates as described in the SPARC V9 manual. The instruction is privileged, except when setting ASI = 0xA (user data).

For compatibility with existing LEON3 and LEON4 software, when CASA is used with ASI 0xA, MMU translation and permissions checking will be done using the current priviledge level as determined by PSR.

104.2.12 Hardware breakpoints

The integer unit can be configured with up to four hardware breakpoints (two in the standard implementation). Each breakpoint consists of a pair of ancillary state registers (see section 104.12.5). Any binary aligned address range can be watched for instruction or data access, and on a breakpoint hit, trap 0x0B is generated.

104.2.13 Instruction trace buffer

The instruction trace buffer consists of a circular buffer that stores executed instructions. This is enabled and accessed only through the processor's debug port via the Debug Support Unit. When enabled, the following information is stored in real time, without affecting performance:

- Instruction address and opcode
- Instruction result
- Load/store data and address
- Trap information
- 30-bit time tag

Note that in multi-processor systems, each processor has its own trace buffer allowing simultaneous tracing of all instruction streams.

104.2.14 Processor configuration register

The ancillary state register 17 (%asr17) and the LEON5 configuration register provides information on how various configuration options were set during synthesis. This can be used to enhance the per-

formance of software, or to support enumeration in multi-processor systems. See section 104.12.5 for layout.

104.2.15Exceptions

LEON5 adheres to the general SPARC trap model. The table below shows the implemented traps and their individual priority. When PSR (processor status register) bit ET=0, an exception trap causes the processor to halt execution and enter error mode, and in the case of CPU0 the external error signal will then be asserted.

*Table 2025.*Trap allocation and priority

Trap	TT	Pri	Description	Class
reset	0x00	1	Power-on reset	Interrupting
cache_state_error	0x60	2	bad state detected in cache memories	Interrupting
data_store_error	0x2b	2	write buffer error during data store	Interrupting
instruction_access_exception	0x01	3	Error or MMU page fault during instruction fetch	Precise
privileged_instruction	0x03	4	Execution of privileged instruction in user mode	Precise
illegal_instruction	0x02	5	UNIMP or other un-implemented instruction	Precise
fp_disabled	0x04	6	FP instruction while FPU disabled	Precise
cp_disabled	0x24	6	CP instruction while Co-processor disabled	Precise
watchpoint_detected	0x0B	7	Hardware breakpoint match	Precise
window_overflow	0x05	8	SAVE into invalid window	Precise
window_underflow	0x06	8	RESTORE into invalid window	Precise
mem_address_not_aligned	0x07	10	Memory access to un-aligned address	Precise
fp_exception	0x08	11	FPU exception	Deferred
cp_exception	0x28	11	Co-processor exception	Deferred
data_access_exception	0x09	13	Access error during data load, MMU page fault	Precise
tag_overflow	0x0A	14	Tagged arithmetic overflow	Precise
division_by_zero	0x2A	15	Divide by zero	Precise
trap_instruction	0x80 - 0xFF	16	Software trap instruction (TA)	Precise
interrupt_level_15	0x1F	17	Asynchronous interrupt 15	Interrupting
interrupt_level_14	0x1E	18	Asynchronous interrupt 14	Interrupting
interrupt_level_13	0x1D	19	Asynchronous interrupt 13	Interrupting
interrupt_level_12	0x1C	20	Asynchronous interrupt 12	Interrupting
interrupt_level_11	0x1B	21	Asynchronous interrupt 11	Interrupting
interrupt_level_10	0x1A	22	Asynchronous interrupt 10	Interrupting
interrupt_level_9	0x19	23	Asynchronous interrupt 9	Interrupting
interrupt_level_8	0x18	24	Asynchronous interrupt 8	Interrupting
interrupt_level_7	0x17	25	Asynchronous interrupt 7	Interrupting
interrupt_level_6	0x16	26	Asynchronous interrupt 6	Interrupting
interrupt_level_5	0x15	27	Asynchronous interrupt 5	Interrupting
interrupt_level_4	0x14	28	Asynchronous interrupt 4	Interrupting
interrupt_level_3	0x13	29	Asynchronous interrupt 3	Interrupting
interrupt_level_2	0x12	30	Asynchronous interrupt 2	Interrupting
interrupt_level_1	0x11	31	Asynchronous interrupt 1	Interrupting

The prioritization follows the SPARC V8 manual.

The data_store_error is delivered as a deferred exception but is non-resumable and therefore classed as interrupting in above table. The traps related to cache/memory access are described further in the cache controller description.

104.2.16Address space identifiers (ASI)

In addition to the address, a SPARC processor also generates an 8-bit address space identifier (ASI), providing up to 256 separate, 32-bit address spaces. During normal operation, the LEON5 processor accesses instructions and data using ASI 0x8 - 0xB as defined in the SPARC manual. Using the LDA/STA instructions, alternative address spaces can be accessed. The different available ASIs are described in section 104.11.2.

104.2.17Partial WRPSR

Partial write %PSR (PWRPSR) is a SPARC V8e option that allows WRPSR instructions to only affect the %PSR.ET field. If the WRPSR instruction's rd field is non-zero, then this is interpreted as a PWRPSR instruction and the WRPSR write will only update ET. This allows for disabling traps with a single instruction. PWRPSR support is always enabled on LEON5.

104.2.18Power-down

The processor can be configured to include a power-down feature to minimize power consumption during idle periods. The power-down mode is entered by performing a WRASR instruction to %asr19:

```
wr %g0, %asr19
```

During power-down, the pipeline is halted until the next interrupt occurs. Signals inside the processor pipeline and caches are then static, reducing power consumption from dynamic switching.

Note: %asr19 must always be written with the data value zero to ensure compatiblity with future extensions.

Note: This instruction must be performed in supervisor mode with interrupts enabled.

When resuming from power-down, the pipeline will be re-started from the instruction following the power-down.

104.2.19Debug and control interface

The integer pipeline provides a command/status interface for starting / controlling and debugging the pipeline. An interface is provided to allow the debug module access the registers of the integer pipeline when it has been stopped for debugging. These features are described further in sections 104.9 and 104.10.

104.2.20Multi-processor systems

In multiprocessor systems, the ID of the processor on which the code is executing can be read out by reading the index field of the LEON4 configuration register. Only processor 0 starts executing after reset, the others are in power-down mode and are activated over the CPU-to-CPU interface or via the legacy interrupt controller interface. See section 104.9 for further details.

104.3 Cache system

104.3.1 Overview

The LEON5 processor pipeline internally implements a Harvard architecture with separate instruction and data buses, connected to two separate caches. As long as the execution does not cause a cache

miss, the cache controllers can serve one beat of an instruction fetch and one data load/store per cycle, keeping the pipeline running at full speed.

On cache miss, the cache controller will assert a hold signal freezing the IU pipeline, and after delivering the data the hold signal is again lifted so execution continues. The miss processing is handled by a state machine common to both the instruction and data caches.

Another important component included in the data cache is the write buffer, allowing stores to proceed in parallel to executing instructions.

Cachability (memory areas that are cachable) for both caches is controlled through the AHB plug&play address information or using a VHDL generic, see section 104.3.17.

104.3.2 Enabling and disabling cache

Both I and D caches are disabled after reset. They are enabled by writing to the cache control register (see 104.12.6). The cache tags are initialized automatically after reset to ensure that all tags are invalid.

104.3.3 Cache operation

Each cache controller has two main memory blocks, the tag memory and the data memory. At each address in the tag memory, a number of cache entries, ways, are stored for a certain set of possible memory addresses. The data memory stores the data for the corresponding ways.

For each way, the tag memory contains the following information:

- Valid bits saying if the entry contains valid data or is free. Both caches have a single valid bit for each cache line.
- The tag, all bits of the cached memory address that are not implied by the set

When a read from cache is performed, the tags and data for all cache ways of the corresponding set are read out in parallel, the tags and valid bits are compared to the desired address and the matching way is selected. In the hit case, this is all done in the same cycle to support the full execution rate of the processor.

In the miss case, the cache will at first deliver incorrect data. However on the following cycle, a hold signal will be asserted to stall the pipeline and preventing it from proceeding with that data. After the miss has been processed, the correct data is injected into the pipeline using a memory data strobe (mds) signal, and afterwards the hold signal can be released. If the missed address is cacheable, then the data read in from the cache miss will be stored into the cache, possibly replacing one of the existing ways.

The instruction cache implements a buffer forwarding scheme to reduce instruction cache miss latency. While the cache line is being read in from the AHB bus and stored into the cache memories, the data is stored temporarily in a flip flop buffer inside the cache controller. The cache can serve instructions to the pipeline directly out of this partially filled buffer, this works even in case of branches as long as the instruction flow stays on the same cache line.

104.3.4 Cache configuration

The cache controller supports different size configurations, selected by the *perfcfg* generic.

Table 2026. Supported cache configurations

perfcfg value	I-cache size (ways x waysize)	I-cache line size	D-cache size (ways x waysize)	D-cache line size	ITLB entries	DTLB entries
0	4 x 4 KiB	32 bytes	4 x 4 KiB	32 bytes	24	24
1	4 x 4 KiB	32 bytes	4 x 4 KiB	32 bytes	16	16
2	1 x 4 KiB	32 bytes	1 x 4 KiB	32 bytes	4	4

The cache controller implements a 32 byte cache line length.

Least-recently-used (LRU) replacement policy is used for the caches. This maintains the order of usage for each set in the cache and replaces the one which has was used last. The LRU information needs to be updated on every cache hit and is therefore not stored with the tags but in separate flip flops.

104.3.5 Address mapping

The addresses seen by the CPU are divided into tag, index and offset bits. The index is used to select the set in the cache, therefore only a limited number of cache lines with the same index part can be stored at one time in the cache. The tag is stored in the cache and compared upon read.

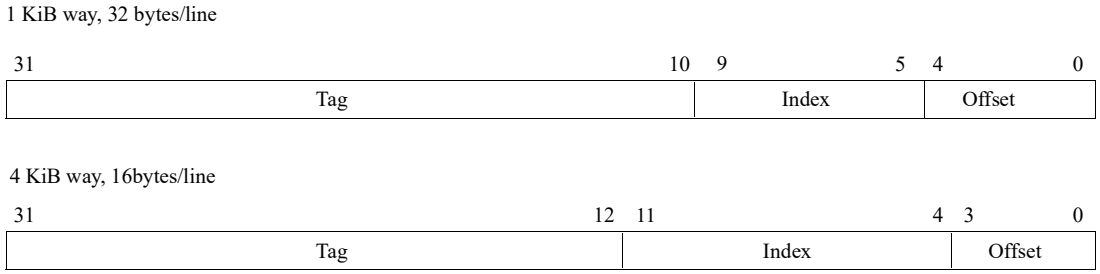


Figure 276. Cache address mapping examples

104.3.6 Data cache policy

The data cache employs a write-through policy, meaning that every store made on the CPU will propagate, via the write buffer, to the bus and there are no “dirty” lines in the cache that has not yet been written out apart from what is in the buffer. The store will also update the cache if the address is present, however a new line will not be allocated in that case.

*Table 2027.*LEON5 Data caching behavior

Operation	In cache	Cacheable	Bus action	Cache action	Load data
Data load	No	No	Read	No change	Bus
	No	Yes	Read	Line allocated/replaced	Bus
	Yes	-	None	No change	Cache
Data load with forced cache miss (ASI 1)	No	No	Read	No change	Bus
	No	Yes	Read	Line allocated/replaced	Bus
	Yes	-	Read	Data updated	Bus
Data load with MMU bypass (ASI 0x1C)	-	-	Read (phys addr)	No change	Bus
Data store	No	No	Write (via buffer)	No change	(N/A)
	No	Yes	Write (via buffer)	No change	(N/A)
	Yes	-	Write (via buffer)	Data updated	(N/A)
Data store with MMU bypass (ASI 0x1C)	-	-	Write (via buffer, phys addr)	No change [1]	(N/A)

[1] If snooping is enabled, the snooping logic will see the write on the AHB bus and invalidate any data cache lines to the same address.

104.3.7 Write buffer

The data cache contains a write buffer able to hold up to four 8,16,32, or 64-bit writes. For half-word or byte stores, the stored data replicated into proper byte alignment for writing to a word-addressed device.

The write is processed in the background so the system can keep executing while the write is being processed. The write buffer acts as a FIFO where new stores can be added continuously to allow one store per cycle throughput. Only “regular” stores that hit in the TLB are handled through the write buffer, other types of writes (for example special register access via ASI) will be handled separately.

While a store is ongoing, any following instruction or data access that requires bus access will block until the write buffer has been emptied. Loads served from cache will however not block, due to the cache policy used there can not be a mismatch between cache data and store buffer (the effect of this behavior on SMP systems is discussed in section 104.3.16).

Since the processor executes in parallel with the write buffer, a write error will not cause an exception to the store instruction immediately but may deliver an exception at a later point in time when the error is detected. How write errors are handled is discussed in more detail in section 104.3.21.

104.3.8 Operating with MMU

The MMU functionality is integrated into the cache controller. When MMU is enabled, the virtual addresses seen by the running code no longer correspond directly to the physical addresses on the AHB bus. The cache will translate each accesses virtual addresses to its physical addresses in parallel with looking up the tag for that access.

The cache uses a virtually-indexed-physically-tagged (VIPT) approach, where the tags are based on the physical addresses but the address into the cache memory is based on the virtual address. Every virtual address is translated on the fly to physical address prior to the tag comparison. If a virtual address is not present in the translation lookaside buffer, the cache will stall the pipeline to translate the addres, and then re-do the tag comparison once the physical address is known

104.3.9 Snooping

The data cache includes support for AHB bus snooping. The AHB bus the processor is connected to, is monitored for writes from other masters to an address which is in the cache. If a write is done to a cached address, that cache line is marked invalid and the processor will be forced to fetch the (new) data from memory the next time it is read.

The snooping logic needs to be able to access the data cache tags whenever a store happens which might be at the same time as a normal loads or store needs to access the tags. This is accomplished either by adding an extra snoop tag memory or by making the data cache tag memory dual ported. The snoop logic also needs to be able to clear valid bits on a snoop hit which is handled either by having a two-ported memory for the regular data cache tags or by putting the valid bits into flip flops. The sub-system allows for some different implementations via the cmemconf generic.

The processor normally avoids snooping on its own writes (as otherwise every store would lead to invalidating its own cache line) but does in some cases snoop on itself in order to detect MMU updating page table entries. Writes done by storing to the MMU/cache bypass ASI are also snooped.

Snooping requires the way size of the data cache to be equal or smaller than the MMU page size, otherwise the index into the physical and virtual tag RAM:s may not match, resulting in aliasing problems. All the standard configurations of the LEON5 subsystems satisfy this requirement.

104.3.10Cache freeze

Each cache can be in one of three modes: disabled, enabled and frozen. If disabled, no cache operation is performed and load and store requests are passed directly to the memory controller. If enabled, the

cache operates as described above. In the frozen state, the cache is accessed and kept in sync with the main memory as if it was enabled, but no new lines are allocated on read misses.

If the DF or IF bit is set, the corresponding cache will be frozen when an asynchronous interrupt is taken. This can be beneficial in real-time system to allow a more accurate calculation of worst-case execution time for a code segment. The execution of the interrupt handler will not evict any cache lines and when control is returned to the interrupted task, the cache state is identical to what it was before the interrupt. If a cache has been frozen by an interrupt, it can only be enabled again by enabling it in the CCR. This is typically done at the end of the interrupt handler before control is returned to the interrupted task.

104.3.11Flushing

Both instruction and data cache are flushed either by executing the FLUSH instruction, setting the FI/FD bits in the cache control register, or by writing to certain ASI address spaces.

Cache flushing takes one clock cycle per cache set, during which the IU will be halted. When the flush operation is completed, the cache will resume the state (disabled, enabled or frozen) indicated in the cache control register.

Note that while the SPARC V8 specifies only that the instructions pointed to by the FLUSH argument will be flushed, the LEON5 has a special case for LEON3/LEON4 compatibility, where it will flush the entire I and D cache if the address given is zero (which is permitted by the manual as the additional flushing only affects performance and not operation).

104.3.12Region flush

The cache controller supports a region flush function where the cache tags are scanned and cache lines matching a specified physical address and mask are invalidated. Note that depending on how the address mask is set, the flushed will require scanning a different amount of cache tags and it will take different amounts of time to complete the flush. The region flush function will block the pipeline until the flush has been completed.

104.3.13Write combining

The cache controller has an optional write combining feature, to automatically merge 64-bit stores to adjacent addresses into larger bursts, up to a full cache line’s worth of data.

The write combining can improve performance in some cases. A system-level example of such a case is when writing out a large buffer to memory through a level-2 cache that does not have individual dirty/valid tracking bits for every 64-bit word. With individual stores, the first 64-bit store of each cache line would lead to an unnecessary fetching in from memory of the remaining cache line data that will immediately just be overwritten by the following stores. When combined into a burst, the cache can see that the entire cache line is written and simply allocate the line, write in the data, and mark the entire cache line as dirty.

The feature can however come with some performance cost in other cases. Since the cache controller can not know if a double store will be followed by another store in the executing program, the store needs to be held by the cache controller for a few cycles to see if another store to the following address is coming after it. This added store latency can result in reduced overall performance.

The write combining can be set into an automatic hold-off mode where double stores are automatically held for a few cycles to see if another store comes in. It can also be set into a manual mode where the write combining is off in the normal case but a write combining hint will activate the write combining for the following store coming in.

104.3.14Memory barrier

In virtually all normal situations, explicit barriers are not needed on SPARC as the processor follows the total store order model and load-to-load, store-to-store and load-to-store ordering is implicitly guaranteed for all operations.

An explicit barrier instruction is provided for any special cases where store-to-load ordering is required or where lookaside reads must be avoided. The barrier is created using the STBAR instruction defined in SPARC V8, but setting certain otherwise unused bits. This is aligned with how a the corresponding memory barrier MEMBAR is defined in SPARC V9.

*Table 2028.*LEON5 memory barrier operation

31 30	29 25	24 19	18 14	13	12 9	8	7	6	5	4	3 2	1	0
op=10	rd=00000	op3=101000	01111	0	unused	WCH	un-used	Sync	MIs	LA	unused	SL	unused

8	Write combining hint for following store
7	Reserved
6	‘Synchronization barrier
5	Memory issue barrier
4	Lookaside barrier
3 : 2	Ignored (used for StoreStore and LoadStore in V9)
1	StoreLoad barrier
0	Ignored (used for LoadLoad barrier in V9)

The barrier operations are in practice implemented the same on LEON, as they will cause the processor to stall the pipeline until current stores in the store buffer have completed and passed through the snoop pipeline. In the future, the StoreLoad barrier may be optimized further as that does not need to block in all cases.

The STBAR instruction is also used to encode a write combining hint, using a normally unused bit 8. The write combining feature is explained further in section 104.3.13.

104.3.15Diagnostic access

The cache tag and data contents can be directly accessed for diagnostics via various ASI:s, see section 104.11.2 for ASI assignments.

When accessing the diagnostic ASIs, the address should be formed in the same way as a regular virtual address as shown in figure 276. The part of the address that would normally be used to index the cache is used as the index. The lowest two bits of the tag part are used to select which way to access. The remaining parts of the tag are ignored and recommended to clear for forward compatibility. The offset part of the address is used as offset for data accesses, but ignored for tag accesses. For a 4 KiB way cache with 32 byte cache lines, that means address bits 11:5 are used as the index and address bits 13:12 are used to select the cache way.

The data read and written also follows the address format in figure 276, except that in this case it is the tag part of the address that has the actual contents of the cache and the index part is ignored. When reading/writing tags, the valid bit is put on bit zero of the data. For software compatibility with earlier LEON models, the valid bit is replicated across all data bits 7:0. In configurations where the valid bit is implemented in flip flops, the valid bits are still written via the diagnostic ASI for data cache tags which makes the difference transparent to software.

Diagnostic access to the snoop tags, as well as writes into the regular data cache tags, are made in a way that allows to do these accesses when there are potential snoop traffic ongoing on the AHB bus in parallel. If the diagnostic access is made on the same cycle as a snoop lookup (corresponding to the address phase of a write access being observed on the AHB bus), the diagnostic access is stalled until the following cycle. In order to avoid a (very unlikely) livelock situation where the diagnostic access stalls forever due to back-to-back stores from the other masters, the cache controller implements a

timeout. If the diagnostic access is stalled for more than 32 cycles, the cache controller raises the bus request signal on the processor bus to produce an idle cycle on the bus, leading to . This functionality is transparent to software.

104.3.16AMBA interface

The LEON5 cache controller has one AHB master interface to the surrounding system. The types of AMBA accesses supported and performed by the processor depend on the accessed memory area's cachability, the maximum bus width, if the corresponding cache is enabled, and if the accessed memory area has been marked as supporting wide bus accesses.

Cacheable instructions are fetched with a burst of 32-bit accesses, or 64- or 128-bit accesses depending on the cache line size and the AHB access size supported by the accessed slave.

The HPROT signals of the AHB bus are driven to indicate if the accesses is instruction or data, and if it is a user or supervisor access.

*Table 2029.*HPROT values

Type of access	User/Super	HPROT
Instruction	User	1100
Instruction	Super	1110
Data	User	1101
Data	Super	1111
MMU	Any	1101

In case of atomic accesses, a locked access will be made on the AMBA bus to guarantee atomicity as seen from other masters on the bus.

104.3.17Cachability

Cachability for both caches can be controlled through the AHB plug&play address information or set manually via the *cached* VHDL generic.

For plug'n'play based cachability, the memory mapping for each AHB slave indicates whether the area is cachable, and this information is used to (statically) determine which access will be treated as cacheable. This approach means that the cachability mapping is always coherent with the current AHB configuration.

When the *cached* VHDL generic is not zero, it is treated as a 16-bit field, defining the cachability of each 256 MiB address block on the AMBA bus. For example, a value of 16#00F3# will define cachable areas in 0 - 0x20000000 and 0x40000000 - 0x80000000.

In order to access the plug'n'play information, the processor takes the ahbso vector as input. Only the static hconfig signals are used so the use of this input will be eliminated through constant propagation during synthesis.

Note that when the MMU is enabled, cacheability is configured explicitly by the user via the page tables (C bit of the Page Table Entry) which will override the mechanisms described in this section. It is on the user to ensure that accesses that need to be as uncached single accesses (such as configuration registers) are done through an address mapping set as uncached.

104.3.18Wide bus support

The cache controller's AHB master can be configured to support wide (64 or 128-bit) AHB bus accesses in accordance with the wide-bus support built into GRLIB (see GRLIB User's Manual). The wide bus support has been designed to allow backward compatibility where existing non-wide capable (32-bit) cores can coexist on the same AHB bus. The wide-bus operation is optional and the cache

controller can also be configured for only 8/16/32-bit AHB access for compatibility with existing LEON3 systems.

Because wide bus capability is not managed automatically via the AMBA plug'n'play infrastructure, information on wide bus capability has to be provided manually into the processor system through two generics, busw and wbmask. These must be correctly configured by the system designer in order to avoid illegal AMBA access combinations.

The area which supports wide access is indicated in the *wbmask* VHDL generic. This VHDL generic is treated as a 16-bit field, defining the 64/128-bit capability of each 256 MiB address block on the AMBA bus. A value of 16#00F3# will thus define areas in 0 - 0x20000000 and 0x40000000 - 0x80000000 to be 64/128-bit capable. The maximum access size to be used in the area(s) marked with *wbmask* is determined by the *busw* VHDL generic.

In a system with only 32-bit wide masters and slaves, the *busw* should be set to 32. All AHB accesses will then be performed using 32-bit bursts or 8/16/32-bit single accesses. The *wbmask* option is ignored in this configuration

104.3.19AMBA access types

Cacheable data is fetched in a burst of 32-bit, 64-bit, or 128-bit accesses, depending on the cache line size, AHB bus width, and whether the access is to a narrow or wide area (see section 104.3.18).

Uncacheable data is fetched using a single access with the same size as the AHB bus for a wide area, or 32-bit width in case of access to a narrow area. A 64-bit uncached load from a 32-bit AHB bus or a narrow area will be performed using two single 32-bit reads.

Store data is performed using the same access size as the store whenever possible. In the case of a 64-bit store on a 32-bit AHB bus or to a narrow area, the store will be performed using a two-beat burst.

Instruction fetches are performed in the same way as cacheable data. Instructions are always fetched in full cache lines regardless of whether instruction cache is enabled or not.

104.3.20Read error handling

An AHB ERROR response received while fetching instructions will cause the cache controller to assert an internal error signal into the pipeline when delivering the remaining instructions on that cache line, and the valid bit written into the cache tag will be set to zero to force a re-fetch on the next access. The internal error signal will cause an instruction access exception trap (tt=0x1) when the delivered instructions reach the exception stage. However because there are cases where the instructions fetched do not get executed in the processor pipeline (due to traps, branches etc), there is no guarantee that a trap will always occur. If the IU later needs to execute the instruction, as it will be fetched again, a cache miss will occur, triggering a new access to the failed address. Instruction access exception traps that are "lost" in this way are however logged in a status bit in the trap register and the AHB error is still captured by the general AHB error logging of the cache controller.

An AHB ERROR response while fetching data into the data cache will signal an error to the pipeline that triggers a data_access_exception trap (tt=0x9) on the load instruction.

An ERROR response during an MMU table walk will lead the MMU to set the fault type to Internal error (1) and generate an instruction or data access exception, depending on which type of access that caused the table walk.

104.3.21Write error handling

Error responses on the AHB bus for writes will cause a data_store_error trap (tt=0x2b) to be delivered to the running application. Because stores are buffered in the cache controller, the store instruction that caused the write has already retired and the data_store_error trap is instead delivered to an instruction later on in the execution flow that may be unrelated to the failing store.

If a store that causes an error response is issued in the application while traps are enabled but an interrupt or other unrelated trap happens before the data_store_error trap is delivered, this could lead to the data_store_error trap being delivered while ET=0 which puts the processor in error mode. The cache controller can be configured into a few different modes via the AHB error register (section 104.12.10) in order to configure how such cases should be handled:

- In direct mode, the data store error trap is always delivered immediately. and will trigger error mode if ET=0.
- In deferred mode, store errors for stores made while ET=1 are only delivered if ET=1 and there are no intervening traps. If any traps occur then the error is “lost”. A specific register field in the cache trap register indicates if any such lost traps have occurred. Errors for stores made while ET=0 will still be delivered immediately and cause error mode.
- In interrupting mode, data_store_error traps are only delivered while ET=1 and can therefore never trigger error mode. Store errors occurring while ET=0 will be left pending in the cache controller, and gets delivered first when traps are enabled.
- In disabled mode, the store error never leads to any trap being delivered.

For backward compatibility with earlier generations of LEON processor, the store error trap delivery can also be blocked (regardless of mode setting used above) using the DWT bit in ASR17.

A write error can also happen when the MMU writes back a page table entry in order to set the referenced or modified bit. These write errors are also delievered as data_store_error traps, in the same way as for stores, and the delivery can be configured in the same way as described above. A status bit exists in the trap register that allows the data_store_error trap handler to determine whether the error was caused by a store instruction or an MMU writeback. The MMU write-back case is very rarely triggered since the MMU already successfully read from the same address location.

104.3.22Error logging

The cache controller monitors the processor bus if any AHB errors occur and logs them into the AHB error register. Both errors for accesses generated by other masters and by the cache controller itself are logged but an error from the processor itself will have priority and overwrite an earlier logged error from another master.

104.3.23Read error masking

AHB errors can be masked from causing data_access_exception traps when loading data by setting the MDAE bit in the AHB error register. If this bit is set, execution will continue as if the error had not happened and the data_access_exception trap will not be generated. AHB errors during instruction fetch and MMU accesses will still generate traps as usual. Errors will still be logged into the AHB error register, and this can be checked after with errors ignored to see if any error occurred.

When the ignore errors mode is enabled, the cache controller can be configured to either pass on the contents of the read data bus at the time of the error, or replace the data with all-ones or all-zeros. If the memory read is cacheable, the replaced data from the error will also be written into cache, so a cache flush may be needed if this is a concern on system level.

There is a similar AHB error masking capability for instructions, enabled using the MIAE bit in the AHB error register. The part of the cache line that could not be fetched due to error will be replaced by an all-zero value, representing an UNIMP instruction, and if execution reaches that instruction it will trigger an unimplemented_instruction trap. However if execution never reaches that instruction, the error will not impact execution and it allows the other part of the cache line can be cached.

104.3.24Internal error trap

The cache controller can signal certain internal errors to the pipeline using an implementation-defined trap (tt=0x60). The reason for the trap is encoded in the trap register. This trap would be an indication

of corrupted state inside the cache memories and normally could be considered a fatal error as incorrect instructions/data may have been delivered to the application at that point.

104.3.25Tightly coupled memory

The cache controller provides an option to add tightly coupled memory (TCM) to the instruction and/or data cache. The TCM is a local memory that can be accessed with the same speed as cached data, but is not backed by external memory. The instruction TCM can only hold code to execute and can not be accessed using regular load/store instructions, while the data TCM on the other hand can only hold data to access using load/store instructions and can not be executed from.

The address of the TCM can be configured at run-time through a configuration register. The TCM can also be disabled so that it is not visible at any memory address, however the contents can always be accessed through dedicated ASIs.

When using MMU, the TCM is mapped directly to a specific virtual address and will appear on the same address regardless of page table contents. The TCM can be set to be visible only in a single MMU context, or in all MMU contexts.

The TCM size can be any power of 2, and the size determines how many bits get implemented in the registers for configuring the TCM's location. For finer control of implementation memory usage, the TCM can be built with its memory space only partially filled (between 9/16 and 15/16 of the size, in steps of 1/16). In partially filled configuration, the unused part of the TCM memory space will return undefined data on read and ignore any writes. The size and presence of TCM can be probed through the LEON5's configuration registers.

The TCM memory contents are not automatically cleared on reset, but can be cleared by setting the wipe TCM bits in the TCM configuration register. The wipe function is non-blocking so execution will continue immediately in parallel with the wipe, however the TCM memory will be inaccessible until the wipe has completed. To see when the wipe has completed, the wipe TCM bits can be polled to see when they have returned to zero. Before a TCM wipe can be started, all previously started TCM wipes will need to have competed. The wipe will initialize the data TCM to zero and the instruction TCM to all-ones (UNIMP instruction opcode).

The TCM is configurable with permission bits to separately allow or deny supervisor read, supervisor write, user read, and user write for data TCM, and supervisor execute and user execute for instruction TCM.

104.4 Memory management unit

104.4.1 Overview

The memory management unit is integrated into the cache controller. This is compatible with the SPARC V8 reference MMU (SRMMU) architecture described in the SPARC V8 manual, appendix H.

The MMU provides address translation of both instructions and data via page tables stored in memory. When needed, the MMU will automatically access the page tables to calculate the correct physical address. The latest translations are stored in a special cache called the translation lookaside buffer (TLB), also referred to as Page Descriptor Cache (PDC) in the SRMMU specification. The MMU also provides access control, making it possible to "sandbox" unpriviledged code from accessing the rest of the system.

104.4.2 MMU/Cache operation

When the MMU is disabled, the MMU is bypassed and the caches operate with physical address mapping. When the MMU is enabled, the cache tags store the physical address and each access from the pipeline is translated from virtual to physical in parallel with the tag memory read-out.

No additional clock cycles are needed for the MMU for instruction fetch, data fetch, or stores, as long as the virtual pages accessed are already cached inside the TLB. If there is a TLB miss the page table must be traversed, resulting in up to four AMBA read accesses and one possible writeback operation. See the SRMMU specification for the exact format of the page table.

An MMU page fault will generate trap 0x09 for the D-cache and trap 0x01 for the I cache, and update the MMU status registers according to table 2030 and the SRMMU specification. In case of multiple errors, they fault type values are prioritized as the SRMMU specification requires. The cache and memory will not be modified on an MMU page fault.

*Table 2030.*LEON5 MMU Fault Status Register, fault type values

Fault type	SPARC V8 ref	Priority	Condition
6	Internal error	1	Never issued by LEON SRMMU
4	Translation error	2	AHB error response while performing table walk. Translations errors as defined in SPARC V8 manual. A translation error caused by an AMBA ERROR response will overwrite all other errors. Other translation errors do no overwrite existing translation errors when FAV = 1.
1	Invalid address error	3	Page table entry for address was marked invalid
3	Privilege violation error	4	Access denied based on page table and su status (see SRMMU spec for how privilege and protection error are prioritized)
2	Protection error	5	
0	None	-	No error (inside trap this means the trap occurred when fetching the actual data)

104.4.3 Translation look-aside buffer (TLB)

Separate TLBs are implemented for instruction and data, the number of entries depending on configuration as listed in table 2026.

The TLBs are replaced using a pseudo-LRU algorithm. Each TLB entry has a single ‘visited’ bit that is set when that TLB entry is accessed, when all visited bits are set to 1 then they are all cleared and the process starts over. When replacing an entry in the TLB, the lowest entry with the visited bit not set will be chosen.

104.4.4 MMU flush and probe

A flush and probe interface is provided for the MMU mapped to ASI 0x1B.

A 32-bit read from the ASI triggers a probe that works as described in section H-3 of the SPARC V8 specification. If the type of the probe is “entire” (type 4), the ITLB and DTLB are checked for a match before falling back to page table walk, the other types always lead to table walk and do not check the TLB:s. The probe does not cause an update to the TLB contents.

A 32-bit write to the ASI triggers a flush function that works as described in section H-3 of the SPARC V8 specification. The flush done is precise as defined in the specification.

104.4.5 MMU diagnostic access

The TLB contents and pseudo-LRU state can be read out via a diagnostic ASI. Each ITLB and DTLB entires is stored in two 32-bit words that are as shown in tables 2031 and below.

*Table 2031.*LEON5 diagnostic TLB entry format, first word (offset 0)

31 … 16	15 … 12	11 … 4	3	2	1	0
VADDR(31:16)	VADDR(15:12)	CONTEXT	L1	L2	L3	VALID

31:12	Virtual address of mapping. Depending on mapping size, the lower bits of the address should be zero.
11:4	MMU context of mapping..
3	Set to 1 if the mapping is a level 1, 2 or 3 mapping and virtual address 31:24 must be identical for an access to hit this TLB entry.
2	Set to 1 if the mapping is a level 2 or 3 mapping and virtual address 23:18 must be identical for an access to hit this TLB entry.
1	Set to 1 if the mapping is a level 3 mapping and virtual address 17:12 must be identical for a n access to hit this TLB entry.
0	Set to 1 if this TLB entry is valid.

*Table 2032.*LEON5 diagnostic TLB entry format, second word (offset 4)

31 … 16	15 … 12	11 … 6	5 … 3	2	1	0
PADDR(31:16)	PADDR(15:12)	RESERVED	ACC	BUSW	C	MOD

31:12	Physical address of mapping. Depending on mapping size, the lower bits of the address should be zero.
11:6	Reserved for future use
5 :3	Access settings, copy of ACC field of PTE
2	Set to 1 if the physical address is on a wide AHB capable address range.
1	Cacheability setting, 1 if accesses are cacheable
0	Cached state of modified bit of PTE

104.5 Floating-point unit

The SPARC V8 architecture defines two (optional) co-processors: one floating-point unit (FPU) and one user-defined co-processor. Two different FPU's can be interfaced the LEON5 pipeline: Frontgrade Gaisler's GRFPU5 and NanoFPU. The characteristics of the FPU's are described in the next sections.

104.5.1 Frontgrade Gaisler's floating-point unit (GRFPU5)

The high-performance GRFPU5 operates on single- and double-precision operands, and implements all SPARC V8 FPU operations including square root and division. The FPU is interfaced to the LEON5 pipeline using a LEON5-specific FPU controller (GRFPC5) that allows FPU instructions to be executed simultaneously with integer instructions. Only in case of a data or resource dependency is the integer pipeline held. The GRFPU5 is fully pipelined and allows the start of one instruction each clock cycle, with the exception is FDIV and FSQRT which can only be executed one at a time. The FDIV and FSQRT are however executed in a separate divide unit and do not block the FPU from performing all other operations in parallel.

All instructions except FDIV and FSQRT has a latency of four cycles, but to improve timing, the LEON5 FPU controller inserts an extra pipeline stage in the result forwarding path. This results in a

latency of five clock cycles at instruction level. The table below shows the GRFPU instruction timing when used together with GRFPC:

*Table 2033.*GRFPU5 instruction timing with GRFPC5

Instruction	Throughput	Latency
FADDS, FADDD, FSUBS, FSUBD,FMULS, FMULD, FSMULD, FITOS, FITOD, FSTOI, FDTOI, FSTOD, FDTOS, FCMPS, FCMPD, FCMPES. FCMPED	1	5
FCMPS, FCMPD, FCMPES, FCMPED	1	3
FDIVS	16	17
FDIVD	16.5 (15/18)	17.5 (16/19)
FSQRTS	24	25
FSQRTD	24.5 (23/26)	25.5 (24/27)

The GRFPC5 controller implements the SPARC deferred trap model, and the FPU trap queue (FQ) can contain up to 8 queued instructions when an FPU exception is taken. When the GRFPU5 is enabled in the model, the version field in %fsr has the value of 4.

The GRFPU5 can handle the full IEEE-754 standard including denormalized numbers as inputs and outputs. Denormalized inputs will make the operation take one extra cycle for MUL, DIV, and SQRT, but has zero penalty for other operations. Denormalized outputs have zero penalty compared to normal outputs.

The GRFPU5 can also be configured to trigger an fp_exception with the FPU trap type set to unfinised_FPOP (tt=2) when there is an input or output with a denormalized number. It can also be programmed to flush denormalized numbers to 0.

104.5.2 NanoFPU

The NanoFPU is a tiny combined FPC and FPU implementation using less than 1500 register bits (including 512 bits for the register file) and a single 16x16 multiplier that can be mapped into a DSP block on most FPGAs.

The NanoFPU performs one operation at a time in a sequential fashion. The operation is performed in parallel with the integer pipeline but any following FPU operation or FPU load/store will stall the pipeline until the operation has completed.

Traps are delivered using the deferred trap model, but with the deferred floating-point queue containing only a single operation.

When the NanoFPU is enabled in the model, the version field in %fsr has the value of 5.

104.5.3 Floating-point state access

The LEON5 provides an interface that allows the CPU or debugger to read/write the state of the floating-point controller. It is accessed through ASI 0x20.

Floating point registers can be read and written through the ASI interface, which saves having to go through memory. The ASI interface supports 64-bit loads and stores from the processor using LDDA/STDA instructions.

The current exception pending state including deferred floating point queue contents can be read out, and to restore this state at a later stage.

104.5.4 Co-processor interface

No implementation for the user-defined co-processor is currently provided.

104.6 Fault tolerance features

104.6.1 Overview

Note that the features described in this section are only available in FT and FT-FPGA releases of GRLIB.

The LEON5 processor can be built with functionality to handle bit errors occurring in the SRAM memories included in the processor. The intent with this functionality is to handle random single-event upsets in the memories. Errors encountered are corrected automatically without any visible side effects to software except for a small delay. A scrubber function is included to prevent error build-up.

Two different modes of memory protection are supported, one intended for FPGA usage and one for ASIC and generic usage. The FPGA native ECC uses error detection and correction functionality already built into the FPGA memory blocks, while the generic version intended for ASIC implements error detection and correction codes entirely in RTL. The generic version is also usable for FPGAs lacking built-in ECC and for FPGA prototyping of ASIC designs.

104.6.2 Cache memory protection

The instruction cache tags, instruction cache data, data cache tags, data cache data, and tightly coupled memories (if enabled) can be protected with error correcting codes using either an RTL EDAC or using FPGA native error correction.

When the RTL/ASIC ECC is enabled, an error correcting code is used to protect the memories that has been designed for low impact on maximum frequency and capability to detect many types of multi-bit upset (the code is described further in section 104.6.8). The error detection part of the ECC decoding is performed in parallel with the cache tag/data lookup in order to allow execution to proceed without any performance overhead in the error-free case.

Detected errors during cache operation (tag/data lookups) are handled in the controller by entering a “correction cycle” where the location with the error is read out, corrected and written back while the processor pipeline is held. After the correction cycle, the requested cache lookup is performed again, and depending on cache hit/miss result, either execution continues or the cache miss is processed as usual. From the point of view of the processor pipeline, the cache controller behavior during the correction cycle is exactly the same as for a normal cache miss, but as the correction cycle is done internally in the tag memories there is no access to the processor bus.

The default behavior on multi-bit errors that are not correctable by the ECC, is to invalidate the cache line with the tag and re-fetch the data from memory. In the event of a multi-bit error in a cache tag, all tags belonging to the same set in the cache are invalidated. Note that because the cache is write-through, the latest contents of any memory location is always available in external memory. With this approach, errors in the cache that are uncorrectable from a local ECC point of view are correctable on system level. This approach is similar to how earlier generation of LEON processors (that only had error detection capability) handle errors in the cache.

The processor can be configured to other behaviors in the event of detecting multi-bit errors. It can be set to flush the entire cache where the error was detected. It can also be set to generate an internal error trap.

For the AHB bus snooping functionality, when ECC is enabled, then the snoop tags are checked for errors and corrected on the fly before they are compared with the address of the AHB store. An extra pipeline stage is added to the snoop pipeline to allow for this error correction without impacting maximum clock frequency. After a snoop tag correction, the corrected snoop tags are stored in a register to be written back into the snoop tag memory at the earliest point where the snoop pipeline is free. Uncorrectable errors in the snoop tag are handled by invalidating all ways of the corresponding cache set. The behavior of the snoop tag error correction is not affected by the configuration registers and errors in the snoop tags do not trigger traps.

The FPGA native ECC mode by default operates the same way as described above, with correction cycles when an error is detected. Because the FPGA built-in ECC produces the corrected data immediately on the same cycle as the memory is read, an additional error handling option is available where correction cycles are not performed but the data is simply taken on the fly and the error is left to be corrected later by the scrubber.

104.6.3 Tightly coupled memory protection

The tightly coupled memory is protected in the same way as described for the cache memory.

Unlike cache memory, the tightly coupled memory does not have a copy of its contents in external memory and errors that are not correctable by the ECC are therefore not possible to mitigate automatically. When uncorrectable errors occur in the instruction TCM, the cache controller replaces the contents with UNIMP instructions which will trigger a trap if the instructions are executed. For uncorrectable errors in the data TCM, the processor can be configured to either deliver all-0 or all-1 data, or to trigger an internal error trap.

104.6.4 Integer register file protection

When the integer register file used for the general purpose registers (globals and register windows) is implemented with SRAM memories, the processor provides options for ECC protection. As in the cache case, there is an RTL variant and a native FPGA variant.

Any errors in the register file that are detected during operation are corrected and written back with an automatic pipeline restart of the instruction that needed the register. This processing is transparent to software.

The RTL/ASIC ECC protection adds parity on the registers which allows bit errors to be detected. Because the register file is made up of four two-port memories holding the same data (in order to provide the four read ports), correction can be performed by getting the register contents from another port that is backed by a different memory. In order to get the correct data, the processor pipeline performs a correction where the register that had the detected error is read from all memories and the first copy found without a parity error is taken as the correct value and written into all memories. In parallel with this correction, the instruction where the error occurred is restarted in the pipeline and gets re-executed. In the unlikely case that all the five copies of the register have a parity error, the instruction will trap with r_register_access_error trap (tt=0x20).

When using FPGA native error correction, correctable errors can be configured to either be corrected with a restart as for the RTL correction, or for the corrected values to be taken on the fly without a pipeline restart. Uncorrectable errors are handled using a pipeline restart and fetch from a redundant copy in the same way as a parity error is handled in the RTL-based case.

When using the flip flop register file, protection options are not available on the IP core level (the flip flops are assumed to be hardened or protection applied through other means). The same applies for the floating-point register file is always in flip flops in LEON5.

104.6.5 Hardware scrubber

When the processor has been configured with some form of error correction (either cache memory, TCM or integer register file protection), a hardware scrubber function is also included. The scrubber can be enabled or disabled and the scrub rate is configurable through a configuration register. The intention of the scrubber is to prevent build-up of errors in parts of the cache or register file that are not accessed or replaced regularly.

Each iteration of the scrubber, one location of every tag and data memory in parallel, as well as one register of the integer register file, are read to check for errors and written back in the event of a detected error. The processor pipeline is stalled during the scrubber iteration.

Under normal conditions, a scrub cycle stalls the processor for 7 cycles. The time required may be extended slightly if there is snoop activity on the AHB bus since the access to the snoop tag RAM has

to be arbitrated. This is handled in a way similar to diagnostic accesses to the snoop tags as described in section 104.3.15.

For a cache with 4 ways, 4 KiB/way and 32 bytes cache line, it will take 128 scrubber iterations to loop through the tag memories and 512 iterations to loop through the data memories. The time it takes to loop through the TCM depends on selected size, as 64 bits of data can be scrubbed per iteration. In a standard 8-register window configuration, it takes 256 scrubber iterations to loop through all integer registers.

The interval to set for the scrubber should be determined based on the necessary scrub rate needed to avoid error buildup, which would come from a system level analysis. If, as an example, a 100 Hz scrub rate is required for all the memories inside the processor and the processor is running at 500 MHz, the scrub period would have to be set to (10000 / 512) * 500 = 9765 cycles.

Setting very low scrubber periods (below the order 50 cycles) is not recommended as it will slow down execution significantly, and very low periods can make the execution to almost appear stalled.

Error in cache memories and tightly coupled memories that are not correctable by the ECC are handled in the same way as when they are discovered functionally as described in sections 104.6.2 and 104.6.3. Uncorrectable errors in the register file (parity or uncorrectable error in all copies) are handled by leaving the error in the register file to be discovered later by a functional access.

104.6.6 Error counting and reporting

The processor contains two bit wide error counters for correctable errors and one bit wide counters for uncorrectable errors. The counters are saturating in order to avoid accidentally wrapping around to zero. The counters are not reset in order to allow for post-mortem analysis following a crash and a reset. They will therefore contain random values on power up and need to be read out and cleared manually by software before they are used.

In order to not account twice for correctable errors that are found during execution and corrected with a correction cycle, the correctable error counter is only incremented when the the error is first encountered and the correction cycle is entered and is then not incremented again in the correction cycle when the value is re-read and corrected.

In addition to the error counters, there is also a counter for the number of times a correction cycle has been triggered. This can be checked to determine if a correctable error was found by the scrubber (not triggering a correction cycle) or found functionally (when loading/fetching instructions).

104.6.7 Error injection

The cache controller supports error injection into the cache tag and data memories, and the TCM data memories. The error injection interface has been designed to allow for easily injecting errors from software running on the processor.

The error injection is performed in two steps. First the error injection is “armed” by configuring the error injection register. Secondly, the error injection is performed by performing a diagnostic ASI read from the memory and location where the error should be injected. The cache controller will during the ASI read perform the normal diagnostic read, but in addition perform a write-back of the value with the selected error injected. By performing the injection with one ASI read access, the error injection is “atomic” from point of view of the cache state, and there is no risk of for example a cache line getting replaced between reading and writing back the value.

The diagnostic read used to inject the error will first correct any existing correctable errors. Therefore errors will not accumulate when injecting errors multiple times into the same location.

For the native FPGA ECC implementation, not all FPGA technologies have support for error injection. There is a read-only field in the error injection register indicatin whether error injection is supported for the current configuration.

104.6.8 Error correcting code description

The cache controller uses an error correcting code that has been designed for low error detection overhead in the cache lookup path, correction of single bit errors, and detection of all wide (up to 4 adjacent bits) errors caused by multiple bit upsets. The code can be viewed as a concatenation of a parity check and a BCH-like code.

The code adds 8 check bits to a data word that is typically 32 bits but can be up to 44 data bits.

Conceptually the code is built up as follows (this example assuming 32 bit wide data):

Step 1: The data is split into four 8-bit parts (“slices”) with each slice having one bit of each nibble (i.e data bits 31,27,...,7,3 in one slice, data bits 30,26,...,6,2 in one slice, data bits 29,25,...,5,1 in one slice, and data bits 28,24,...,4,0 in one slice).

Step 2: For each slice, four checkbits are formed by taking different XOR combinations of the 8 data bits of the slice, similar to a BCH encoder. Each data bit gets included in a unique combination of two or more checkbits.

Step 3: The 4x4 checkbits for the separate parts are combined by bitwise XOR into one set of 4 checkbits. This forms the lower four output checkbits of the encoding process.

Step 4: For the remaining four checkbits, the data bits and the lower four checkbits are viewed as one data word and again split into four parts with each part having one bit of each nibble. All the 9 bits of each part are now XOR:ed to create a parity bit of each part. The four generated parity bits are taken as the top four output checkbits of the encoding process

Since all computations made are based on exclusive-or gates, the steps above to create the different checkbits can be combined into a single XOR function of the data bits.

Decoding is done in two stages. First a parity check is made comparing the four top check bits read out to the XOR of the data bits together with the four lower check bits computed in the same way as in step 4 of the encoding process described above. The parity check generates an “error detected” signal for the cache controller to stop the current access and enter into a correction cycle.

For the error correction, a syndrome is created by generating the lower four checkbits using identical steps to the encoder steps 1-3 above and bitwise XOR:ing the resulting check bits with the check bits read out from the memory. With the result of the parity check indicating which part of the data word has the error, and the syndrome, a look up table can determine which bit caused the error should be corrected. Note that a syndrome with one bit set would be an indication of a bit error in the check bit itself. A parity check showing errors in more than one slice or a syndrome not matching any single-bit error would be an indication of an uncorrectable error at the decoder level. Note that as explained in section 104.6.2, uncorrectable errors in the cache memories can be recovered at higher level by the cache controller by re-reading from memory.

As is the case with any error correcting code, multi-bit error cases that are beyond what the ECC can handle will defeat the check and may result in incorrect data delivered either without any indication of error at all or incorrect data indicated as a correctable error.

A special case exists with this code, where some multi-bit error cases can defeat the initial parity check but are still detected by the syndrome check. These are called “late detected errors”. As the data may have already have been delivered by the cache controller by the time the error is detected, it may be too late to recover from the error. In this case the cache controller can be configured generate an internal error trap to break execution. If the pipeline was held at the time this happens for any reason (for example due to an unrelated cache miss), this gives extra time to act on the error and the pipeline will handle the error in the same way as it would for a regular uncorrectable error. The scrubber will always have the extra time to catch a late detected error so such errors found by the scrubber will always be acted on as uncorrectable errors. Two separate one-bit counters exist for non-recovered and recovered late errors.

104.7 Block redundancy

104.7.1 Overview

Note that the features described in this section are only available in FT and FT-FPGA releases of GRLIB.

Note: Applying block-level TMR on the processor pipeline is still under development and not yet recommended for production designs.

The processor core can be built in a configuration where selected sub-blocks are instantiated multiple times (dual or triple redundant) with comparison logic and voters at the sub-block boundary. This can be used to implement dual core lock step or as an alternative to tool-inserted TMR for certain technologies.

For block redundancy, the processor core has been divided into the following five parts:

- Main processor logic, including processor pipeline, cache controller, MMU, and floating point controller (also including NanoFPU if used).
- Integer register file
- Cache memory, including ECC logic if FT is enabled. Note this cache memory normally expected to be protected with ECC rather than block redundancy, but included for completeness.
- GRFPU5, if GRFPU5 selected as floating point unit
- Floating point register file, if GRPFU5 selected as floating point unit

Each of these five parts can be individually configured as redundant via the redconf generic.

104.7.2 Error signaling

A brerr signal is added to the subsystem top level to indicate that a fatal error was detected on one of the processor cores. For dual-redundant blocks, any miscompare between the two copies would be considered an uncorrectable eror as it is impossible to determine which of the two copies have the correct output value. For triplicated blocks, an uncorrectable error would occur when two sub-blocks fail at exactly the same time, or a second sub-block copy failing before a prior error has been properly washed (as explained below).

It is up to the system designer to decide how to use this signal, but typical usage would be to reset the system. The system designer may also choose to ignore the signal, or possibly delay the reset for some time, as there is a chance of the processor still functioning correctly.

Internally, both uncorrectable and correctable errors are reported via the block-redundancy status register and can be configured to trigger a custom trap 0x61.

104.7.3 Block TMR error recovery

Since there are only voters on the block boundaries, block-redundant TMR configuration requires custom recovery procedures to ensure the state inside a failing block is re-synchronized the same state as the other two working copies.

For the GRFPU5, the recovery is implemented fully in hardware by applying a reset the FPU whenever there are no ongoing floating-point operations. As the GRFPU5 does not need to keep any state between operations there is no additional handling required. Therefore, the GRFPU5 in block-TMR configuration normally requires no special handling by software. Extremely floating point intensive applications may want to occasionally ensure no FPU operations are in progress for a few cycles, however in practice normal software housekeeping activities such as timer interrupt handling and task switching, etc, will achieve this as a side effect.

For integer and floating point register files implemented with block TMR, error recovery is achieved by re-writing all registers. This could be done periodically, or only when an error is detected (via the trap 0x61 handler).

The processor pipeline in block-TMR configuration requires a more complex procedure to recover. The process consists of first dumping processor state to RAM, commanding the debug module to perform a “self-reboot” all three copies via the cpu-to-cpu interface ASI. Onced rebooted the cores then restoring back the state.

104.7.4 Fault domain merging

When the pipeline block and another sub-block have been configured with the same type of redundancy (dual or triple), the fault domains can be configured to be merged. When this is done, the voters and error checkers between the blocks are removed and each string is connected 1:1 to a string of the other sub-block. This can save on resources and avoid crossing between the strings in the internal interfaces, but has the downside of errors not being detected until the pipelines miscompares and requiring a full pipeline recovery procedure.

Note that when the pipeline and register files have been merged into the same fault domain, washing register files is no longer possible simply by reading and writing back registers as voters will no longer crossed in the read-to-write path between the pipeline and register file. Instead the processor needs to make sure the data passes through the debug module’s stash area, or through memory.

104.7.5 Combining block redundancy and ECC

Normally only one of either ECC or block redundancy is used for a specific block (cache or register file), but the two types of protection can be used at the same time. When the cache or register file with ECC enable is triplicated, the correctable error signals are combined with logical OR instead of majority voting in order to ensure correctable errors are accounted and handled rather than voted out.

104.8 Subsystem integration

104.8.1 AMBA bus fabric

The subsystem includes an AMBA 2.0 AHB and APB bus fabric, using the GRLIB AHBCTRL and APBCTRL IP cores. Ports are included to connect external AHB masters and AHB slaves to the processor bus, and to connect additional APB slaves to the APB peripheral bus. The system uses the AMBA Plug and play support and interrupt steering capabilities provided by GRLIB infrastructure (see GRLIB IP Library User’s Manual).

In order to guarantee that software plug’n’play scanning routines find the built in peripherals before any cores (such as additional UARTs) connected externally to the subsystem, the leon5sys block per-

forms a reordering of the AHB and APB slaves relative to how they appear externally to the subsystem.

Table 2034. Processor AHB bus masters

Master index	Connection
0	Processor 0
1	Processor 1
...	...
(ncpu-1)	Processor (ncpu-1)
ncpu	External master 0
ncpu+1	External master 1
...	...
(ncpu+nextmst-1)	External master (nextmst-1)
ncpu+nextmst	Debug module

Table 2035. Processor AHB bus slaves

Slave index, internal/actual	Slave index, external view	
0	nextslv	APB bridge controller (APBCTRL)
1	0	External slave 0
...	...	...
nextslv	nextslv-1	External slave (nextslv-1)

Table 2036. APB bus slaves

Slave index, internal/actual	Slave index, external view	
0	nextapb	Console UART (APBUART)
1	nextapb+1	IRQ controller (IRQMP)
2	nextapb+2	Timer (GPTIMER)
3	0	External APB slave 0
...	...	...
(nextapb+2)	(nextapb-1)	External APB slave (nextapb-1)

104.8.2 Standard peripherals

The following standard peripherals are included in the subsystem block:

- IRQ controller (IRQMP), used to deliver interrupts to the processors
- General purpose timer (GPTIMER), used for time keeping
- Console UART (APBUART), used for standard console output

The peripherals are connected to the subsystem's APB bus.

104.8.3 Reset and start up

The processors are reset indirectly by the debug module, which is in turn reset by the external reset signal. The other parts of the subsystem are directly reset by the external reset signal.

The debug module ensures that the reset pulse is stretched to the number of cycles required by the pipeline and then starts up the first processor (unless the break signal is asserted).

While running, the debug module supports reseting and rebooting individual processors in a controlled way. This is commanded through the CPU to CPU interface

104.8.4 Alternative boot address function

An optional feature exists to have several alternative fixed boot addresses and alternate between them on each reboot of the processor. This may allow recovery from cases where the regular boot ROM is corrupted and causes execution to crash by booting up from a back-up ROM after a watchdog reset.

This functionality is implemented with a small block that monitors the reset signal, and steps up a counter indicating which boot address to use next. This block needs a separate reset signal of its own, which should only be asserted on power-up. The boot-retry block can run in a different (typically slower) clock domain to the processor system.

104.9 Multiprocessing support

104.9.1 Overview

LEON5 is designed to be used in multi-processor systems both in SMP and AMP configurations. The write-through caches combined with the snooping mechanism built into the level-1 cache controllers guarantees memory coherency in shared-memory systems consistent with the SPARC TSO (Total Store Order) memory model. A CPU to CPU interface is provided to allow the processors to start and stop each other.

104.9.2 Cache coherency

Cache coherency is ensured by snooping writes on the processor AHB bus and invalidating cache lines written to by other processors. For this to work, all running processors must have snooping enabled in their cache control register.

Provided snooping is enabled, the processors will achieve a memory model compliant with SPARC Total Store Order (TSO), as seen from software running in parallel on the cores.

104.9.3 Processor start-up

The debug module will start up CPU0 after reset and leave the other processors in stopped state. The other processors will have to be started up by software running on one of the already running processors. Software wishing to only execute on one processor may simply ignore that any of the other processors exist and run on CPU0 as if it was a single-core system.

For LEON3 and LEON4 system compatibility, a legacy method is supported where CPU0 writes to the multiprocessor status register to wake up the other processors, that will then start executing from the same reset address as CPU0. The software at the reset address has to be made SMP-aware by typically checking the processor ID (through ASR17) and jumping directly to the entry point when processors other than CPU0 are running.

The LEON5 CPU-to-CPU interface provides a more flexible interface for CPU0 to control the other processors. Processors may be started, stopped, and rebooted to start at any entry point. A 32-bit “boot word” can be set by the controlling CPU to each of the controlled CPUs. The contents of the boot word is for the that may for example be used to hold a unique virtual processor ID, or a pointer to a per-CPU data structure.

104.9.4 Processor execution control

When viewed from the outside, the integer unit execution follows a state machine model as shown in figure 277 and table 2037. The integer unit always resets into a stopped state, and has to be started by the debug module. The debug module also has additional commands to force the processor into a specific state.

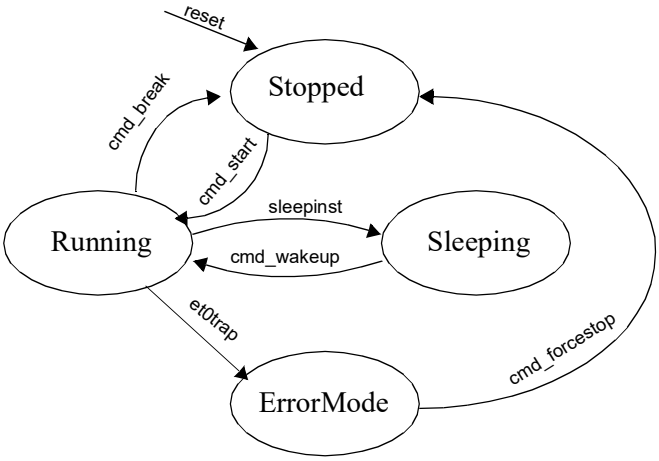


Figure 277. LEON5 state machine model

*Table 2037.*Processor pipeline externally visible state descriptions

State	Encoding	Description
Stopped	00	Reset state. Waiting for debug module to start operation. Debug access allowed
Running	01	Processor executing code. Stopped on error mode condition, sleep instruction, or command from debug module. The processor may also go to stopped state on certain conditions for debugging if enabled via Bx signals from the debug module.
Sleeping	11	Processor idle. Will go back to running state when debug module issues a wakeup command (on a pending interrupt or wakeup command from other CPU).
ErrorMode	10	Processor in error mode due to trap in trap condition. Debug accesses allowed. To resume from this state, the pipeline must be either reset or forced into stopped state by the debug module.

104.9.5 Processor reset operation

The following table indicates the reset values of a subset of the registers which are affected by the reset. Since the processor resets into a stopped state, the registers may in some cases be modified via the debug port before starting execution.

*Table 2038.*Processor reset values

Register	Reset value, memmap=0	Reset value, memmap=1
Trap Base Register	0x0	0xC0000000
PC (program counter)	0x0	0xC0000000
nPC (next program counter)	0x4	0xC0000004
PSR (processor status register)	ET=0, S=1	ET=0, S=1

104.9.6 CPU-to-CPU interface

The CPU to CPU interface allows a processor to control and monitor other processors. This interface is accessed from software over ASI 0x22, and is implemented using signals between the debug module and cache controllers.

A permission matrix tracks which processors are allowed to control which, and any illegal commands will be ignored. By default CPU0 is allowed to control all remaining CPUs and the other CPUs are not allowed to control any processor.

For monitoring, the processor can read out the user-visible state (as described in section 104.9.4) of each of the processors it has permission to control. For controlling, it may send start, break and wakeup commands to the other processors by writing to specific registers. A custom “boot word” can be set by the controlling processor, that the processors can read out.

There is in addition a reboot command that will make the debug module reset the processor pipeline and start up at a specified address. The reset is done in a controlled way that guarantees there is no currently ongoing access to the AMBA bus by the processor that is restarted.

Through a delegation register, permissions can be handed over to a group of processors, for example CPU0 may hand over control over CPU5-7 to CPU5. Each processor has read only access to its segment of the permissions matrix to see how many other CPUs it has control over at the current time. This allows partitioning of processors where the software dynamically adapts to the number of processors it’s been assigned.

104.10 Debug support

104.10.1Overview

Debug support is implemented by a combination of features in the various parts of the LEON5 subsystem.

The processor pipeline provides debug signals that allows the execution to be stopped at any time, and also to make it stop automatically at certain conditions. When stopped, the software visible state of the pipeline can be accessed through the debug signals. Also, loads and stores to arbitrary addresses and ASIs can be performed through the debug port, allowing access to any other software-visible registers and spaces.

The L1 caches, MMU, and FPU implement diagnostic features that can be accessed from the processor, either functionally from software or through the debug interface while debugging. These features are accessed through alternative address spaces.

The subsystem includes a central debug and control module that connects to the debug ports of all CPUs. The module handles resetting the processors and starting them up after reset, unless configured to break the execution instantly for debugging.

The debug module provides ports for external AHB debug masters. The debug masters can access the processors debug ports, read from the trace buffers, access internal registers of the debug module itself, and perform accesses directly on the processor AHB bus.

104.10.2Command bus

Each processor has a command bus that is used for execution control (starting, stopping, etc). The commands are used both for functional (inter-processor control, waking up on interrupt) and debugging purposes. The command bus is driven by the debug module which acts as a control hub also for the functional start/stop control.

The commands into the pipeline from the debug module are listed in table 2039. When the processor is in Running state, the command needs to be given continuously on the command bus until the externally visible state changes. In the Stopped, Sleeping and ErrorMode states, commands only need to be

given for one cycle to have effect. Commands may not be given at the same time as a debug register access is taken place.

Table 2039.

Command	Encoding	
None	000	No command.
Start	001	Moves the pipeline from stopped to running state so that it starts executing. This is given automatically by the debug module after reset for processor 0.
Wakeup	010	Moves the pipeline from sleeping to running state.
Break	011	Moves the pipeline to stopped state if it is in Running or Sleeping states.
ForceStop	100	Moves the pipeline to stopped state from any state. Used typically by debugger to get processor back to initial state.
ForceRun	101	Moves the pipeline to running state from any state. Included for completeness, not used normally.
ForceErr	110	Moves the pipelne to ErrMode state from any state. Included for completeness, not used normally.
ForceSleep	111	Moves the pipeline to Sleeping state from any state. Typically used by debugger to resume execution for processors that were sleeping at the time of breaking.

While debugging, commands can be sent to each processor through the debug module's register interface.

104.10.3Current and previous state tracking

The debug module contains status registers that track which state each processor is currently in. Refer to section 104.9.4 for a description of the state model.

The debug module looks for changes of processor state and keeps a register tracking the previous state each processor was in before entering the current one. In addition, there is a bit set indicating when a processor has been stopped if this was caused by a user level command (i.e one processor stopping another through the CPU-to-CPU interface) or by a stop command entered via the debug register interface. This information is needed by the debugger after breaking execution to resolve whether the processors actually stopped by the debugger or if they were already in stopped state anyway, and then to determine whether they should be resumed when continuing execution.

104.10.4Start-up handling

After reset, the debug module will start up CPU0 to boot from its default reset address. This can be suppressed by asserting the dsubreak signal during reset, if this is set then the processor will be left in stopped state and no instructions will be executed until the user connects with a debugger and starts the processor.

104.10.5Wake-up handling

The functionality to wake up the processor from sleep mode (i.e while waiting for an interrupt) is managed by a combination of the processor, interrupt controller and debug module. Each processor sends an interrupt pending signal up to the debug module, that then responds by issuing a wakeup command on the command bus to that processor.

CPU-to-CPU command handling

Processors starting and stopping each other, either via the CPU-to-CPU interface described in section 104.9.6 or via the legacy IRQMP status register interface, are handled through the debug module. The

start/stop commands are stored in “pending” bits that are the forwarded to the command bus of the accessed CPUs.

A block user commands bit (blockusr) can be set in the debug module to block any of these commands to be delivered to the CPU. The commands are left pending until the blockusr bit is cleared. This is to allow for debugging of a single CPU even if other CPUs are still running that may be trying to start and stop it. The debug module interface has been designed to allow atomically setting the blockusr bit at the same time as sending the stop command to the processors.

104.10.6Processor break conditions

Control signals into the pipeline from the debug module can configure it to automatically break execution into stopped state (and set the usrblock bit) when certain traps occur.

- BA - any traps
- BE - any traps except window over/underflow, floating point disabled, interrupts, or software traps
- BS - software trap 1 (software breakpoint)
- BW - watchpoint exception (hardware breakpoint)

These signals are tied to register bits in the debug module.

104.10.7Break matrix

To allow breaking multiple cores at once for debugging, the debug module contains a (NCPU x NCPU) bit matrix to control what should happen when one CPU stops. When a CPU transitions into Stopped or ErrorMode state, that CPUs row in the break matrix is evaluated and any CPUs that have their break bit set to 1 will be sent the break command and have it’s usrblock bit set.

104.10.8Debug access

An interface is implemented between the debug module and the LEON5 pipeline to allow the debug module access the registers of the integer pipeline. The pipeline must be in Stopped or ErrorMode state while the register access is performed. This interface also allows provides indirect access to the cache and all registers that can be accessed from software via alternate address space identifiers.

104.10.9Debug master access

The debug module provides a set of ports for AHB masters to connect to for debug access. Standard debug masters from the GRLIB library can be connected to these debug ports.

From the debug master’s point of view, the behavior is as if the master was directly connected to the processor bus, with the addition of an extra AHB slave area representing the debug module. The debug masters will see the AMBA plug and play information on the processor bus, however in addition the debug module overlays additional plug and play records. One slave record is added for the debug register interface, and master records are added for each debug master connected to the debug module.

The debug module implements a transparent bridge between each debug master and the processor AHB bus, that provides pipeline registers so that there are no direct combinatorial paths between the debug masters and the processor bus, or between the debug masters.

The debug masters can be blocked from accessing the system using the dsuen signal.

104.10.10Register area

The debug module provides a register area for debugging and control that can be accessed only by the debug AHB masters. Depending on which address is accessed, the access may be handled directly

inside the debug module, or it may lead to an internal access into the LEON5 pipeline. The register map and descriptions are in section 104.16.

104.10.11Debug sessions

The debug module provides support for a concept called debug sessions. The purpose of this is to allow debug scenarios where multiple debuggers are active on the system in parallel, debugging different subsets of the system without interfering with each other. The debug module provides 4 sessions if the number of CPUs are greater than four, or a number of sessions is equal to the number of CPUs if less than or equal to four.

Each debugger has to claim a unique session ID when connecting, and the debug module provides a simple protocol to achieve this. When a session has been claimed, by default it gets all resources that are not currently in use. The debugger should check that it got all needed resources and then free up the ones it does not intend to use for the next session.

Each session provides a poll register where the processors and other resources allocated to the session can be polled for events using a single register read. The session also provides commands to sample the state of all allocated CPUs atomically or to send a command to all allocated CPUs atomically. A per-session buffer is provided to hold the read/write data for these action.

104.10.12Debug session allocation protocol

The debug session IDs are arbitrated using the following protocol that must be followed by all debuggers in order for the sessions to work (and debuggers not supporting multiple debug sessions are recommended to follow it for forward compatibility):

1. Read the global debug status register to get the number of sessions Nsess supported

2. Try each possible session ID between 0 and Nsess-1:

2.1 Write to the session’s claim register with the pending bit set to 1

2.2 Read back the session’s claim register.

2.3 If both the pending and claimed bit are high in the read back value from 2.2 then we are the owners of this session and can proceed to step 3. If getting some other values on the two bits, then go back to 2.1 and try the next ID. If all session IDs failed then all sessions are already in use, stop or retry from step 1.

3. Read the resource masks from the acquired session’s register area

4. If the resource masks do not have the needed bits set, the targeted CPUs are already being debugged. Go to step 7 to deallocate the session, then stop or retry from step 1.

5. Clear the bits of the resources we do not need and write back to the session resource masks.

6. The session is now set up and can be used for debugging. When done, proceed to step 7.

7. Free the session ID by writing 1 to the claimed register

104.10.13UART interception

The debug module contains a UART debug function that works transparently to the system being debugged. It can capture the outgoing data over the transmit lane for reading out via the debug modules register area. Optionally the characters can still also go out to the external UART.

An overrun bit indicates if characters were lost in the debug interface due to not reading fast enough. Optionally the flow control signals into the UART can be asserted when the debug interface FIFO is full in order to avoid overruns, flow control must be enabled in the UART for this to have effect.

The data capture is integrated with the debug sessions so that the UART capture and the processors can be polled using a single register read.

UART reception can also be intercepted by the debug module to inject input characters. The debug module supports optionally using flow control (to avoid overrun inside the UART) and optional blocking of external input.

104.10.14Internal deadlock detection

A debug access deadlock detection feature is implemented in the debug module, to detect if a debug access into the pipeline does not return with data within a reasonable time. This will normally only trigger due to an hardware fault (corrupted state of the pipeline) but could also happen due to incorrectly accessing the debug interface of the pipeline without stopping the processor first. The feature is intended to allow for some investigation into the cause of an unexpected stall, but note that a full reset is still required after a deadlock to recover from the error.

If a debug register access leading into the LEON5 pipeline times out and the deadlock detection triggers, the debug module will finish the access towards to the debug master (release the HREADY signal) and set a number of internal registers to indicate that a deadlock occurred and capture which address was accessed. Note that the data returned to the master for the access that failed will be unpredictable.

In order to see that a deadlock triggered, the processor status register can be checked. The event poll register will also indicate the deadlock detection status, so this can be checked as part of the regular event polling without any extra access.

Note that the deadlock detection is only activated on debug accesses over parts of the DSU5 memory area, regular accesses towards memory or peripherals that go into the processor AHB bus may have any number of wait states and if they stall indefinitely the debug master will be correspondingly locked up until the design is reset.

104.10.15Low level debug register

In order to debug AHB bus lockup situations, the debug module has a low level debug register that can be accessed by the other debug masters even if one master is stuck waiting for an access to complete. In systems with more than one debug link, this allows going in via another debug link and getting some basic information.

The debug register contains the current HREADY status of the bus and whether the deadlock detection hit. It also allows to read-out a set of design-specific signals fed in on the sysstat input of the subsystem.

The low level debug register can be written in order to request a soft reset of the system. If the debug register is written with the value 0x99, then the rstreqn signal of the subsystem will be asserted. Whether or not this actually triggers a reset depends on if the signal is connected externally.

104.11 Memory map

104.11.1Physical memory map

The subsystem allows two different memory maps depending on the memmap generic. Memory map 0 provides the most compatibility and similarity with existing LEON3 systems, while memory map 1 provides a larger address space for memory and has peripheral registers aligned on 4 KiB page boundaries.

The subsystem implements the following memory map as seen from the processor (using physical addresses) or from other masters on the processor bus, or from masters on the debug ports:

*Table 2040.*Physical bus memory map, memmap=0

Range	Function	Comment
0x00000000 - 0x7FFFFFFF	For use by external AHB slaves (main memory)	Typically, boot ROM at 0x00000000, RAM at 0x40000000
0x80000000 - 0x8000000FF	For use by external APB slave	Typically used by memory controller
0x80000100 - 0x800001FF	Console UART registers	See APBUART documentation
0x80000200 - 0x800002FF	IRQ controller registers	See IRQMP documentation
0x80000300 - 0x800003FF	Timer registers	See GPTIMER documentation
0x80000400 - 0x800FFFFF	For use by external APB slaves	
0x80100000 - 0x8FFFFFFF	For use by external AHB slaves	
0x90000000 - 0x9FFFFFFF	Debug access	Only accessible from debug ports
0xA0000000 - 0xFFFFEFFF	For use by external AHB slaves	
0xFFFFF000 - 0xFFFFFFFF	AMBA Plug'n'play information	Read only. Different information when seen from debug port or via processor bus.

*Table 2041.*Physical bus memory map, memmap=1

Range	Function	Comment
0x00000000 - 0xDFFFFFFF	For use by external AHB slaves (main memory)	Typically, RAM at 0x00000000, RAM/IO/ROM at 0x80000000, boot ROM at 0xC0000000
0xE0000000 - 0xEFFFFFFF	Debug access	Only accessible from debug ports
0xF0000000 - 0xFF8FFFFF	For use by external AHB slaves	
0xFF900000 - 0xFF900FFF	Console UART registers	See APBUART documentation
0xFF901000 - 0xFF903FFF	For use by external APB slave	
0xFF904000 - 0xFF904FFF	IRQ controller registers	See IRQMP documentation
0xFF905000 - 0xFF907FFF	For use by external APB slave	
0xFF908000 - 0xFF908FFF	Timer registers	See GPTIMER documentation
0xFF909000 - 0xFF9FFFFF	For use by external APB slaves	
0xFFA00000 - 0xFFFFEFFF	For use by external AHB slaves	
0xFFFFF000 - 0xFFFFFFFF	AMBA Plug'n'play information	Read only. Different information when seen from debug port or via processor bus.

104.11.2 Alternative address spaces

The alternative address spaces (ASIs) are implemented inside the LEON5 cache controller and only accessible from inside the processor or indirectly through the debug module. The table shows the ASI usage for LEON5.

Table 2042. ASI usage

ASI	Address	Usage	Ref
0x01	virtual	Forced cache miss. Address is virtual address as for regular load.	
0x02	System control registers		
	0x00000000	Cache control register	104.12.6
	0x00000008	Instruction cache configuration register	104.12.7
	0x0000000C	Data cache configuration register	104.12.7
	0x00000010	LEON5 configuration register 2	104.12.3
	0x00000014	LEON5 FT configuration register (FT only)	104.13.2
	0x00000018	Region flush mask register	104.12.8
	0x0000001C	Region flush address register	104.12.9
	0x00000020	AHB error register	104.12.10
	0x00000024	AHB error address register	104.12.11
	0x0000002C	Trap register	
	0x00000030	Cache error injection register (FT configurations only)	104.13.4
	0x00000034	Scrubber position register (FT configuration only)	104.13.5
	0x00000038	Error counter register (FT configuration only)	104.13.6
	0x0000003C	Boot word register read-out	104.12.13
	0x00000040	TCM configuration register (TCM configuration only)	104.12.14
	0x00000044	TCM FT configuration register (TCM+FT configuration only)	104.13.3
	0x00000048	Instruction TCM control register	104.12.15
	0x0000004C	Data TCM control register	104.12.16
	0x00000060	Block redundancy config register (BR configuration only)	104.14.2
	0x00000064	Block redundancy error state (BR configuration only)	104.14.3
	0x00000068	Block redundancy trap configuration (BR configuration only)	104.14.4
0x08, 0x09	-	Not supported	
0x0A	virtual	Access memory as if in user mode. Note: for backward compatibility with LEON3/LEON4, a special case is made for the CASA instruction. CASA with ASI 0xA will access memory in user or supervisor mode depending on current status in the PSR register.	
0x0B	virtual	Access memory as if in supervisor mode.	
0x0C	cache-index	Instruction cache tags diagnostic access	
0x0D	cache-index	Instruction cache data diagnostic access	
0x0E	cache-index	Data cache tags diagnostic access	
0x0F	cache-index	Data cache data diagnostic access	
0x10, 0x13	(ignored)	Flush instruction and data cache	
0x11	(ignored)	Flush data cache	
0x18, 0x03	(ignored)	Flush TLB and I/D cache	
0x19, 0x04	MMU control registers		
	0x00000000	MMU control register	104.12.19
	0x00000100	Context pointer register	104.12.20
	0x00000200	Context register	104.12.20
	0x00000300	Fault status register	104.12.21
	0x00000400	Fault address register	104.12.22

Table 2042. ASI usage

ASI	Address	Usage	Ref
0x1B	vtag	MMU flush/probe	
0x1C	physical	MMU and cache bypass	
0x1E	cache-index	Snoop tags diagnostic access	
0x20	Floating point controller register area		
	0x00000000 - 0x0000007C	Floating-point register diagnostic access	
	0x00000080	Floating-point state register (FSR) diagnostic access	
	0x00000088	Deferred floating-point queue diagnostic read out	
	0x00000090	LEON5 FPC configuration register	
	0x00000094	FPC exception state modification register	
	0x00000098 - 0x0000009C	Deferred floating-queue entry write buffer	
0x21		Reserved for coprocessor	
0x22	CPU to CPU interface		
	0x00000000 - 0x00000004	Effective control bitmap	104.15.2
	0x00000008 - 0x0000000C	Delegation bitmap	104.15.3
	0x00000010 - 0x0000001C	CPU states read out	104.15.4
	0x00000020	Reboot request reset address	104.15.5
	0x00000024	Reboot request control and status	104.15.6
	0x00000028	Legacy interface control register	104.15.7
	0x00000030 - 0x00000034	SMP broadcast group register	104.15.8
	0x00000038	SMP broadcast configuration register	104.15.9
	0x0000003C	SMP broadcast request register	104.15.10
	0x00000040 - 0x00000044	Send start command	104.15.11
	0x00000048 - 0x0000004C	Clear pending start command	104.15.11
	0x00000050 - 0x00000054	Send break command	104.15.11
	0x00000058 - 0x0000005C	Clear pending break command	104.15.11
	0x00000060 - 0x00000064	Send wakeup command	104.15.11
	0x00000068 - 0x0000006C	Clear pending wakeup command	104.15.11
	0x000000F8 - 0x000000FC	Block-redundancy data stash area (block-redundant configuration only)	104.14.5
	0x00000100 - 0x000001FC	Read/write boot words	104.15.12
0x23	TLB diagnostic access		
	0x00000000 - 0x000000FC	ITLB entries	
	0x00000100	ITLB pseudo-MRU state	
	0x00000200 - 0x000002FC	DTLB entries	
	0x00000300	DTLB pseudo-MRU state	
0x24		Integer unit BTB/BHT diagnostic interface	
0x25	cache-index	LRU diagnostic access	
0x26	tcmaddr	Instruction TCM data access	
0x27	tcmaddr	Data TCM data access	

104.11.3 IRQ assignments

The internal peripherals have been assigned interrupts in the IRQ controller corresponding to table 2043. For interrupts 1-15, they will be delivered from the IRQ controller using the corresponding interrupt number and PIL level, extended interrupts 16-31 are delivered as interrupt number 12. Exter-

nal AHB and APB devices may drive any of the interrupt levels via the GRLIB interrupt steering infrastructure (HIRQ and PIRQ signals).

Table 2043. Interrupt assignments

Interrupt number	Peripheral	Comment
1		Free for use by external peripherals
2	APBUART	UART interrupt line
3-7		Free for use by external peripherals
8	GPTIMER	Timer 1 interrupt
9	GPTIMER	Timer 2 interrupt
12	IRQMP	Interrupt level used for extended interrupts 16-31.
13-15		Reserved for use by software in multiprocessor systems.
16-31		Free for use by external peripherals. Extended interrupts delivered as interrupt 12, see IRQMP manual for details.

104.12 Processor configuration registers

104.12.1PSR, WIM, TBR registers

The %psr, %wim, %tbr registers are implemented as required by the SPARC V8 manual. They can be read and written from software through dedicated instructions.

*Table 2044.*LEON5 Processor state register (%psr)

31 28	27 24	23 20	19 16
IMPL	VER	ICC	RESERVED
0xF	0x5	0	0
r	r	r	r

15 14	13	12	11 8	7	6	5	4 0
RESERVED	EC	EF	PIL	S	PS	ET	CWP
0	0	0	0	1	1	0	0
r	r	rw*	rw	rw	rw	rw	rw

31:28	Implementation ID (IMPL), read-only hardwired to “1111” (15)
27:24	Implementation version (VER), read-only hardwired to “0101” (5) for LEON5.
23:20	Integer condition codes (ICC), see sparcv8 for details
19:14	Reserved
13	Enable coprocessor (EC), read-only if coprocessor not implemented
12	Enable floating-point (EF), read-only if FPU not implemented.
11:8	Processor interrupt level (PIL) - controls the lowest IRQ number that can generate a trap
7	Supervisor (S)
6	Previous supervisor (PS), see sparcv8 for details
5	Enable traps (ET)
4:0	Current window pointer

*Table 2045.*LEON5 Window invalid mask (%wim)

31 NWIN	NWIN-1 0
RESERVED	WIM
0	NR
r	rw

*Table 2046.*LEON5 Trap base address regsiter (%tbr)

31 12	11 4	3 0
TBA	TT	R
*	0	0
rw	rw	r

31:12	Trap base address (TBA) - Top 20 bits used for trap table address
11:4	Trap type (TT) - Last taken trap type. Read only.
3:0	Always zero, read only

104.12.2ASR17, LEON5 configuration register 1

The ancillary state register 17 (%asr17) provides information on how various configuration options were set during synthesis. This can be used to enhance the performance of software, or to support enumeration in multi-processor systems. There are also a few bits that are writable to configure certain aspects of the processor. The layout of this register has been preserved to keep compatibility with LEON3/LEON4 code, new fields specific to LEON5 have been added to a separate register.

*Table 2047.*LEON5 configuration register 1 (%asr17)

31	30	29	28	27	26	25	24	23	22	21	18	17	16
INDEX				DBP	RES	DBPM	REXV		REXM		RESERVED	CS	CF[1]
*				0	0	0	00		00		0	*	*
r				rw	r	rw	r		r		r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CF[0]	DWT	SVT	LD	FPU		M	V8	NWP			NWIN				
*	*	0	0	01		0	1	*			*				
r	rw	r	r	r		r	r	r			r				

31:28	Processor index (INDEX) - In multi-processor systems, each LEON core gets a unique index to support enumeration.
27	Disable Branch Prediction (DBP) - Disables branch prediction when set to ‘1’. Default value is ‘0’.
26	Reserved for future implementations
25	Disable Branch Prediction on instruction cache misses (DBPM) - When set to ‘1’ this avoids instruction cache fetches (and possible MMU table walk) for predicted instructions that may be annullated. This feature is on by default (reset value ‘1’).
24:21	Always reads 0000 to indicate REX not supported on LEON5
20:18	Reserved for future implementations
17:15	Always reads 000 to indicate that clock switching is not supported on LEON5
14	Disable write error trap (DWT). When set, a write error trap (tt = 0x2b) will be ignored. Set to zero after reset.
13	Always reads 0 to indicate that single vector trapping is unsupported on LEON5.
12	Always reads 0 to indicate a load-delay of 1
11:10	FPU option. Always “01” on LEON5 (GRFPU5 or NanoFPU)
9	Always reads 0 to indicate that the optional multiply-accumulate (MAC) instruction is unavailable on LEON5
8	Always reads 1 to indicate that the SPARC V8 multiply and divide instructions are available
7:5	Number of implemented watchpoints (NWP) (0 - 4)
4:0	Number of implemented registers windows corresponds to NWIN+1.

104.12.3LEON5 configuration register 2

The LEON5 configuration register 2 located at ASI 2 offset 0x10 provides information on how various configuration options were set during synthesis.There are also a few bits that are writable to configure certain aspects of the processor.

*Table 2048.*LEON5 configuration register 2 (L5CFG2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DTAGCONF		FTEN	ITCME	DTCME	RES	FSREV			RESERVED						
*		*	*	*	0	001			0						
r		r	r	r	r	r			r						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED			HWCE	AWCE	STBPV	RES	FSTBP	FBP	FBTB	DLAR	DLWI	DBTB	SINGIS	DLALU	FPSP
0			1	0	0	0	0	0	0	0	0	0	0	0	1
r			rw	rw	rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw

31:30 Data cache tag configuration

29 FT configuration. If set to 1, the FT configuration register is valid.

28 Instruction TCM enabled. If set to 1, the TCM configuration register is valid.

27 Data TCM enabled. If set to 1, the TCM configuration register is valid.

26 Reserved for future implementations

25 : 23 Feature set revision.This field is incremented when several features have been added to the LEON5 core to avoid having to have individual feature bits for each feature..

000: First version.

001: Cache trap register added. MMUWTM/AHBWTM fields in AHB error register. Memory barrier functionality added. Current version.

010-111: Reserved for future versions.

22 Block-redundant configuration. If set to 1, the block redundancy configuration regsiter is valid.

21 : 13 Reserved for future implementations

12 Enable hint-based write combining

11 Enable automatic write combining

10 When dynamic branch prediction is disabled with bit 8, select between always-taken (1) or never-taken (0) static branch prediction.

9 Reserved for future implementations

8 Set to 1 to disable dynamic branch prediction and force static branch prediction

7 Set to 1 to freeze branch predictor state

6 Set to 1 to freeze BTBcontents

5 Set to 1 to disable late arithmetic in pipeline

4 Set to 1 to disable late icc update in pipeline

3 Disable branch target buffer

2 Set to 1 to force single issue of all instructions executed in pipeline

1 Set to 1 to disable late ALU in pipeline

0 Set to 1 to allow starting FPU operations speculatively (in predicted branches)

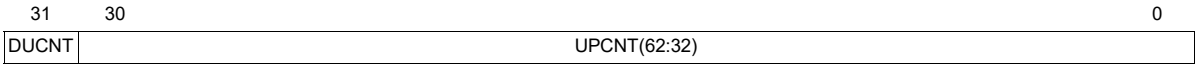
104.12.4ASR22-23 - Up-counter

The ancillary state registers 22 and 23 (%asr22-23) contain an internal up-counter that can be read by software without causing any access on the on-chip AMBA bus. The number of available bits in the counter is implementation dependent and is decided by the number of counter bits in the DSU time tag counter. %ASR23 contains the least significant part of the counter value and %ASR22 contains the most significant part. In case the implementation does not contain a debug support unit connected to the processor then the up-counter is not available (value is always zero).

The time tag value accessible in these registers is the same time tag value used for the system's trace buffers (if implemented) and for all processors connected to the same debug support unit. The time tag counter will increment when any of the trace buffers is enabled, or when the time tag counter is forced to be enabled via the DSU register interface, or when any processor has its %ASR22 Disable Up-counter (DUCNT) field set to zero.

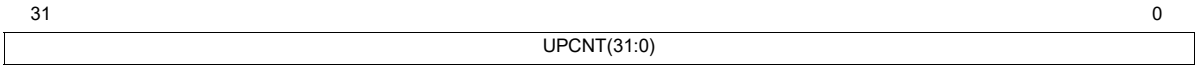
The up-counter value will increment even if all processors have entered power-down mode.

*Table 2049.*LEON5 up-counter MSbs (%ASR22)



31	Disable Up-counter (DUCNT) - Disable upcounter. When set to '1' the up-counter may be disabled. When cleared, the counter will increment each processor clock cycle. Default (reset) value is '1'.
30:0	Counter value (UPCNT(62:32)) - Most significant bits of internal up-counter. Read-only.

*Table 2050.*LEON5 up-counter LSbs (%ASR23)



31:0	Counter value (UPCNT(31:0)) - Least significant bits of internal up-counter. Read-only.

104.12.5ASR24-31, Hardware watchpoint/breakpoint registers

Each breakpoint consists of a pair of ancillary state registers (%asr24/25, %asr26/27, %asr28/29 and %asr30/31) registers; one with the break address and one with a mask:

%asr24, %asr26
%asr28, %asr30

31 … 2	1	0
WADDR[31:2]		IF
NR	0	0
rw	r	rw

%asr25, %asr27
%asr29, %asr31

31 … 2		0
WMASK[31:2]	DL	DS
NR	0	0
rw	rw	rw

Figure 278. Watch-point registers

WADDR - Address to compare against

WMASK - Bit mask controlling which bits to check (1) or ignore (0) for match

IF - break on instruction fetch from the specified address/mask combination

DL - break on data load from the specified address/mask combination

DS - break on data store to the specified address/mask combination

Note: Setting IF=DL=DS=0 disables the breakpoint

When there is a hardware watchpoint match and DL or DS is set then trap 0x0B will be generated.

104.12.6Cache control register

The cache control register located at ASI 0x2, offset 0, contains control and status registers for the I and D cache.

*Table 2051.*LEON5 Cache Control Register (CCR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	STE	R	PS	TB				DS	FD	FI	FT		R	ST	R
0	*	0	*	0				0	0	0	*		0	*	0
r	rw*	r	rw*	rw*				rw	rw	rw	r		r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IP	DP	ITE		IDE		DTE		DDE		DF	IF	DCS		ICS	
0	0	0		0		0		0		0	0	0		0	
r	r	rw*		rw*		rw*		rw*		rw*	rw*	rw		rw	

31	Reserved
30	Snoop Tag Flag (STE) - Used on LEON3FT/LEON4FT to indicate that a parity error was detected in the data physical (snoop) tags. Always reads zero on LEON5, for error counters refer to the separate FT error counter register.
29	Reserved
28	Parity Select (PS) - Used on LEON3FT/LEON4FT to modify the diagnostic ASI so that it will return 4 check bits in the lsb bits. Always reads zero on LEON5.
27:24	Test Bits (TB) -Used on LEON3FT/LEON4FT for bits to be xored during diagnostic write. Not used on LEON5 and always reads zero. Refer to error injection register for error injection support.
23	Data cache snoop enable (DS) - if set, will enable data cache snooping.
22	Flush data cache (FD). If set, will flush the data cache. Always reads as zero.
21	Flush Instruction cache (FI). If set, will flush the instruction cache. Always reads as zero.
20:19	FT scheme (FT) - “00” = no FT, “01” = generic RTL-based ECC implemented, “10” - Technology-specific protection implemented.
18	Reserved for future implementations
17	Separate snoop tags (ST). This bit is always set to 1 on LEON5 for LEON3/LEON4 compatibility, to indicate that ASI 0x1E is available.
16	Reserved
15	Instruction cache flush pending (IP). This bit is set when an instruction cache flush operation is in progress
14	Data cache flush pending (DP). This bit is set when an data cache flush operation is in progress.
13:12	Instruction Tag Errors (ITE) - Used on LEON3FT/LEON4FT to indicate number of detected parity errors in the instruction tag cache. Always reads zero on LEON5, for error counters refer to the separate FT error counter register.
11:10	Instruction Data Errors (IDE) - Used on LEON3FT/LEON4FT to indicate number of detected parity errors in the instruction data cache. Always reads zero on LEON5, for error counters refer to the separate FT error counter register.
9:8	Data Tag Errors (DTE) - Used on LEON3FT/LEON4FT to indicate number of detected parity errors in the data tag cache.Always reads zero on LEON5, for error counters refer to the separate FT error counter register.
7:6	Data Data Errors (DDE) - Used on LEON3FT/LEON4FT to indicate number of detected parity errors in the data data cache. Always reads zero on LEON5, for error counters refer to the separate FT error counter register.
5	Data Cache Freeze on Interrupt (DF) - If set, the data cache will automatically be frozen when an asynchronous interrupt is taken.
4	Instruction Cache Freeze on Interrupt (IF) - If set, the instruction cache will automatically be frozen when an asynchronous interrupt is taken.
3:2	Data Cache state (DCS) - Indicates the current data cache state according to the following: X0= disabled, 01 = frozen, 11 = enabled.
1:0	Instruction Cache state (ICS) - Indicates the current data cache state according to the following: X0= disabled, 01 = frozen, 11 = enabled.

104.12.7I-cache and D-cache configuration registers

The configuration of the two caches if defined in two read-only registers: the instruction and data configuration registers. They are located under ASI 2 at offset 8 and 12.

*Table 2052.*LEON5 Cache configuration register

31	30	28	27	26	24	23	20	19	18	16
CL	REPL		SN	WAYS		WSIZE		LR	LSIZE	
0	001		*	*		*		0	*	
r	r	r	r	r		r		r	r	

15	4	3	2	1	0
RESERVED		M	SO	RESERVED	
0		1	0	0	
r		r	r	r	

31	Cache locking (CL). Set if cache locking is implemented (always zero on LEON5)
30:28	Cache replacement policy (REPL). Always reads “01” (least recently used) on LEON5.
27	Cache snooping (SN). Set if snooping is implemented.
26:24	Cache associativity (WAYS). Number of ways in the cache: 000 - direct mapped, 001 - 2-way associative, 010 - 3-way associative, 011 - 4-way associative
23:20	Way size (WSIZE). Indicates the size (KiB) of each cache way. Size = 2^{SIZE}
19	Local ram (LR). Used on LEON3 to indicate that local scratch pad ram is implemented, always reads 0 for LEON5. For TCM availability on LEON5, refer to LEON5 configuration register 2.
18:16	Line size (LSIZE). Indicated the size (words) of each cache line. Line size = 2^{LSZ}
15:4	RESERVED
3	MMU present (M). This bit is set to ‘1’ if an MMU is present. Always reads 1 on LEON5
2	SO (supervisor only access) bit is present. Always reads 0 on LEON5
1:0	Reserved for future implementations

104.12.8Region flush mask register

The region flush mask register is located at ASI 0x2 address 0x18. It controls which address bits are compared for the region flush function.

*Table 2053.*LEON5 Region flush mask register

31	16
REGION FLUSH MASK[31:16]	
0000000000000000	
rw	

15	4	3	0
REGION FLUSH MASK[15:4]		RESERVED	
000000000000		0	
rw		r	

31 : 4	Region flush mask. Address bits with the corresponding mask bit set to 1 will be compared with the address register during the region flush,
3 : 0	Reserved

104.12.9Region flush address register

The region flush address register is located at ASI 0x2 address 0x1C. It controls the address bits are matched for the region flush function, and holds the bits to initiate the region flush.

*Table 2054.*LEON5 Region flush address register

31	16
REGION FLUSH ADDRESS[31:16]	

*Table 2054.*LEON5 Region flush address register

0000000000000000					
rw					
15 ... 4		3	2	1	0
REGION FLUSH ADDRESS[15:4]		RESERVED		IRFL	DRFL
000000000000		0		0	0
rw		r		rw	rw

31 : 4	Region flush address
3 : 2	Reserved
1	Instruction cache region flush enable, set to 1 to initiate region flush on instruction cache
0	Data cache region flush enable, set to 1 to initiate region flush on data cache

104.12.10AHB error register

The AHB error register is located at ASI 0x2 address 0x20. It contains the AHB error monitoring status, and allows configuring error response handling.

*Table 2055.*LEON5 AHB error register

31 30	29 28	27	26	25	24	23 ... 21	20 ... 16
MMUWTM	AHBWTM	MIAE	MDRV	MDR	MDAE	RESERVED	ACCERR
10	01	0	0	0	0	0	nr
rw	rw	rw	rw	rw	rw	r	wc

15	14 ... 11	10 ... 8	7 6	5 4	3	2	1	0
CHWR	CHMASTER	CHSIZE	RESERVED	CETYPE	OERM	ERRM	OER	ERR
nr	nr	nr	0	nr	nr	nr	nr	nr
r	r	r	r	r	wc	wc	wc	wc

31 : 30	Write trap mode for errors during by MMU writeback 00: Disabled. 01: Deferred, 10: Interrupting, 11: Direct See section 104.3.21 for further description of the different modes.
29 : 28	Write trap mode for errors during regular store. 00: Disabled. 01: Deferred, 10: Interrupting, 11: Direct See section 104.3.21 for further description of the different modes.
27	Set to 1 to mask instruction access traps when getting AHB errors for instruction cache fetches. The fetched instruction is instead replaced by an UNIMP instruction.
26	Select if replacement value for data cache fetches should be 0:all-zeros or 1:all-ones. Only used if bits 25 and 24 are both set.
25	Select if data words from error responses should be 0:passed through as-is or 1:replaced with known value. Only used if bit 24 is set.
24	Set to 1 to mask data access traps when getting AHB errors for data fetches. The contents of the fetched data is determined by bits 26:25.
23 : 21	Reserved
20 : 16	Accumulated AHB errors. Each bit is set to 1 when a certain AHB error occurs for accesses initiated by this processor. Write 1 to each bit in order to clear it. Register is not reset. 20: MMU write, 19: MMU read, 18: Data write, 17: Data read, 16: Instruction read
15	Captured AHB error HWRITE
14 : 11	Captured AHB error master index
10 : 8	Captured AHB error HSIZE
7 :6	Reserved
5: 4	Captured AHB error type. Only valid if captured error came from this CPU (bit 0 is set). 00: Instruction fetch, 01: Data fetch or data store, 10: MMU access, 11: Reserved
3	Multiple AHB errors occurred for accesses made by other AHB masters. Write 1 to clear.
2	Multiple AHB errors occurred for accesses made by this processor. Write 1 to clear.

Table 2055. LEON5 AHB error register

1	AHB error occurred for access made by other AHB master. Write 1 to clear.
0	AHB error occurred for access made by this processor. Write 1 to clear.

104.12.11 AHB error address register

The AHB error register is located at ASI 0x2 address 0x24. It contains the 32-bit address on the AHB bus for the captured error that is reported in the AHB error register.

104.12.12 Cache trap register

The cache trap register is located at ASI 0x2 address 0x2C. It contains some status information on the traps that are generated by the cache controller.

Table 2056. LEON5 cache trap register

31 26	25 22	21 18	17 16
RESERVED	CSACC	CSPEND	CSSRC
0	nr	0000	nr
r	wc	wc	r

15 13	12 11	10 9	8	7 5	4 2	1 0
RESERVED	SELOST	SEPEND	SESRC	IEXLOST	IEXPEND	IEXSRC
0	00	00	nr	000	000	nr
r	wc	wc	r	wc	wc	r

31 : 26	Reserved
25 : 22	Cache state error accumulated traps. Bit mask indicating which cache state errors have occurred. Bit positions matching the cache state error source field. This field is not reset.
21 : 18	Cache state error trap pending. Bit mask indicating which cache state errors have occurred but where the trap has not yet been delivered. Bit positions matching the cache state error source field.
17 :16	Cache state error source. This field is set after a cache error trap to indicate the cause of the trap. 00: Duplicated tag in instruction cache 01: Duplicated tag in data cache 10: Uncorrectable error occurred and configured to trap when this happens (FT only). 11: Unrecoverable late error detected (FT only)
15 : 13	Reserved
12 : 11	Data store error lost. Bit mask indicating data store errors that did not cause a trap depending on store error mode configuration.
10 : 9	Data store error pending: Bit mask indicating data store error traps that have not yet been delivered. Bit positions matching the data store error source field.
8	Data store error source. This field is set after a data_store_error trap to indicate the cause of the trap. 0: AHB error on write of data from store instruction 1: AHB error during MMU PTE writeback operation
7 : 5	Instruction access exception lost traps. Bit mask indicating instruction access exceptions that did not cause a trap. Bit positions matching the instruction access exception source field
4 : 2	Instruction access exception pending. Bit mask indicating instruction access exceptions that have not yet caused a trap. Bit positions matching the instruction access exception source field
1 : 0	Instruction access exception source. The field is set after an instruction_access_exception trap to indicate the source of the trap. 00: AHB error during instruction featch 01: MMU lookup error 10: TCM permission error 11: Reserved

104.12.13Boot word register

The boot word register is located at ASI 0x2 address 0x3C. It contains a 32-bit word that can be set by the monitoring processor through the CPU-to-CPU interface. The interpretation of the word is to be defined by software.

104.12.14TCM configuration register

The TCM configuration register is at ASI 0x2 address 0x40. It is only implemented if the ITCME or DTCME bits in the LEON5 configuration register 2 is set, and should not be accessed unless these bits have been checked.

*Table 2057.*LEON5 TCM configuration register

31	30 24	23 21	20 16
ITWP	RESERVED	ITCMFRAC	ITCMSIZE
0	0	*	*
rw	r	r	r

15	14 8	7 5	4 0
DTWP	RESERVED	DTCMFRAC	DTCMSIZE
0	0	*	*
rw	r	r	r

31	ITCM wipe (ITWP). Set to 1 to clear the contents of the instruction TCM. Reads 1 if wipe is in progress, otherwise 0.
29:24	Reserved
23:21	Instruction TCM fraction. 0: fully occupied, 1: 9/16, 2: 10/16, 3: 11/16, ..., 7: 15/16
20:16	Instruction TCM size in power of 2 of bytes. 0:unimplemented, 1-3: unused, 4: 16 bytes, 5: 32 bytes, ...
15	DTCM wipe (DTWP). Set to 1 to clear the contents of the data TCM. Reads 1 if wipe is in progress, otherwise 0.
14:9	Reserved
8:5	Data TCM fraction. 0: fully occupied, 1: 9/16, 2: 10/16, 3: 11/16, ..., 7: 15/16
4:0	Data TCM size in power of 2 of bytes. 0:unimplemented, 1-3: unused, 4: 16 bytes, 5: 32 bytes, ...

104.12.15Instruction TCM control register

The Instruction TCM control register is at ASI 0x2 address 0x48. It is only implemented if the ITCME or DTCME bits in the LEON5 configuration register 2 is set, and should not be accessed unless these bits have been checked.

*Table 2058.*LEON5 instruction TCM control register

31	16
INSTRUCTION TCM ADDR(31:16)	
0000000000000000	
rw	

15	8	7	5	4	3	2	1	0
INSTRUCTION TCM CONTEXT		RESERVED		ITXS	ITXU	ITEVA	ITEVC	ITEP
00000000		0		0	0	0	0	0
rw		r		rw	rw	rw	rw	rw

31 : 16	Instruction TCM address. Configures on which address as seen by the CPU the instruction TCM will be mapped. The TCM must always be mapped to a multiple of its own size (rounded up to nearest power of two if using fractional size), therefore if the instruction TCM is larger than 64 KiB, the lowest bits of this field will be forced to constant zero.
15:8	Context ID used when the ITENC bit is set to 1.
7:5	Reserved
4	Set to 1 to allow supervisor execution from the instruction TCM, if set to 0 attempts to jump to the TCM in supervisor mode will trigger an instruction_access_exception.
3	Set to 1 to allow user mode execution from the instruction TCM, if set to 0 attempts by jump to the TCM in user mode will trigger an instruction_access_exception.
2	Instruction TCM enable in MMU-enabled mode, on all contexts
1	Instruction TCM enable in MMU-enabled mode, on specified context only
0	Instruction TCM enable in MMU-off mode

104.12.16Data TCM control register

The Data TCM control register is at ASI 0x2 address 0x4C. It is only implemented if the ITCME or DTCME bits in the LEON5 configuration register 2 is set, and should not be accessed unless these bits have been checked.

*Table 2059.*LEON5 instruction TCM control register

31								16
DATA TCM ADDR(31:16)								
0000000000000000								
rw								
15	8	7	5	4	3	2	1	0
DATA TCM CONTEXT		RES	DTWS	DTWU	DTRS	DTRU	DTEVA	DTEVC	DTEP
00000000		0	0	0	0	0	0	0	0
rw		r	rw	rw	rw	rw	rw	rw	rw

31 : 16	Data TCM address. Configures on which address as seen by the CPU the data TCM will be mapped. The TCM must always be mapped to a multiple of its own size (rounded up to nearest power of two if using fractional size), therefore if the data TCM is larger than 64 KiB, the lowest bits of this field will be forced to constant zero.
15:8	Context ID used when the DTEVC bit is set to 1.
7:5	Reserved
6	Set to 1 to allow supervisor writes to the data TCM, if set to 0 attempts to store into the TCM in supervisor mode will trigger a data_access_exception. Note that this only affects regular stores, store-alternate to the data TCM ASI are always allowed regardless of this setting.
5	Set to 1 to allow user mode writes to the data TCM, if set to 0 attempts by store to the TCM in user mode will trigger a data_access_exception.
4	Set to 1 to allow supervisor reads from the data TCM, if set to 0 attempts to load from the TCM in supervisor mode will trigger a data_access_exception. Note that this only affects regular loads, load-alternate from the data TCM ASI are always allowed regardless of this setting.
3	Set to 1 to allow user mode reads from the data TCM, if set to 0 attempts by load from the TCM in user mode will trigger a data_access_exception.
2	Instruction TCM enable in MMU-enabled mode, on all contexts
1	Instruction TCM enable in MMU-enabled mode, on specified context only
0	Instruction TCM enable in MMU-off mode

104.12.17FSR register

The floating-point state register (fsr) is implemented according to the SPARC V8 specification. It can be accessed via the LDFSR/STFSR instructions as defined in SPARC. It can also be accessed via the FPU diagnostic interface, ASI 0x20 address 0x40.

*Table 2060.*LEON5 Floating-point state register (%fsr)

31	30	29	28	27	23	22	21	20	19	17	16
RD		RESERVED		TEM		NS	RESERVED		V ER		FTT(2)
00		0		00000		0	0		*		0
rw		r		rw		rw	r		r		r*

15	14	13	12	11	10	9	5	4	0
FTT(1:0)		QNE	RES	FCC		AEXC		CEXC	
00		0	0	00		00000		00000	
r*		r	r	rw		rw		rw	

31:30	Rounding direction (RD)
29:28	Reserved

*Table 2060.*LEON5 Floating-point state register (%fsr)

27:23	Trap enable mask (TEM)
22	Non-standard mode (NS)
21:20	Reserved
19:17	Version (VER) - Reads 4 (100) when GRFPC5 is used and 5 (101) when NanoFPU is used.
16:14	Floating-point trap type (FTT). Automatically clears when STFSR is issued. Read-only when using LDFSR/STFSR but can be modified via diagnostic reigster interface.
13	Queue not empty (QNE)
12	Reserved
11:10	Floating-point condition code (FCC)
9:5	Accrued exceptions (AEXC)
4:0	Current exception (CEXC)

104.12.18LEON5 Floating-point controller configuration register

The LEON5 floating-point controller configuration register is located at ASI 0x20 offset 0x50.

*Table 2061.*LEON5 Floating-point controller configuration register

31 29	28 16
VER	RESERVED
*	0
r	r

15 8	7 0
RESERVED	DENORMCFG
0	01010101
r	*

31:29	Version (VER) - Reads 4 (100) when GRFPC5 is used and 5 (101) when NanoFPU is used.
28:8	Reserved
7:6	Handling of denormalized operands and outputs: 7:6 Handling of denormal results when non-standard bit (FSR.NS) is set to 1. 5:4 Handling of denormal operands when non-standard bit (FSR.NS) is set to 1 3:2 Handling of denormal results when non-standard bit (FSR.NS) is set to 0 1:0 Handling of denormal operands when non-standard bit (FSR.NS) is set to 0 Values: 00:Trigger unfinished exception, 01:Handle according to IEEE standard, 10/11:Flush to zero Only writable when using GRFPC5, read-only when using NanoFPU. For IEEE standards compliance set to 01010101. For identical behavior to LEON3/LEON4 with GRFPU, set to 10101000

104.12.19MMU control register

The MMU control register is located in ASI 0x19 offset 0, and the layout can be seen in table 2062.

*Table 2062.*LEON5 MMU control register

31 28	27 24	23 21	20 18	17 16
IMPL	VER	ITLB	DTLB	PSZ
0	1	*	*	0
r	r	r	r	rw*

15	14	13 2	1	0
TD	ST	RESERVED	NF	E
NR	0	0	0	0
rw*	r	r	rw	rw

31:28	MMU Implementation ID. Hardcoded to “0000”

*Table 2062.*LEON5 MMU control register

27:24	MMU Version ID. Hardcoded to “0001”.
23:21	Number of ITLB entries. The number of ITLB entries is calculated as 2^{ITLB}. If the TLB is shared between instructions and data, this field indicates to total number of TLBs.
20:18	Number of DTLB entries. The number of DTLB entries is calculated as 2^{DTLB}. If the TLB is shared between instructions and data, this field is zero.
17:16	Page size. The size of the smallest MMU page. 0 = 4 Kib; 1 = 8 Kib; 2 = 16 Kib; 3 = 32 Kib. If the page size is programmable, this field is writable, otherwise it is read-only.
15	TLB disable. When set to 1, the TLB will be disabled and each data access will generate an MMU page table walk.
14	Separate TLB. This bit is set to 1 if separate instruction and data TLBs are implemented
13	This bit only exists if *mmuen* generic is set to 2. This bit is written to the SO (supervisor only access) part of the TAG during diagnostic writes.
13:2	Reserved for future implementations
1	No Fault. When NF= 0, any fault detected by the MMU causes FSR and FAR to be updated and causes a fault to be generated to the processor. When NF= 1, a fault on an access to ASI 9 is handled as when NF= 0; a fault on an access to any other ASI causes FSR and FAR to be updated but no fault is generated to the processor.
0	Enable MMU. 0 = MMU disabled, 1 = MMU enabled.

104.12.20MMU context pointer and context registers

The MMU context pointer register is located in ASI 0x19 offset 0x100 and the MMU context register is located in ASI 0x19 offset 0x200. They together determine the location of the root page table descriptor for the current context. Their definition follows the SRMMU specification in the SPARC V8 manual with layouts shown below..

*Table 2063.*LEON5 MMU context pointer register

31 2	1 0
CONTEXT TABLE POINTER	RES
NR	0
rw	r

31:2	Context table pointer, physical address bits 35:6 (note address is shifted 4 bits)
1:0	Reserved, always 0

*Table 2064.*LEON5 MMU context register

31 8	7 0
RESERVED	CONTEXT
0	0
r	rw

31:8	Reserved
7:0	Current context ID

In the LEON5, the context bits are OR:ed with the lower MMU context pointer bits when calculating the address, so one can use less context bits to reduce the size/alignment requirements for the context table.

104.12.21MMU fault status register

The MMU fault status register is located in ASI 0x19 offset 0x300, and the definition is based on the SRMMU specification in the SPARC V8 manual. The SPARC V8 specifies that the fault status regis-

ter should be cleared on read, on the LEON5 only the FAV bit is cleared on read. The FAV bit is always set on error in the LEON5 implementation, so it can be used as a valid bit for the other fields..

Table 2065. LEON5 MMU fault status register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED														EBE	
0														0	
r														r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EBE						L		AT			FT			FAV	OW
0						0		0			0			0	0
r						r		r			r			r	r

31:18	Reserved
17:10	External bus error (EBE) - Never set on the LEON3
9:8	Level (L) - Level of page table entry causing the fault
7:5	Access type (AT) - See V8 manual
4:2	Fault type (FT) - See table 2030
1	Fault address valid (FAV) - Cleared on read, always written to 1 on fault
0	Overwrite (W) - Multiple faults of the same priority encountered

104.12.22MMU fault address register

The MMU fault address register is located in ASI 0x19 offset 0x400, and the definition follows the SRMMU specification in the SPARC V8 manual..

*Table 2066.*LEON5 MMU fault address register

31 12	11 0
FAULT ADDRESS	RESERVED
NR	0
r	r

31:12	Top bits of virtual address causing translation fault
11:0	Reserved, always 0

104.13 Fault tolerance registers

104.13.1Overview

In fault tolerant configuration, some additional configuration and status registers become available in ASI 0x2. These registers are only defined if the FTEN bit is set to 1 in the L5CFG2 register (see section 104.12.3). Note that also for FT releases, these registers are only inplemented if ECC features are actually enabled in the processor at build time via either the cmemconf or rfconf generic.

104.13.2FT configuration register

The FT configuration register is located on ASI 0x2 address 0x14.

*Table 2067.*LEON5 FT configuration register (L5FTCFG)

31 30	29 28	27 23	22	21	20 19	18 17	16
CACHEFT	RFFT	RESERVED	RFUEM	RFSEM	CUEM	CCEM	SCREN
*	*	0	0	0	00	00	0
r	r	r	rw	rw	rw	rw	rw

15 0
SCRUBPER
0000000011111111
rw

31:30	Cache memory fault tolerance configuration. 0:none, 1:FPGA builtin error correction, 2:RTL/ASIC error correction, 3:reserved for future use
29:28	Register file fault tolerance configuration. 0:none, 1:FPGA builtin error correction, 2:RTL/ASIC error correction, 3:reserved for future use
22	Register file FPGA uncorrectable error handling mode. (used in RFFT=1 configuration only) 0: pipeline restart with writeback from redundant copy 1: raise cache internal error exception
21	Register file FPGA correctable error handling mode. 0:pipeline restart with writeback from redundant copy 1:on the fly correction, no restart/writeback (valid in RFFT=1 configuration only)
20 : 19	Cache memory uncorrectable error handling mode. 0: flush only the affected cache line (for data error) or set (for tag error) and resume 1: flush entire affected cache 2: raise cache internal error exception 3: flush affected cache and raise cache internal error exception
18 : 17	Cache memory correctable error handling mode. 0: correct error, writeback and resume 1: correct data on the fly, no writeback (valid in CACHEFT=1 configuration only) 2: reserved for future use 3: flush entire cache and resume
16	Enable hardware scrubber for cache memory and register file
15 : 0	Scrub interval, in cycles, minus one. See section 104.6.5 for some guidelines on how to set this field.

104.13.3TCM FT configuration register

The TCM FT configuration register is located at ASI 2 offset 0x44. It is only available if both FTEN bit and either ITCME or DTCME bits are set in LEON5 configuration register 2.

*Table 2068.*LEON5 TCM FT configuration register (L5TCMFTCFG)

31	16
RESERVED	
0	
r	

15	10	9	8	7	3	2	1	0
RESERVED		ITUEM	ITCEM	RESERVED		DCUEV	DCUEM	DCCEM
0		0	0	0		0	0	0
r		rw	rw	r		rw	rw	rw

31 : 10	Reserved for future implementations
9	Instruction TCM uncorrectable error handling mode 0: replace data with all-zeros (UNIMP instructions) 1: raise cache internal error exception
8	Instruction TCM correctable error handling mode. 0: correct error, writeback and resume 1: correct data on the fly, no writeback (valid in CACHEFT=2 configuration only)
7 : 3	Reserved for future implementations
2	Data TCM uncorrectable error replacement value 0: replace failing locations with 0x00000000 data 1: replace failing locations with 0xffffffff data
1	Data TCM uncorrectable error handling mode. 0: replace data with all-ones or all-zeros depending on DCUEV field 1: raise cache internal error exception
0	Data TCM correctable error handling mode. 0: correct error, writeback and resume 1: correct data on the fly, no writeback (valid in CACHEFT=2 configuration only)

104.13.4Cache error injection register

The cache error injection register is located on ASI 0x2 address 0x30. This register is used together with the diagnostic cache ASI to perform error injection.

*Table 2069.*LEON5 cache error injection register (L5ERRINJ)

31 30	29 16
EISUPP	RESERVED
*	r
r	0

15 14	13 8	7 2	1	0
RESERVED	EIBIT2	EIBIT1	EIUE	EIEN
r	rw	rw	rw	rw
0	*	*	*	*

31:30	Error injection support: 0:Error injection not supported 1:Can inject correctable error only, in unspecified bit 2:Can inject uncorrectable error or correctable error, in unspecified bits 3:Can inject in arbitrary bits selected via EIBIT1,EIBIT2 fields
29 : 14	Reserved for future implementations
13 : 8	Error injection bit number 2. The bit selected by this index will be inverted in error injection in addition to bit selected with EIBIT1 field. Should be set to the same value as EIBIT1 when injecting single bit error. Field is only implemented if EISUPP=3
7 : 2	Error injection bit number. The bit selected by this index will be inverted in error injection. Field is only implemented if EISUPP=3
1	Inject uncorrectable error. Only implemented if EISUPP=2
0	Enable error injection. The next read access to diagnostic ASIs 0xC,0xD,0xE,0xF,0x1D,0x26,0x27 will cause an error injection to the read address.

104.13.5Scrubber position register

The scrubber position register is located on ASI 0x2 address 0x34. It provides read-only access to the current scrubber position and pending state, for diagnostic purposes.

*Table 2070.*LEON5 scrubber position register (L5SCRPOS)

31	30	29 16
CSCRP	SSCRP	RESERVED
0	0	0
r	r	´r

15 0
SCRUBPOS
0000000000000000
r

31	Scrub pending to cache tags, cache data and register file
30	Scrub pending to snoop tags
29 : 16	Reserved for future implementations
15 : 0	Scrubber position, determines the next cache line and register to scrub

104.13.6Error counter register

The error counter register is located on ASI 0x2 address 0x38. It provides access to read and clear error correction counters. The counters are saturating and will stop incrementing when reaching the

maximum value. This register is not reset in order to allow diagnosis following a reset and may therefore have random values after first power up.

*Table 2071.*LEON5 error counter register (L5ERRCTR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED		NLUE	RLUE	ICUE	DCUE	ICCEC		DCCEC		RFRESC		RFDEC	RFSEC		CC(1)
0		(nr)	(nr)	(nr)	(nr)	(nr)		(nr)		(nr)		(nr)	(nr)		(nr)
´r		wc	wc	wc	wc	wc		wc		wc		wc	wc		wc

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC(0)	ITUEC	IDUEC	DLUEC	DSUEC	DDUEC	ITCEC		IDCEC		DTCEC		STCEC		DDCEC	
(nr)	(nr)	(nr)	(nr)	(nr)	(nr)	(nr)		(nr)		(nr)		(nr)		(nr)	
wc	wc	wc	wc	wc	wc	wc		wc		wc		wc		wc	

31 : 30	Reserved for future implementations
29	Non-recovered late uncorrectable error counter
28	Recovered late uncorrectable error counter
27	Instruction TCM uncorrectable error counter
26	Data TCM uncorrectable error counter
25 : 24	Instruction TCM correctable error counter
23 : 22	Data TCM correctable error counter
21 : 20	Register file restart counter
19	Register file double error counter
18 : 17	Register file single error counter
16 : 15	Cache correction cycle counter
14	Instruction cache tag uncorrectable error counter
13	Instruction cache data uncorrectable error counter
12	Data cache lookup tag uncorrectable error counter
11	Data cache snoop tag uncorrectable error counter
10	Data cache data uncorrectable error counter
9 : 8	Instruction cache tag correctable error counter
7 : 6	Instruction cache data correctable error counter
5 : 4	Data cache lookup tag correctable error counter
3 : 2	Data cache snoop tag correctable error counter
1 : 0	Data cache data correctable error counter

104.14 Block redundancy registers

104.14.1Overview

In block redundant configuration, some additional configuration and status registers become available in the cache controller plus some registers are added to the debug module. These registers are only defined if the BREN bit is set to 1 in the L5CFG2 register (see section 104.12.3). Note that also for FT releases, these registers are only inplemented if block-redundancy features are actually enabled in the processor at build time via the redconf generic.

104.14.2Block redundancy configuration register

The block redundancy configuration register is located at ASI 0x2, address 0x60. The layout of this register mirrors the structure of the redconf generic.

*Table 2072.*LEON5 block redundancy configuration register (L5BRCFG)

31															16
RESERVED															
0															
r															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED		MFRF	MFPU	MCM	MIRF	FRFRED		FPURED		CMRED		IRFRED		PLRED	
0		*	*	*	*	*		*		*		*		*	
r		r	r	r	r	r		r		r		r		r	

31:14	Reserved for future use
13	Merged fault domains for floating-point register file and pipeline (1=merged, 0=separate)
12	Merged fault domains for GRFPU5 and pipeline (1=merged, 0=separate)
11	Merged fault domains for cache and pipeline (1=merged, 0=separate)
10	Merged fault domains for integer register file and pipeline (1=merged, 0=separate)
9 : 8	Redundancy for floating-point register file 0: single, 1:dual-redundant, 2:triple-redundant, 3:reserved Only valid for configurations where GRFPU5 is used.
7 : 6	Redundancy for floating-point register file 0: single, 1:dual-redundant, 2:triple-redundant, 3:reserved Only valid for configurations where GRFPU5 is used.
5 : 4	Redundancy for cache memory and TCM 0: single, 1:dual-redundant, 2:triple-redundant, 3:reserved
3 : 2	Redundancy for integer register file 0: single, 1:dual-redundant, 2:triple-redundant, 3:reserved
1 : 0	Redundancy for processor pipeline, MMU, cache controller, and floating-point controller. 0: single, 1:dual-redundant, 2:triple-redundant, 3:reserved

104.14.3Block redundancy error state register

The block redundancy error state register is located at ASI 0x2, address 0x64. The register contains status bits indicating which errors have been detected, and a status bit used for trap handling. Note that the terms uncorrectable and correctable errors in this register refers to miscompares at the block boundaries and should not be confused with uncorrectable and correctable ECC errors.

*Table 2073.*LEON5 block redundancy error state register (L5BRERR)

31	30:20	19	18:16
RTRT	RESERVED	BRUFR	BRCFR
0	0	0	000
wc	r	wc	wc

15	14:12	11	10:8	7	6:4	3	2:0
BRUFP	BRCFP	BRUCM	BRCCM	BRUIR	BRCIR	BRUPL	BRCPL
0	000	0	000	0	000	0	000
wc	wc	wc	wc	wc	wc	wc	wc

31	Redundancy trap taken. This bit is set to 1 when the redundancy error trap is taken by the processor, and further traps will be masked until this bit has been cleared. Write 1 to clear.
30:20	Reserved for future use
19	Block-redundancy uncorrectable error flag for floating-point register file. Set to 1 indicate an uncorrectable error occurred. Write 1 to clear.
18:16	Block-redundancy correctable error mask for floating-point register file. Bits set to 1 indicate a miscompare has been detected for one of the redundant copies. In dual-redundant mode, only bit 0 is used. Write 1 to clear
15	Block-redundancy uncorrectable error flag forGRFPU5. Set to 1 indicate an unecorrectable error occurred. Write 1 to clear
14:12	Block-redundancy correctable error mask for GRFPU5. Bits set to 1 indicate a miscompare has been detected for one of the redundant copies. In dual-redundant mode, only bit 0 is used. Write 1 to clear
11	Block-redundancy uncorrectable error flag for cache memory and TCM. Set to 1 indicate an uncorrectable error occurred. Write 1 to clear
10:8	Block-redundancy correctable error mask for cache memory and TCM. Bits set to 1 indicate a miscompare has been detected for one of the redundant copies. In dual-redundant mode, only bit 8 is used. Write 1 to clear
7	Block-redundancy uncorrectable error flag for integer register file. Set to 1 indicate an unecorrectable error occurred. Write 1 to clear
6:4	Block-redundancy correctable error mask for integer register file. Bits set to 1 indicate a miscompare has been detected for one of the redundant copies. In dual-redundant mode, only bit 4 is used. Write 1 to clear
3	Block-redundancy uncorrectable error flag for processor pipeline, MMU, cache controller, and floating-point controller. Set to 1 indicate an uncorrectable error occurred. Write 1 to clear.
2:0	Block-redundancy correctable error mask for processor pipeline, MMU, cache controller, and floating-point controller. Bits set to 1 indicate a miscompare has been detected for one of the redundant copies. In dual-redundant mode, only bit 0 is used. Write 1 to clear

104.14.4Block redundancy trap configuration register

The block redundancy trap configuration register is located at ASI 0x2, address 0x68. The register configures under which conditions the block redundancy trap should be delivered to the processor.

*Table 2074.*LEON5 block redundancy trap configuration register (L5BRTRAP)

31											16
RESERVED											
0											
r											
15	10	9	8	7	6	5	4	3	2	1	0
RESERVED		TUFR	TCFR	TUFP	TCFP	TUCM	TCCM	TUIR	TCIR	TUPL	TCPL
0		0	0	0	0	0	0	0	0	0	0
r		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31:10	Reserved for future use
9	Enable trap on block-redundancy uncorrectable error for floating-point register file
8	Enable trap on block-redundancy correctable error for floating-point register file
7	Enable trap on block-redundancy uncorrectable error for GRFPU5
6	Enable trap on block-redundancy correctable error for GRFPU5
5	Enable trap on block-redundancy uncorrectable error for cache memory and TCM
4	Enable trap on block-redundancy correctable error for cache memory and TCM
3	Enable trap on block-redundancy uncorrectable error for integer register file
2	Enable trap on block-redundancy correctable error for integer register file
1	Enable trap on block-redundancy uncorrectable error for pipeline
0	Enable trap on block-redundancy correctable error for pipeline

104.14.5Pipeline reboot data stash

When the pipeline is build block-redundant (the PLRED field in the block-redundancy configuration register is non-zero), the debug module adds a 64-bit stash area per CPU core that can be accessed via ASI 0x22, addresses 0xF8 and 0xFC. The stash area can hold any data to be preserved across a pipeline self-reboot, and can be accessed either with 32-bit or with 64-bit load/store alternate instructions.

104.15 CPU-to-CPU interface registers

104.15.1Overview

The CPU to CPU interface provides a set of registers shared between the processors. Each processor gets a unique view into the interface through loads and stores to ASI 0x22.

Note that all processor indexes in the CPU to CPU interface are relative, so CPU0 refers to the processor that is accessing the interface, CPU1 to the processor with the next higher index, and so on. The purpose of this is to allow making software that does not have to know the absolute processor IDs it is running on.

The layout of the register interface has been designed to allow systems with potentially up to 64 processors, therefore many functions are covered with multiple 32-bit registers. Only the first of each register needs to be accessed for systems with 16 or less CPUs (or where each CPU only needs to control at most 15 other processors).

104.15.2Effective control bitmap

The read-only registers at offsets 0x0 (for CPU 0-31) and 0x4 (for CPU 32-63) provide the part of the effective control matrix, showing which CPUs the reading processor can control and monitor. The effective control matrix takes into account both the direct permissions matrix and any delegation of control currently enabled through the delegation matrix.

104.15.3Delegation bitmap

Registers at offsets 0x8 (for CPU 0-31) and 0xC (for CPU 32-63) allow the accessing processor to configure it's part of the delegation matrix. By setting the delegation setting to 1 for CPU #n and 0 for CPUs #(n+1) up to #(n+k), CPU #n will be allowed to control CPU (n+1)..(n+k).

104.15.4CPU state read-out

Read-only registers at offsets 0x10 (CPU 0-15), 0x14 (CPU16-31), 0x18 (CPU32-47), 0x1C (CPU48-63) provide access to the current two-bit CPU state according to the model in section 104.9.4.

Processors that the reading processor does not have permission to access will report a state of 00 regardless of actual current state.

104.15.5Reboot request reset address

The reboot request reset address at offset 0x20 holds the address to be used for the next reboot command issued by this processor.

104.15.6Reboot request control and status

The reboot request control and status register at offset 0x24 is used to command a reboot of another CPU, and to monitor the progress of the reboot request.

104.15.7Legacy interface control register

The legacy interface control register at offset 0x28 is only available for CPU0. It contains one bit to disable the legacy, IRQ controller based, boot interface.

The bit can only be set through the CPU-to-CPU interface, clearing it requires a reset or access through the debug port.

104.15.8SMP broadcast group register

The SMP broadcast group register at offset 0x30 and 0x34 allows the processor to define an SMP broadcast group. The group may include the processor itself and any of the processors it has permissions to control. Any SMP broadcast made by any of the processors in the group will reach the other processors in the same group.

104.15.9SMP broadcast configuration

The SMP broadcast configuration register at offset 0x38 allows the processor to mask any incoming SMP broadcast from other members of the SMP group.

104.15.10SMP broadcast request

The SMP broadcast request register at offset 0x3C allows the processor to broadcast commands to the other processors of the same SMP group.

104.15.11Processor control commands

The set of registers at 0x40 up to 0x6C allows the processor to send start, stop and wakeup commands to the other processors.

104.15.12Boot word access

The set of registers at 0x100 (for CPU relative index 0, the processor itself) up to 0x1FC (CPU relative index 63) allows the processor to read and write the "boot word" registers of other processors.

The processor allowed to access its own boot register as well as those of the processors that it is allowed to control.

104.16 Debug module registers

104.16.1Overview

The debug module's register interface is only accessible from debug masters.

Note that backward compatibility is not guaranteed for this interface. Compatibility breaking changes are indicated by incrementing the version number of the DSU5 entry in the AHB plug'n'play information that is visible for the debug masters.

104.16.2Memory map

Registers in the debug module are either global, per CPU, or per session.

*Table 2075.*Debug module registers address overview

[27:22]	[21:19]	[18:9]	[8]	[7]	[6]	[5:2]	Description
zero	000	zero	0	0	register		DSU registers, global
cpu index	000	zero	0	1	0	register	DSU registers, per CPU
session index	000	zero	0	1	1	register	DSU registers, per session
zero	000	zero	1	0	register		AHB tracing registers
zero	000	zero	1	1	register		Instruction trace registers
cpu index	001	address					Instruction trace buffer
zero	010	address					AHB trace buffer
cpu index	011	address					Integer register file
cpu index	100	zero		0	0	register	IU special registers
cpu index	100	zero		0	1	register	ASR registers asr16-31
cpu index	100	zero		1	0	0000	Pipeline debug register 1
cpu index	100	zero		1	1	0000	Pipeline debug register 2
cpu index	101	zero				0000	Single step control
cpu index	101	zero				0001	Single step counter
cpu index	110	zero					Pipeline debug register 3
cpu index	111	address					Cache controller ASI access

104.17 Software considerations

104.17.1Register file initialization on power up

It is recommended that the boot code for the processor writes all registers in the IU and FPU register files before launching the main application. This allows software to be portable to both FT and non-FT versions of the LEON3, LEON4 and LEON5 processors.

104.17.2LEON3 and LEON4 compatibility

The LEON5 subsystem has been designed to provide a high degree of binary compatibility with existing LEON3 and LEON4 software:

- Backward compatible layout of registers and feature detection bits
- Compatible assignments of ASIs and ASRs'
- Support for V9 CASA instruction as for LEON3/LEON4

- Similar memory map and peripherals as LEON3/LEON4 systems
- Support for the standard total store order model

Potential areas where compatibility problems may arise:

- Some software may depend on explicit checking of the PSR.VER field and not work as intended when it detect the value 5 rather than 3.
- Any code written that assumes every instruction takes at least of one cycle, for example using NOP instructions to create a certain number of cycles delay, may not work as expected due to the dual issue capability of LEON5.
- Code that is accessing the diagnostic ASIs to read cache tags have to be adjusted to account for that the cache lines are physically tagged on LEON5.

104.18 Vendor and device identifiers

The core has vendor identifiers 0x01 (Frontgrade Gaisler) and device identifiers 0x0BA for the LEON5 AHB master interface on the processor bus, and 0xBB for the LEON5 debug support unit (register area) on the debug ports. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

104.19 Implementation

104.19.1Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_async_reset_enable* is set.

104.19.2Technology mapping

LEON5 has two technology mapping VHDL generics, *fabtech* and *memtech*. The *fabtech* generic controls the implementation of some pipeline features, while *memtech* selects which memory blocks will be used to implement cache memories and the IU/FPU register file. *Fabtech* can be set to any of the provided technologies (0 - NTECH) as defined in the TECHMAP.GENCOMP package. See the GRLIB Users's Manual for available settings for *memtech*.

The LEON5 core maps all usage of RAM memory on the *syncram*, *syncram_2p* and *syncram_dp* components from the technology mapping library (TECHMAP).

104.19.3Register file implementation

The LEON5 requires an integer register file with 4 read ports and 2 write ports. The register file can be configured to use either a flip-flop based implementation or an implementation based on two-port RAMs. The two-port RAM configuration will use a total of 16 syncram_2p instances.

The floating-point register file always uses a flip flop based implementation.

104.19.4RAM usage

The instruction cache will use only single-port syncram instances for both tags and data.

The data cache tags can be configured into three different modes depending on the cmemconf generic. Depending on configuration it will use either syncram, syncram_2p or syncram_dp instantiations and

valid bits will be either included in the tag or held separately in flip flops. The different modes are functionally equivalent.

The data cache data memory is always implemented using single-port memories. It can be configured to use memories with individual byte-write stobes if such are available in the technology. When byte strobes are not available, byte/half-word updates are handled by forwarding back read data from the same memory which may impact timing on some technologies.

The LEON5 makes use of the “read-hold” feature of the RAM techmap and assumes the memory keeps its current data output when disabled in order to save power and reduce logic. If this is not the case, emulation logic inside the techmap will be added to provide the equivalent functionality.

104.19.5Clock gating

If the cgen generic is set to 1, the processors will support block-level clock gating of the processors while in power-down mode. The clock for each CPU is supplied separately on the gclk input of the subsystem, and a gclken signal indicates when the clock should be enabled. The gclken may be pipelined externally if needed.

104.19.6Block redundancy concerns

When block redundancy is used, care needs to be taken to ensure that redundant copies are not optimized out by the synthesis tools.

Some attributes specific to Synplify have been set in the source code when block-redundancy is enabled to achieve this:

syn_keep is set to true on signals between the different sub-parts of the processor core.

syn_hier is set to “hard” on the top of each sub-part to prevent logic optimization across the block boundary.

syn_radhardstyle attribute is set to “none” on the top of the entities holding dual- and triple-redundant copies. This is done to override any global radhardstyle setting, to allow a mix of block-TMR and local TMR.

104.20 Configuration options

Table 2076 shows the configuration options of the core (VHDL generics).

*Table 2076.*Configuration options

Generic	Function	Allowed range	Default
fabtech	Target technology	0 - NTECH	0 (inferred)
memtech	Vendor library for regfile and cache RAMs. Bits 17 and 18 of this generic can be used to for the IU register file and FP register file to inferred technology: + 2**17: Force inferred technology for IU register file + 2**18: Force inferred technology for FP register file	0 - 16#FFFFFFFF#	0 (inferred)
ncpu	Number of processors instantiated in subsystem	1 - 16	1
nextmst	Number of external masters connected to processor AHB bus Note due to limitations of VHDL, this generic must be at least 1. If no external AHB masters are implemented, select one master and use the ahbm_none constant.	1 - NAHBMST	1
nextslv	Number of external slaves connected to processor AHB bus	1 - NAHBSLV	1
nextapb	Number of external slaves connected to APB bus	0 - NAPBSLV	0

*Table 2076.*Configuration options

Generic	Function	Allowed range	Default
ndbgmst	Number of debug masters connected to debug ports Note due to limitations of VHDL, this generic must be at least 1. If no external debug masters are implemented, select one master and use the ahbm_none constant.	1 - NAHBMST	
cached	Fixed cacheability mask. See sections 104.3 and 104.3.16 for more information.	0 - 16#FFFF#	0
wbmask	Wide-bus mask. Indicates which address ranges are 64/128 bit capable. Treated as a 16-bit vector with LSB bit (right-most) indicating address 0 - 0x10000000. When busw is set to 32, this setting is ignored. See section 104.3.16 for more information.	0 - 16#FFFF#	0
busw	Data bus width of the AHB bus. This controls the size of the accesses made to address areas marked as wide-access capable with the wbmask generic. This value must be set smaller or equal to the global AHBDW setting in GRLIB. For 32-bit only bus with no wide AHB slaves, set this value to 32. See section 104.3.16 for more information.	32,64, 128	64
memmap	Selects memory map, see section 104.11.1.	0-1	0
ahbsplit	Enable split response support in AHB controller	0-1	0
rfconf	Register file configuration: 0: Register file based on 2-port memories 1: Register file based on flip flops In releases with fault tolerance available, one of the following constants may be added to the rfconf value to configure fault tolerance when using memories: 16: FPGA builtin error correction on memories 32: RTL-based error correction on memories	0-1, 16, 32	0
cmemconf	Cache memory configuration. Sum of data cache tag configuration, data cache data configuration and fault tolerance configuration. Data cache tag configuration: 0 - two memories (one two-port, one one-port), valid bits in two-port memory 1 - one dual port memory, valid bits in flip flops 2 - two single port memories, valid bits in flip flops 3 - reserved Data cache data configuration: 0 - Use standard 32-bit one-port memories for data 4 - Use one-port memories with byte writes for data Fault tolerance configuration (FT releases only): 0 - No error correction 16 - FPGA builtin error correction on memories 32 - RTL-based error correction on memories	0-7, 16, 18, 32, 34	0
fpuconf	FPU configuration: 0: NanoFPU, one per processor 1: GRFPU5, one per processor	0 - 1	0

*Table 2076.*Configuration options

Generic	Function	Allowed range	Default
tcmconf	TCM configuration, sum of data TCM configuration and instruction TCM configuration. Data TCM configuration: 0: not implemented 1: 16 B, 2: 32 B, 3: 64 B, ..., 7: 1 KiB, 8: 2 KiB, ..., 29: 4 GiB 0: fully occuped, 32*1: 9/16, 32*2: 10/16, ..., 32*7: 15/16 Instruction TCM configuration: 0 - not implemented 256*1: 16 B, 256*2: 32 B, 256*3: 64 B, ..., 256*29 - 4 GiB 0: fully occuped, 256*32*1: 9/16, 256*32*2: 10/16, ..., 256*32*7: 15/16	0 - 65535	0
perfcfg	Performance configuration for cache and TLBs. 0: Highest performance 1: General purpose 2: Minimal See section 104.3.4.	0 - 2	0
redconf	Block redundancy configuration (FT releases only). Pipeline redundancy: 1*0 - single, 1*1 - dual, 1*2 - triple Integer regfile redundancy: 4*0 - single, 4*1 - dual, 4*2 - triple Cache memory redundancy: 16*0 - single, 16*1 - dual, 16*2 - triple GRFPU5 redundancy: 64*0 - single, 64*1 - dual, 64*2 - triple FPU register file redundancy: 256*0 - single, 256*1 - dual, 256*2 - single 2048 - merge pipeline and integer regfile fault domains 4096 - merge pipeline and cache memory fault domains 8192 - merge pipeline and GRFPU5 fault domains 16384 - merge pipeline and FPU register file fault domains	0 - 32767	0
mulimpl	Selects multiplier implementation 0: Inferred multiplier 1-2: Reserved 3: DesignWare multiplier	0 - 3	0
statcfg	Configures whether to include a statistics unit 0: no statistics unit 1: small unit 2: medium size unit 3: large size unit	0 - 3	0
breten	Selects whether to include the alternative boot address retry functionality (see section 104.8.4 for more information)	0 - 1	0
disas	Print instruction disassembly in VHDL simulator console. Does not affect synthesis output	0 - 1	0
ahbtrace	Enable AHB trace output in VHDL simulator console. Does not affect synthesis output	0 - 1	0

*Table 2076.*Configuration options

Generic	Function	Allowed range	Default
devid	Device ID for AMBA plug'n'play information table		0
cgen	Enable clock gating support, eacstath CPU will be clocked by individual clock on gclk input	0 - 1	0
scantest	Enable scan test support, propagates down into techmap syncram instantiations	0 - 1	0

104.21 Signal descriptions

Table 2077 shows the interface signals of the core (VHDL ports).

*Table 2077.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RSTN	N/A	Input	Reset	Low
GCLK[0:(NCPU-1)]	N/A	Input	Gateable clock per CPU, used only if cgen generic is set	
GCLKEN[0:(NCPU-1)]	N/A	Output	Clock enable per CPU to gate GCLK inputs	High
BRETCLK	N/A	Input	Clock used for alternative boot address retry function (only used if breten=1)	-
BRETRSTN	*	Input	Reset used for alternative boot address retry function (only used if breten=1)	Low
RSTREQN	*	Output	Request for reset from debug master (optional, see section X for details)	High
AHBMI	*	Output	AHB external master input signals	-
AHBMO[(NCPU+NEXTMST-1) : NCPU]	*	Input	AHB external master output signals	-
AHBSI	*	Output	AHB external slave input signals	-
AHBSO [(NEXTSLV-1):0]	*	Input	AHB external slave output signals	-
AHBPNP	N/A	Output	Vector of AHB Plug'n'play HCONFIG records for all slaves on the processor bus. This is needed by certain IP to determine cacheability of memory areas.	-
DBGMI[(NDBGMST-1):0]	*	Output	AHB debug master input signals Note: Unlike typical AHB signals, this is a separate input record per master. Each element must be connected to the corresponding master	-
DBGMO[(NDBGMST-1):0]	*	Input	AHB debug master output signals	-
APBI	*	Output	APB external slave input signals	-
APBO	*	Input	APB external slave output signals Note: Only a subset of the vector is used corresponding to elements 0:(NEXTAPB-1)	-
DSUEN	N/A	Input	DSU enable, allow debug master access	High
DSUBREAK	N/A	Input	DSU break. On reset, disables automatic start-up of CPU0. After reset, rising edge will make the debug module break the processors.	High
CPU0ERRN	N/A	Output	CPU0 error mode indication	Low
BRERR	N/A	Output	Block redundancy error indication	High

*Table 2077.*Signal descriptions

Signal name	Field	Type	Function	Active
SYSSTAT[15:0]	N/A	Input	Status vector for read-out from debug interface (optional)	-
DBGTSTOP	N/A	Output	Indicates that the timer is frozen during debugging (for freezing timers external to the subsystem)	High
DBGTIME[31:0]	N/A	Output	Value of internal cycle timer (for use in external trace buffers)	
UARTI	*	Input	Console UART input/receive signals	-
UARTO	*	Output	Console UART output/transmis signals	-
BRETIN	*	Input	Signals to configure the alternative boot address retry functionality. Only used if breten=1	-
BRETOUT	*	Output	Status signals for the alternative boot address retry functionality. Only used if breten=1	-
TESTEN	N/A	Input	The scan test signals are propagated into the AHB controller to be distributed via the AMBA records. See the section on scan test mode support in GRLIB IP Library User's Manual. These signals have default values in the component and can be left unconnected when the scantest generic is 0.	High
TESTRST	N/A	Input		Low
SCANEN	N/A	Input		High
TESTOEN	N/A	Input		-
TESTSIG[(1+GRLIB_CONFIG_ARRAY(grlib_techmap_testin_extra):0]	N/A	Input		-

* see GRLIB IP Library User's Manual

104.22 Library dependencies

Table 2078 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2078.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	LEON5	Component, signals	LEON5 component declaration, interrupt and debug signals declaration

105 LOGAN - On-chip Logic Analyzer

105.1 Introduction

The LOGAN core implements an on-chip logic analyzer for tracing and displaying of on-chip signals. LOGAN consists of a circular trace buffer and a triggering module. When armed, the logic analyzers stores the traced signals in the circular buffer until a trigger condition occurs. A trigger condition will freeze the buffer, and the traced data can then be read out via an APB interface.

The depth and width of the trace buffer is configurable through VHDL generics, as well as the number of trigger levels.

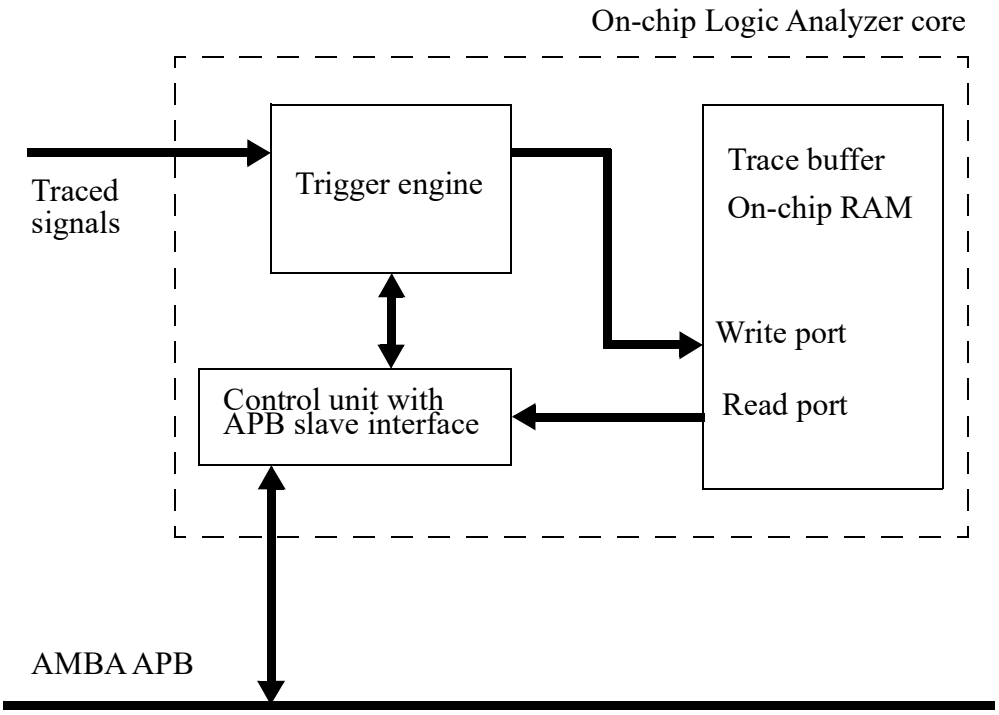


Figure 279. On-chip Logic Analyzer block diagram

105.2 Operation

105.2.1 Trace buffer

When the logic analyzer is armed, the traced signals are sampled and stored to the trace buffer on the rising edge of the sample clock (TCLK). The trace buffer consists of a circular buffer with an index register pointing to the next address in the buffer to be written. The index register is automatically incremented after each store operation to the buffer.

105.2.2 Clocking

LOGAN uses two clocks: TCLK and the APB clock. The trace signals are sampled on the rising edge of the sample clock (TCLK), while the control unit and the APB interface use the APB clock. TCLK and the APB clock does not need to be synchronized or have the same frequency.

105.2.3 Triggering

The logic analyzer contains a configurable number of trig levels. Each trig level is associated with a pattern and a mask. The traced signals are compared with the pattern, only comparing the bits set in the mask. This allows for triggering on any specific value or range. Furthermore each level has a match counter and a boolean equality flag. The equality flag specifies whether a match means that the pattern should equal the traced signals or that it should not be equal. It is possible to configure the trigger engine to stay at a certain level while the traced signals have a certain value using this flag. The match counter is a 6 bit counter which can be used to specify how many times a level should match before proceeding to the next. This is all run-time configurable through registers described in the register section.

To specify post-, center- or pre-triggering mode, the user can set a counter register that controls when the sampling stops relative to the triggering event. It can be set to any value in the range 0 to *depth*-1 thus giving total control of the trace buffer content.

To support the tracing of slowly changing signals, the logic analyzer has a 16-bit sample frequency divider register that controls how often the signals are sampled. The default divider value of 1 will sample the signals every clock cycle.

The *usequal* configuration option has a similar purpose as the sample frequency divider. The user can define one of the traced signals as a qualifier bit that has to have a specified value for the current signals to be stored in the trace buffer. This makes sampling of larger time periods possible if only some easily distinguished samples are interesting. This option has to be enabled with the *usequal* generic and the qualifier bit and value are written to a register.

105.2.4 Arming

To start operation, the logic analyzer needs to be armed. This is done by writing to the status register with bit 0 set to 1. A reset can be performed anytime by writing zero to the status register. After the final triggering event, the trigged flag will be raised and can be read out from the status register. The logic analyzer remains armed and trigged until the trigger counter reaches zero. When this happens the index of the oldest sample can be read from the trace buffer index register.

105.3 Registers

Both trace data and all registers are accessed through an APB interface. The LOGAN core will allocate a 64 kbyte block in the APB address space.

Table 2079. APB address mapping

APB address offset	Registers
0x0000	Status register
0x0004	Trace buffer index
0x0008	Page register
0x000C	Trig counter
0x0010	Sample freq. divider
0x0014	Storage qualifier setting
0x2000 - 0x20FF	Trig control settings
0x6000 - 0x6FFF	Pattern/mask configuration
0x8000 - 0xFFFF	Trace data

105.3.1 Status register

Table 2080. 0x0000 - STAT - Status register

31	30	29	28	27 20	19 6	5 0
usereg	qualifier	armed	trigged	dbits	depth	trig levels
*	*	0	0	*	*	*
r	r	rw	rw	r	r	r

31: 28 These bits indicate whether an input register and/or storage qualifier is used and if the Logic Analyzer is armed and/or trigged.

27: 20 Number of traced signals (Read Only).

19: 6 Last index of trace buffer. Depth-1 (Read Only).

5: 0 Number of trig levels (Read Only).

105.3.2 Trace buffer index

Table 2081. 0x0004 - INDEX - Trace buffer index register

31 abits	abits-1 0
RESERVED	the index of the oldest sample
0	0
r	r

31: *abits* Reserved.

abits-1:0 The index of the oldest sample in the buffer. The trace buffer index simply points to where a new sample will be written , in a circular buffer it is the oldest sample if it was previously written. *abits* is the number of bits needed to represent the configured depth.

Note that this register is written by the trigger engine clock domain and thus needs to be known stable when read out. Only when the ‘armed’ bit in the status register is zero is the content of this register reliable.

105.3.3 Page register

Table 2082. 0x0008 - PAGE - Trace buffer index register

31 4	3 0
RESERVED	current page
0	0
r	rw

31: 4 Reserved.

3: 0 This register selects what page that will be used when reading from the trace buffer.

The trace buffer is organized into pages of 1024 samples. Each sample can be between 1 and 256 bits. If the depth of the buffer is more than 1024 the page register has to be used to access the other pages. To access the i:th page the register should be set i (where i=0..15).

105.3.4 Trig counter

Table 2083. 0x000C - TRIGC - Trig counter register

31 abits	abits-1 0
RESERVED	trig counter value
0	0
r	rw

31:abits Reserved.

nbits-1:0 Trig counter value. A counter is incremented by one for each stored sample after the final triggering event and when it reaches the value stored in this register the sampling stops. 0 means posttrig and *depth-1* is pretrig. Any value in between can be used.

105.3.5 Sample frequency divider

*Table 2084.*0x0010 - CLKDIV - Sample freq. divider register

31 16	15 0
RESERVED	divider value
0	0x0001
r	rw

31: 16 Reserved.

15: 0 A sample is stored on every i:th clock cycle where i is specified through this register. This resets to 1 thus sampling occurs every cycle if not changed.

105.3.6 Storage qualifier

*Table 2085.*0x0014 - SQUAL - Storage qualifier register

31 9	8	7 0
RESERVED	val	qualifier
0	0	0
r	rw	rw

31: 9 Reserved.

8: Qualify storage if bit is 1/0.

7: 0 Which bit to use as qualifier.

105.3.7 Trig control registers

This memory area contains the registers that control when the trigger engine shall proceed to the next level, i.e the match counter and a one bit field that specifies if it should trig on equality or inequality. There are *trigl* words where each word is used like in the figure below.

Table 2086. 0x2000-0x20FF - TCTRL - Trigger control register

31 7	6 1	0
RESERVED	match counter	eq
0	NR	nr
r	rw	rw

31: 7 Reserved.

6: 1 Match counter. A counter is increased with one on each match on the current level and when it reaches the value stored in this register the trigger engine proceeds to the next level or if it is the last level it raises the trigged flag and starts the count of the trigger counter.

0: Specifies if a match is that the pattern/mask combination is equal or inequal compared to the traced signals. 1 - equal

0- inequal.

105.3.8 Pattern/mask configuration

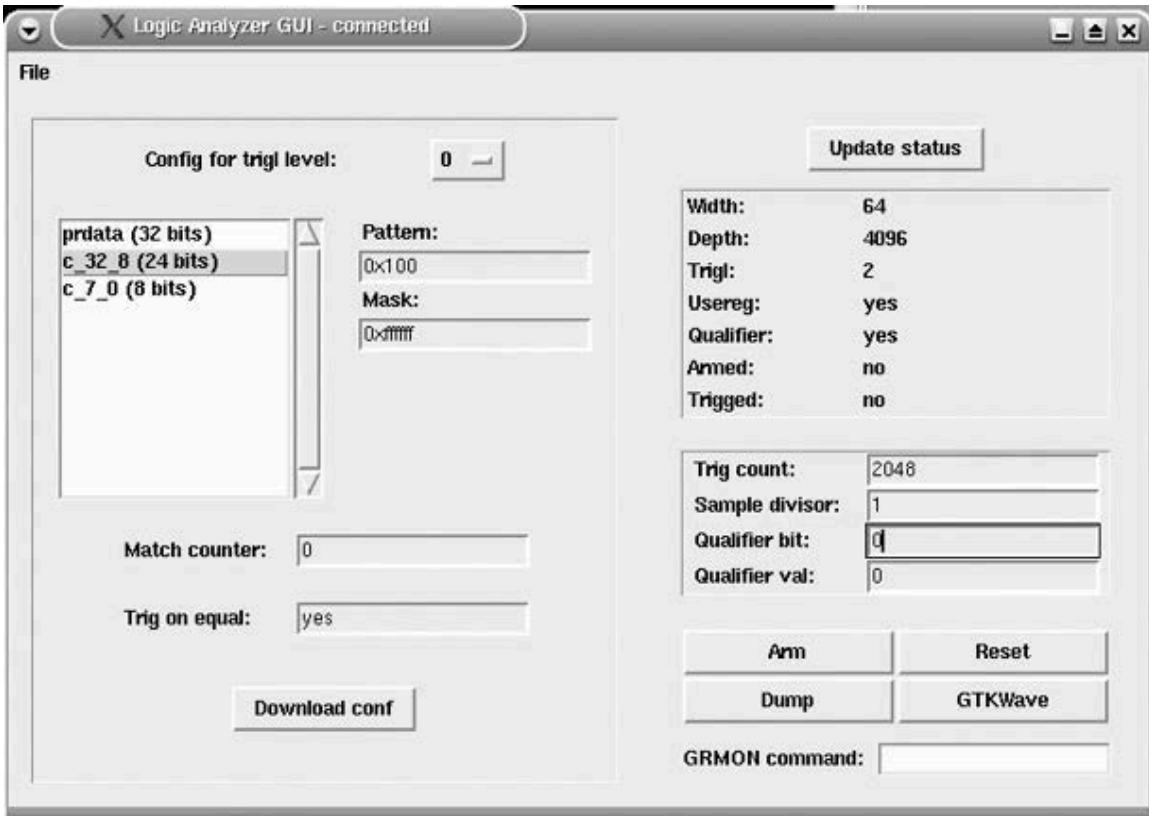
In these registers the pattern and mask for each trig level is configured. The pattern and mask can contain up to 8 words (256 bits) each so a number of writes can be necessary to specify just one pattern. They are stored with the LSB at the lowest address. The pattern of the first trig level is at 0x6000 and the mask is located 8 words later at 0x6020. Then the next trig levels starts at address 0x6040 and so on.

105.3.9 Trace data

It is placed in the upper half of the allocated APB address range. If the configuration needs more than the allocated 32 kB of the APB range the page register is used to page into the trace buffer. Each stored word is *dbits* wide but 8 words of the memory range is always allocated so the entries in the trace buffer are found at multiples of 0x20, i.e. 0x8000, 0x8020 and so on.

105.4 Graphical interface

The logic analyzer is normally controlled by the LOGAN debug driver in GRMON. It is also possible to control the LOGAN operation using a graphical user interface (GUI) written in Tcl/Tk. The GUI is provided with GRMON, refer to the GRMON manual for more details.



105.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x062. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

105.6 Implementation

105.6.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

105.7 Configuration options

Table 2087 shows the configuration options of the core (VHDL generics).

*Table 2087.*Configuration options

Generic	Function	Allowed range	Default
dbits	Number of traced signals	1 - 255	32
depth	Number of stored samples	256 - 16384	1024
trigl	Number of trigger levels	1 - 63	1
usereg	Use input register	0 - 1	1
usequal	Use storage qualifier	0 - 1	0
pindex	APB slave index	0 - NAPBSLV - 1	0
paddr	The 12-bit MSB APB address	0 -16#FFF#	0
pmask	The APB address mask	16#000 - 16#F00#	F00
memtech	Memory technology	0 - NTECH	0

The usereg VHDL generic specifies whether to use an input register to synchronize the traced signals and to minimize their fan out. If usereg=1 then all signals will be clocked into a register on the positive edge of the supplied clock signal, otherwise they are sent directly to the RAM.

105.8 Signal descriptions

Table 2088 shows the interface signals of the core (VHDL ports).

*Table 2088.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	System clock	-
TCLK	N/A	Input	Sample clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
SIGNALS	N/A	Input	Vector of traced signals	-

* See GRLIB IP Library users manual

105.9 Library dependencies

Table 2089 shows libraries used when instantiating the core (VHDL libraries).

*Table 2089.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

105.10 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

entity logan_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    ... -- other signals
    );
end;

architecture rtl of logan_ex is

  -- AMBA signals
signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector := (others => apb_none);
signal signals : std_logic_vector(63 downto 0);

begin

-- Logic analyzer core
  logan0 : logan
  generic map (dbits=>64,depth=>4096,trigl=>2,usereg=>1,usequal=>0,
            pindex => 3, paddr => 3, pmask => 16#F00#, memtech => memtech)
  port map (rstn, clk, clk, apbi, apbo(3), signals);

end;
```

106 MCTRL - Combined PROM/IO/SRAM/SDRAM Memory Controller

106.1 Overview

The memory controller handles a memory bus hosting PROM, memory mapped I/O devices, asynchronous static ram (SRAM) and synchronous dynamic ram (SDRAM). The controller acts as a slave on the AHB bus. The function of the memory controller is programmed through memory configuration registers 1, 2 & 3 (MCFG1, MCFG2 & MCFG3) through the APB bus. The memory bus supports four types of devices: PROM, SRAM, SDRAM and local I/O. The memory bus can also be configured in 8- or 16-bit mode for applications with low memory and performance demands.

Chip-select decoding is done for two PROM banks, one I/O bank, five SRAM banks and two SDRAM banks.

The controller decodes three address spaces (PROM, I/O and RAM) whose mapping is determined through VHDL-generics.

Figure 280 shows how the connection to the different device types is made.

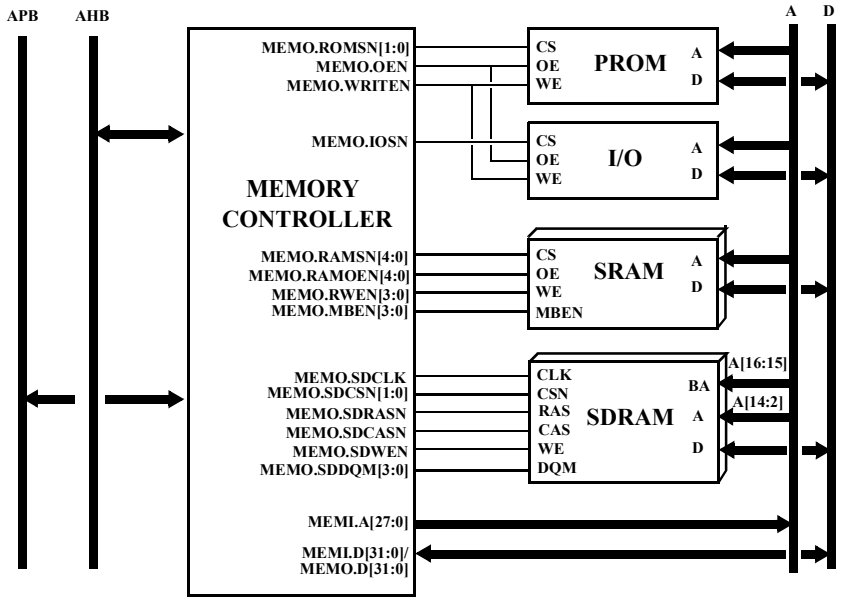


Figure 280. **Memory controller conected to AMBA bus and different types of 32-bit memory devices**

106.2 PROM access

Accesses to prom have the same timing as RAM accesses, the differences being that PROM cycles can have up to 15 waitstates.

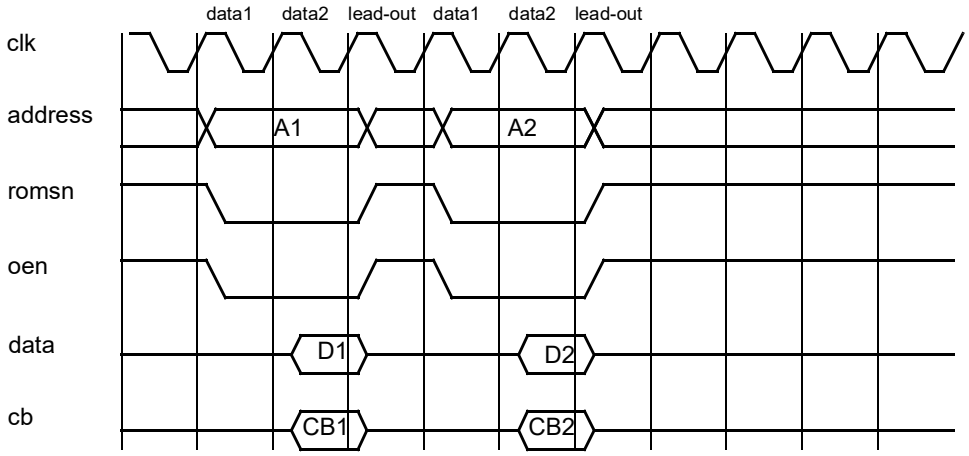


Figure 281. **Prom non-consecutive read cyclecs.**

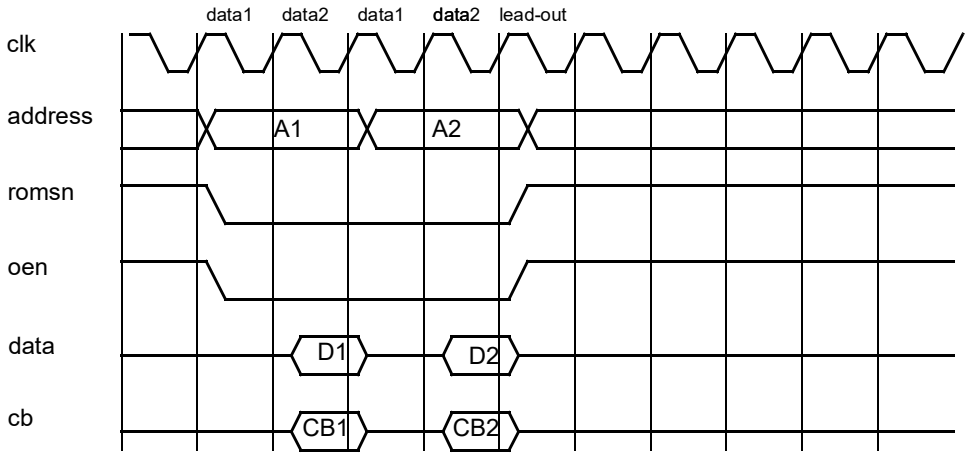


Figure 282. **Prom consecutive read cyclecs.**

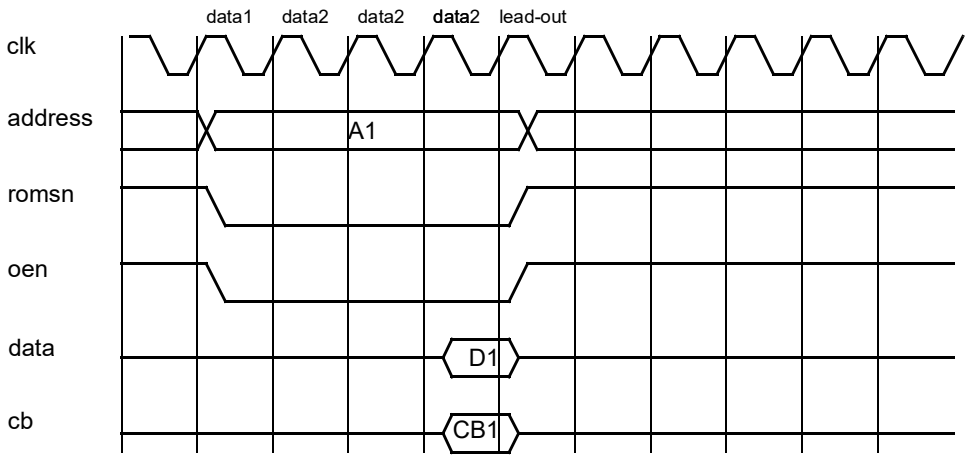


Figure 283. **Prom read access with two waitstates.**

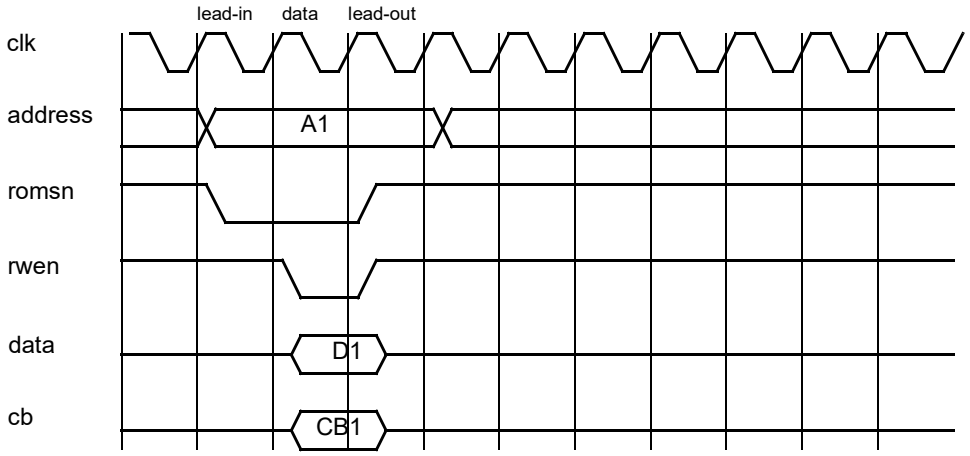


Figure 284. **Prom write cycle (0-waitstates)**

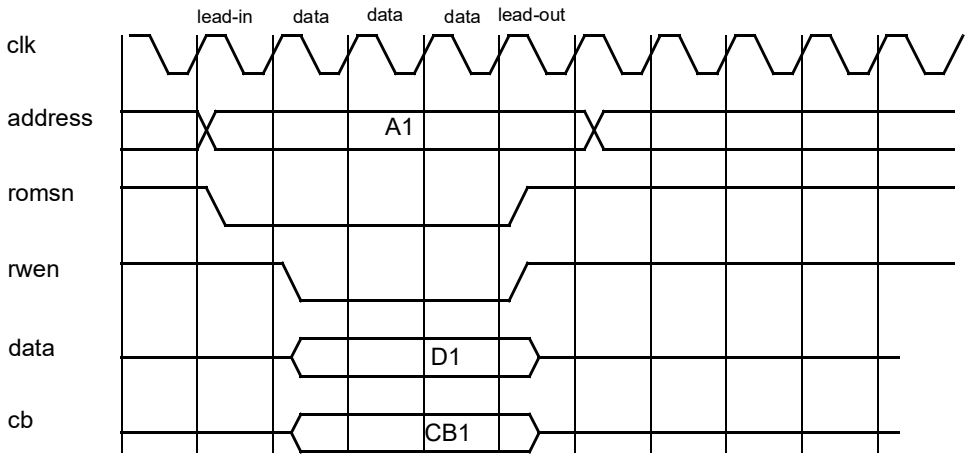


Figure 285. **Prom write cycle (2-waitstates)**

Two PROM chip-select signals are provided, MEMO.ROMSN[1:0]. MEMO.ROMSN[0] is asserted when the lower half of the PROM area as addressed while MEMO.ROMSN[1] is asserted for the upper half.

106.3 Memory mapped I/O

Accesses to I/O have similar timing to ROM/RAM accesses, the differences being that a additional waitstates can be inserted by de-asserting the MEMI.BRDYN signal. The I/O select signal (MEMO.IOSN) is delayed one clock to provide stable address before MEMO.IOSN is asserted.

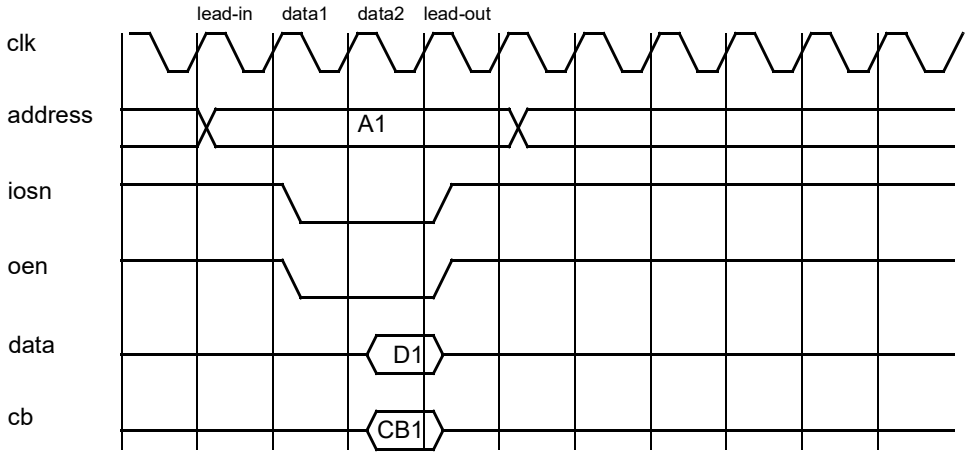


Figure 286. **I/O read cycle (0-waitstates)**

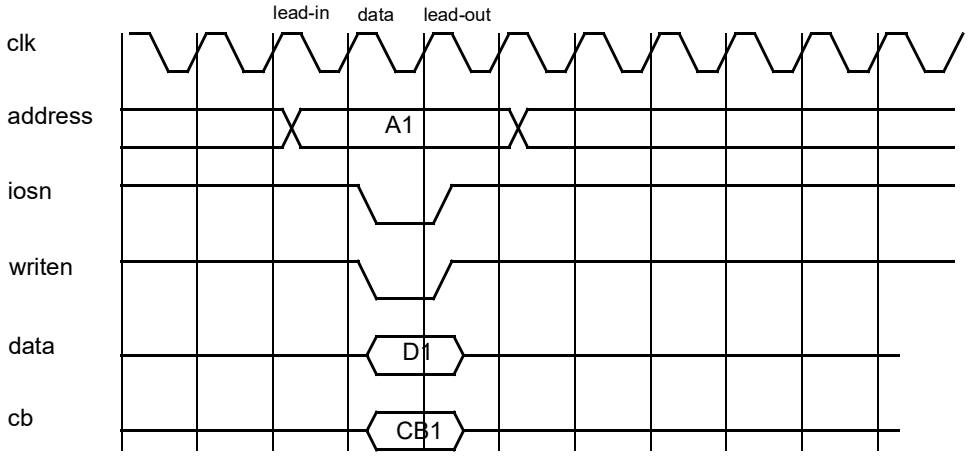


Figure 287. **I/O write cycle (0-waitstates)**

106.4 SRAM access

The SRAM area can be up to 1 Gbyte, divided on up to five RAM banks. The size of banks 1-4 (MEMO.RAMSN[3:0]is programmed in the RAM bank-size field (MCFG2[12:9]) and can be set in binary steps from 8 Kbyte to 256 Mbyte. The fifth bank (RAMSN[4]) decodes the upper 512 Mbyte (controlled by means of the *sdrasel* VHDL generic) and cannot be used simultaneously with SDRAM memory. A read access to SRAM consists of two data cycles and between zero and three waitstates. Accesses to MEMO.RAMSN[4] can further be stretched by de-asserting MEMI.BRDYN until the data is available. On non-consecutive accesses, a lead-out cycle is added after a read cycle to prevent bus contention due to slow turn-off time of memories or I/O devices. Figure 288 shows the basic read cycle waveform (zero waitstate).

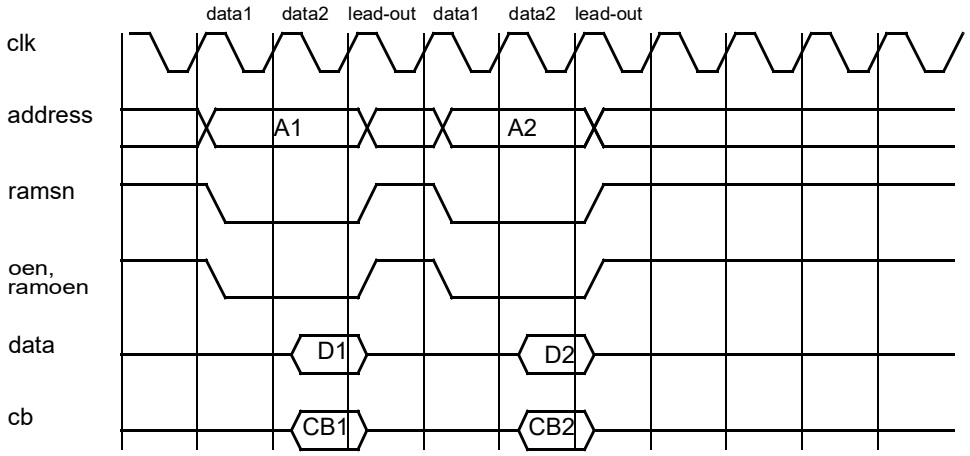


Figure 288. **SRAM non-consecutive read cyclecs.**

For read accesses to MEMO.RAMSN[4:0], a separate output enable signal (MEMO.RAMOEN[n]) is provided for each RAM bank and only asserted when that bank is selected. A write access is similar to the read access but takes a minimum of three cycles:

Through an (optional) feed-back loop from the write strobes, the data bus is guaranteed to be driven until the write strobes are de-asserted. Each byte lane has an individual write strobe to allow efficient byte and half-word writes. If the memory uses a common write strobe for the full 16- or 32-bit data, the read-modify-write bit in the MCFG2 register should be set to enable read-modify-write cycles for sub-word writes.

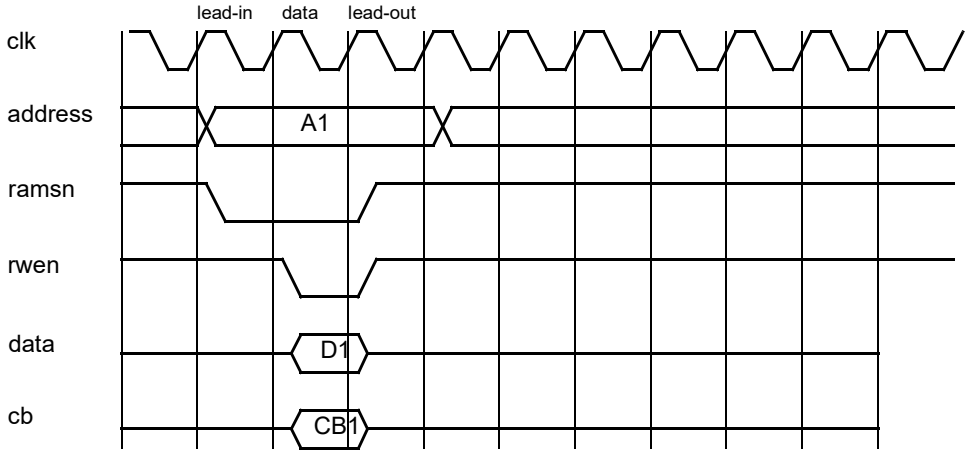


Figure 289. **Sram write cycle (0-waitstates)**

A drive signal vector for the data I/O-pads is provided which has one drive signal for each data bit. It can be used if the synthesis tool does not generate separate registers automatically for the current technology. This can remove timing problems with output delay.

106.5 8-bit and 16-bit PROM and SRAM access

To support applications with low memory and performance requirements efficiently, it is not necessary to always have full 32-bit memory banks. The SRAM and PROM areas can be individually configured for 8- or 16-bit operation by programming the ROM and RAM size fields in the memory configuration registers. Since read access to memory is always done on 32-bit word basis, read access to 8-bit memory will be transformed in a burst of four read cycles while access to 16-bit memory will

generate a burst of two 16-bits reads. During writes, only the necessary bytes will be writen. Figure 290 shows an interface example with 8-bit PROM and 8-bit SRAM. Figure 291 shows an example of a 16-bit memory interface.

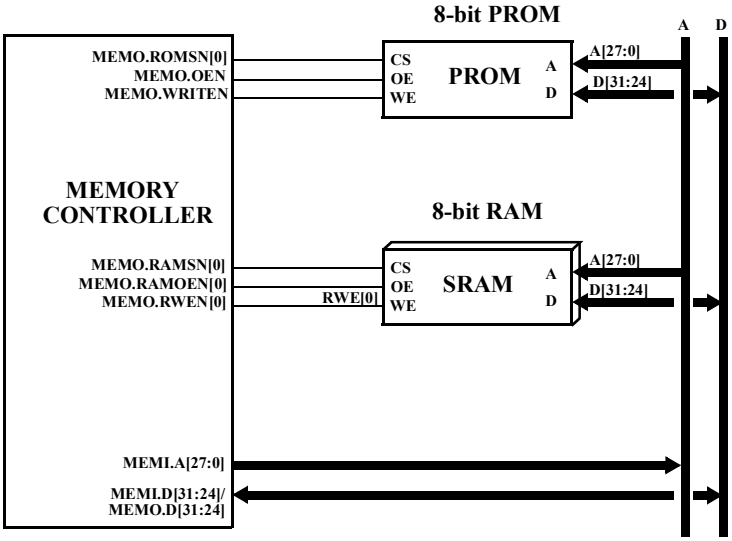


Figure 290. **8-bit memory interface example**

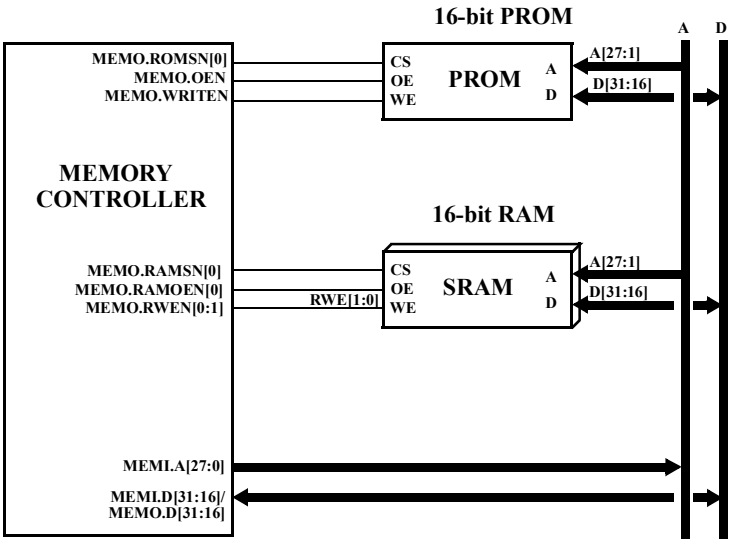


Figure 291. **16-bit memory interface example**

106.6 Burst cycles

To improve the bandwidth of the memory bus, accesses to consecutive addresses can be performed in burst mode. Burst transfers will be generated when the memory controller is accessed using an AHB burst request. These includes instruction cache-line fills, double loads and double stores. The timing of a burst cycle is identical to the programmed basic cycle with the exception that during read cycles, the lead-out cycle will only occurs after the last transfer. Burst cycles will not be generated to the IO area.

Only word (HSIZE = "010") bursts of incremental type (HBURST=INCR, INCR4, INCR8 or INCR16) are supported.

106.7 8- and 16-bit I/O access

Similar to the PROM/RAM areas, the I/O area can also be configured to 8- or 16-bit mode. However, the I/O device will NOT be accessed by multiple 8/16 bit accesses as the memory areas, but only with one single access just as in 32-bit mode. To access an I/O device on a 16-bit bus, LDUH/STH instructions should be used while LDUB/STB should be used with an 8-bit bus.

106.8 SDRAM access

106.8.1 General

Synchronous dynamic RAM (SDRAM) access is supported to two banks of PC100/PC133 compatible devices. This is implemented by a special version of the SDCTRL SDRAM controller core from Frontgrade Gaisler, which is optionally instantiated as a sub-block. The SDRAM controller supports 64M, 256M and 512M devices with 8 - 12 column-address bits, and up to 13 row-address bits. The size of the two banks can be programmed in binary steps between 4 Mbyte and 512 Mbyte. The operation of the SDRAM controller is controlled through MCFG2 and MCFG3 (see below). Both 32- and 64-bit data bus width is supported, allowing the interface of 64-bit DIMM modules. The memory controller can be configured to use either a shared or separate bus connecting the controller and SDRAM devices. When the VHDL generic **mobile** is set to a value not equal to 0, the controller supports mobile SDRAM.

106.8.2 Address mapping

The two SDRAM chip-select signals are decoded. SDRAM area is mapped into the upper half of the RAM area defined by BAR2 register. When the SDRAM enable bit is set in MCFG2, the controller is enabled and mapped into upper half of the RAM area as long as the SRAM disable bit is not set. If the SRAM disable bit is set, all access to SRAM is disabled and the SDRAM banks are mapped into the lower half of the RAM area.

106.8.3 Initialisation

When the SDRAM controller is enabled, it automatically performs the SDRAM initialisation sequence of PRECHARGE, 2x AUTO-REFRESH and LOAD-MODE-REG on both banks simultaneously. When mobile SDRAM functionality is enabled the initialization sequence is appended by a LOAD-EXTMODE-REG command. The controller programs the SDRAM to use page burst on read and single location access on write.

106.8.4 Configurable SDRAM timing parameters

To provide optimum access cycles for different SDRAM devices (and at different frequencies), some SDRAM parameters can be programmed through memory configuration register 2 (MCFG2) The programmable SDRAM parameters can be seen in tabel 2090.

***Table 2090.*SDRAM programmable timing parameters**

Function	Parameter	Range	Unit
CAS latency, RAS/CAS delay	t_{CAS}, t_{RCD}	2 - 3	clocks
Precharge to activate	t_{RP}	2 - 3	clocks
Auto-refresh command period	t_{RFC}	3 - 11	clocks
Auto-refresh interval		10 - 32768	clocks

Remaining SDRAM timing parameters are according the PC100/PC133 specification.

When mobile SDRAM support is enabled, one additional timing parameter (TXSR) can be programmed though the Power-Saving configuration register.

*Table 2091.*Mobile SDRAM programmable minimum timing parameters

SDRAM timing parameter	Minimum timing (clocks)
Exit Self Refresh mode to first valid command (t_{XSR})	tXSR

106.9 Refresh

The SDRAM controller contains a refresh function that periodically issues an AUTO-REFRESH command to both SDRAM banks. The period between the commands (in clock periods) is programmed in the refresh counter reload field in the MCFG3 register. Depending on SDRAM type, the required period is typically 7.8 or 15.6 µs (corresponding to 780 or 1560 clocks at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by setting bit 31 in MCFG2.

106.9.1 Self Refresh

The self refresh mode can be used to retain data in the SDRAM even when the rest of the system is powered down. When in the self refresh mode, the SDRAM retains data without external clocking and refresh are handled internally. The memory array that is refreshed during the self refresh operation is defined in the extended mode register. These settings can be changed by setting the PASR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the PASR bits are changed. The supported “Partial Array Self Refresh” modes are: Full, Half, Quarter, Eighth, and Sixteenth array. “Partial Array Self Refresh” is only supported when mobile SDRAM functionality is enabled. To enable the self refresh mode, set the PMODE bits in the Power-Saving configuration register to “010” (Self Refresh). The controller will enter self refresh mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits are cleared. When exiting this mode the controller introduce a delay defined by tXSR in the Power-Saving configuration register and a AUTO REFRESH command before any other memory access is allowed. The minimum duration of this mode is defined by tRAS. This mode is only available then the VHDL generic **mobile** >=1.

106.9.2 Power-Down

When entering the power-down mode all input and output buffers, excluding SDCKE, are deactivated. All data in the SDRAM is retained during this operation. To enable the power-down mode, set the PMODE bits in the Power-Saving configuration register to “001” (Power-Down). The controller will enter power-down mode after every memory access (when the controller has been idle for 16

clock cycles), until the PMODE bits is cleared. The REFRESH command will still be issued by the controller in this mode. When exiting this mode a delay of one clock cycles are added before issue any command to the memory. This mode is only available then the VHDL generic **mobile** >=1.

106.9.3 Deep Power-Down

The deep power-down operating mode is used to achieve maximum power reduction by eliminating the power of the memory array. Data will not be retained after the device enters deep power-down mode. To enable the deep power-down mode, set the PMODE bits in the Power-Saving configuration register to “101” (Deep Power-Down). To exit the deep power-down mode the PMODE bits in the Power-Saving configuration register must be cleared. The controller will respond with an AMBA ERROR response to an AMBA access, that will result in a memory access, during Deep Power-Down mode. This mode is only available then the VHDL generic **mobile** >=1 and mobile SDRAM functionality is enabled.

106.9.4 Temperature-Compensated Self Refresh

The settings for the temperature-compensation of the Self Refresh rate can be controlled by setting the TCSR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the TCSR bits are changed. Note that some vendors implements a Internal Temperature-Compensated Self Refresh feature, which makes the memory to ignore the TCSR bits. This functionality is only available then the VHDL generic **mobile** >=1 and mobile SDRAM functionality is enabled.

106.9.5 Drive Strength

The drive strength of the output buffers can be controlled by setting the DS bits in the Power-Saving configuration register. The extended mode register is automatically updated when the DS bits are changed. The available options are: full, three-quarter, one-half, and one-quarter drive strengths. This functionality is only available then the VHDL generic **mobile** >=1 and mobile SDRAM functionality is enabled.

106.9.6 SDRAM commands

The controller can issue four SDRAM commands by writing to the SDRAM command field in MCFG2: PRE-CHARGE, AUTO-REFRESH, LOAD-MODE-REG (LMR) and LOAD-EXTMODE-REG (EMR). If the LMR command is issued, the CAS delay as programmed in MCFG2 will be used, remaining fields are fixed: page read burst, single location write, sequential burst. If the EMR command is issued, the DS, TCSR and PASR as programmed in Power-Saving configuration register will be used. To issue the EMR command, the EMR bit in the MCFG4 register has to be set. The command field will be cleared after a command has been executed. Note that when changing the value of the CAS delay, a LOAD-MODE-REGISTER command should be generated at the same time.

106.9.7 Read cycles

A read cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a READ command after the programmed CAS delay. A read burst is performed if a burst access has been requested on the AHB bus. The read cycle is terminated with a PRE-CHARGE command, no banks are left open between two accesses.

106.9.8 Write cycles

Write cycles are performed similarly to read cycles, with the difference that WRITE commands are issued after activation. A write burst on the AHB bus will generate a burst of write commands without idle cycles in-between.

106.9.9 Address bus connection

The memory controller can be configured to either share the address and data buses with the SRAM, or to use separate address and data buses. When the buses are shared, the address bus of the SDRAMs should be connected to A[14:2], the bank address to A[16:15]. The MSB part of A[14:2] can be left unconnected if not used. When separate buses are used, the SDRAM address bus should be connected to SA[12:0] and the bank address to SA[14:13].

106.9.10Data bus

SDRAM can be connected to the memory controller through the common or separate data bus. If the separate bus is used the width is configurable to 32 or 64 bits. 64-bit data bus allows the 64-bit SDRAM devices to be connected using the full data capacity of the devices. 64-bit SDRAM devices can be connected to 32-bit data bus if 64-bit data bus is not available but in this case only half the full data capacity will be used. There is a drive signal vector and separate data vector available for SDRAM. The drive vector has one drive signal for each data bit. These signals can be used to remove timing problems with the output delay when a separate SDRAM bus is used. SDRAM bus signals are described in section 106.14, for configuration options refer to section 106.16.2.

106.9.11Clocking

The SDRAM clock typically requires special synchronisation at layout level. For Xilinx and Altera device, the GR Clock Generator can be configured to produce a properly synchronised SDRAM clock. For other FPGA targets, the GR Clock Generator can produce an inverted clock.

106.10 Using bus ready signaling

The MEMI.BRDYN signal can be used to stretch access cycles to the I/O area and the ram area decoded by MEMO.RAMSN[4]. The accesses will always have at least the pre-programmed number of waitstates as defined in memory configuration registers 1 & 2, but will be further stretched until MEMI.BRDYN is asserted. MEMI.BRDYN should be asserted in the cycle preceding the last one. The use of MEMI.BRDYN can be enabled separately for the I/O and RAM areas.

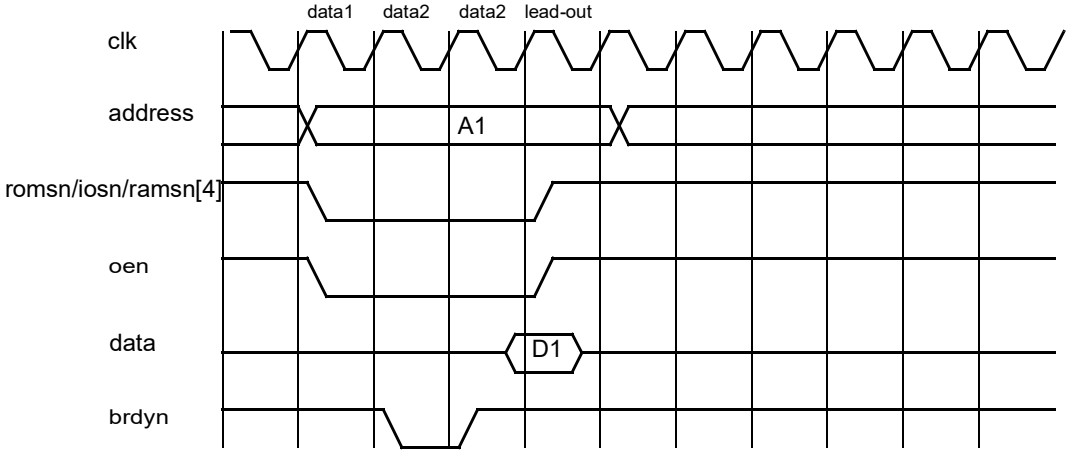


Figure 292. **READ cycle with one extra data2 cycle added with BRDYN (synchronous sampling). Lead-out cycle is only applicable for I/O accesses.**

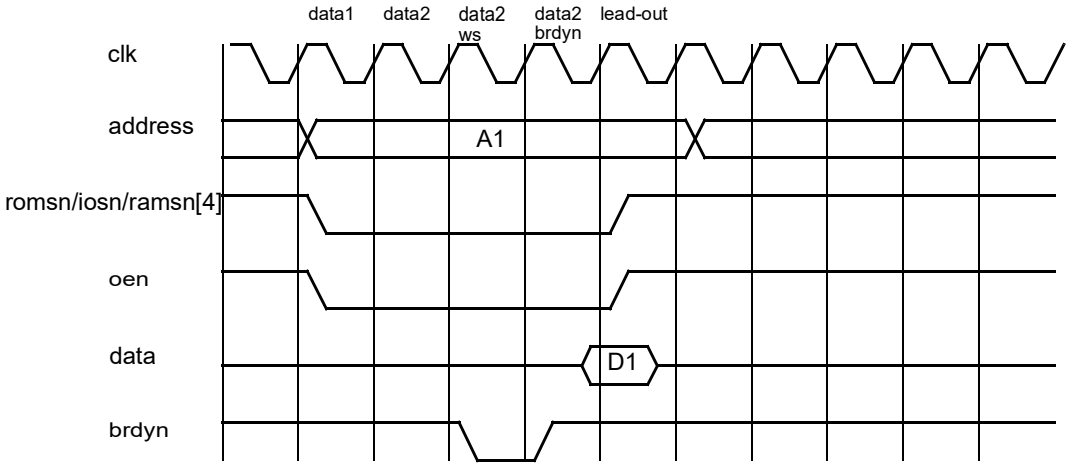


Figure 293. **Read cycle with one waitstate (configured) and one BRDYN generated waitstate (synchronous sampling).**

106.11 Access errors

An access error can be signaled by asserting the MEMI.BEXCN signal, which is sampled together with the data. If the usage of MEMI.BEXCN is enabled in memory configuration register 1, an error response will be generated on the internal AMBA bus. MEMI.BEXCN can be enabled or disabled through memory configuration register 1, and is active for all areas (PROM, I/O an RAM).

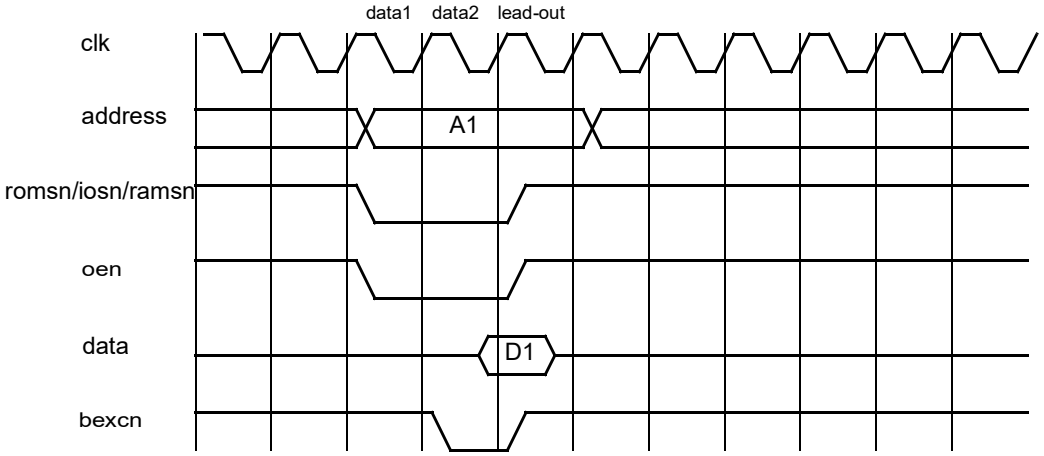


Figure 294. **Read cycle with BEXCN.**

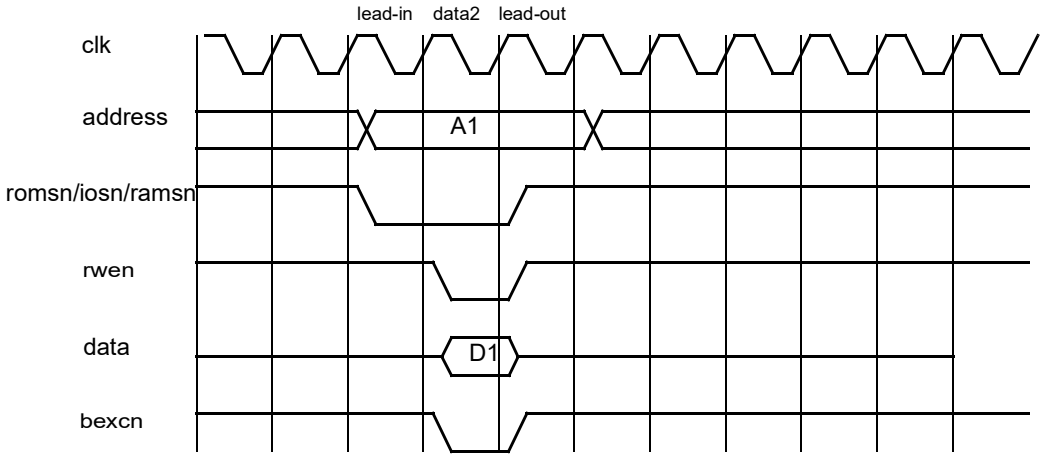


Figure 295. **Write cycle with BEXCN. Chip-select (iosn) is not asserted in lead-in cycle for io-accesses.**

106.12 Attaching an external DRAM controller

To attach an external DRAM controller, MEMO.RAMSN[4] should be used since it allows the cycle time to vary through the use of MEMI.BRDYN. In this way, delays can be inserted as required for opening of banks and refresh.

106.13 Endianness

The core is designed for big-endian systems.

106.14 Registers

The memory controller is programmed through registers mapped into APB address space.

Table 2092. **Memory controller registers**

APB address offset	Register
0x0	MCFG1
0x4	MCFG2
0x8	MCFG3
0xC	MCFG4 (Power-Saving configuration register)

106.14.1 Memory configuration register 1 (MCFG1)

Memory configuration register 1 is used to program the timing of rom and local I/O accesses.

Table 2093. **0x00 - MCFG1 - Memory configuration register 1.**

31 29	28 27	26	25	24	23 20	19	18
RESERVED	IOBUSW	IBRDY	BEXCN	R	IO WAITSTATES	IOEN	RESERVED
0	NR	0	0	0	0	0	0
r	rw	rw	rw	r	rw	rw	r

12	11	10	9 8	7 4	3 0
RESERVED	PWEN	R	PROM WIDTH	PROM WRITE WS	PROM READ WS
0	0	0	*	0xF	0xF
r	rw	r	rw	rw	rw

31 : 29 RESERVED

28 : 27 I/O bus width (IOBUSW) - Sets the data width of the I/O area (“00”=8, “01”=16, “10” =32).

Table 2093. **0x00 - MCFG1 - Memory configuration register 1.**

26	I/O bus ready enable (IBRDY) - Enables bus ready (BRDYN) signalling for the I/O area. Reset to '0'.
25	Bus error enable (BEXCN) - Enables bus error signalling. Reset to '0'.
24	RESERVED
23 : 20	I/O waitstates (IO WAITSTATES) - Sets the number of waitstates during I/O accesses ("0000"=0, "0001"=1, "0010"=2,..., "1111"=15).
19	I/O enable (IOEN) - Enables accesses to the memory bus I/O area.
18:12	RESERVED
11	PROM write enable (PWEN) - Enables write cycles to the PROM area.
10	RESERVED
9 : 8	PROM width (PROM WIDTH) - Sets the data width of the PROM area ("00"=8, "01"=16, "10"=32).
7 : 4	PROM write waitstates (PROM WRITE WS) - Sets the number of wait states for PROM write cycles ("0000"=0, "0001"=1, "0010"=2,..., "1111"=15).
3 : 0	PROM read waitstates (PROM READ WS) - Sets the number of wait states for PROM read cycles ("0000"=0, "0001"=1, "0010"=2,...,"1111"=15). Reset to "1111".

During power-up, the prom width (bits [9:8]) are set with value on MEMI.BWIDTH inputs. The prom waitstates fields are set to 15 (maximum). External bus error and bus ready are disabled. All other fields are undefined.

106.14.2 Memory configuration register 2 (MCFG2)

Memory configuration register 2 is used to control the timing of the SRAM and SDRAM.

Table 2094. **0x04 - MCFG2 - Memory configuration register 2.**

31	30	29 27	26	25 23	22 21	20 19	18	17	16
SDRF	TRP	SDRAM TRFC	TCAS	SDRAM BANKSZ	SDRAM COLSZ	SDRAM CMD	D64	R	MS
0	1	0b111	1	0	0b10	0	*	0	*
rw	rw	rw	rw	rw	rw	rw	r	r	r

15	14	13	12 9	8	7	6	5 4	3 2	1 0
RES	SE	SI	RAM BANK SIZE	R	RBRDY	RMW	RAM WIDTH	RAM WRITE WS	RAM READ WS
0	0	0	NR	0	NR	NR	NR	0	0
r	rw	rw	rw	r	rw	rw	rw	rw	rw

31	SDRAM refresh (SDRF) - Enables SDRAM refresh.
30	SDRAM TRP parameter (TRP) - t_{RP} will be equal to 2 or 3 system clocks (0/1).
29 : 27	SDRAM TRFC parameter (SDRAM TRFC) - t_{RFC} will be equal to 3+field-value system clocks.
26	SDRAM TCAS parameter (TCAS) - Selects 2 or 3 cycle CAS delay (0/1). When changed, a LOAD-COMMAND-REGISTER command must be issued at the same time. Also sets RAS/CAS delay (t_{RCD}).
25 : 23	SDRAM bank size (SDRAM BANKSZ) - Sets the bank size for SDRAM chip selects ("000"=4 Mbyte, "001"=8 Mbyte, "010"=16 Mbyte.... "111"=512 Mbyte). When configured for 64-bit wide SDRAM data bus (sdbits=64), the meaning of this field doubles so that "000"=8 Mbyte, .., "111"=1024 Mbyte
22 : 21	SDRAM column size (SDRAM COLSZ) - "00"=256, "01"=512, "10"=1024, "11"=2048 except when bit[25:23]="111" then "11"=4096
20 : 19	SDRAM command (SDRAM CMD) - Writing a non-zero value will generate a SDRAM command. "01"=PRECHARGE, "10"=AUTO-REFRESH, "11"=LOAD-COMMAND-REGISTER. The field is reset after the command has been executed.
18	64-bit SDRAM data bus (D64) - Reads '1' if the memory controller is configured for 64-bit SDRAM data bus width, '0' otherwise. Read-only.
17	RESERVED
16	Mobile SDR support enabled. '1' = Enabled, '0' = Disabled (read-only)

Table 2094. **0x04 - MCFG2 - Memory configuration register 2.**

15	RESERVED
14	SDRAM enable (SE) - Enables the SDRAM controller.
13	SRAM disable (SI) - Disables accesses RAM if bit 14 (SE) is set to '1'.
12 : 9	RAM bank size (RAM BANK SIZE) - Sets the size of each RAM bank ("0000"=8 kbyte, "0001"=16 kbyte, ..., "1111"=256 Mbyte).
8	RESERVED
7	RAM bus ready enable (RBRDY) - Enables bus ready signalling for the RAM area.
6	Read-modify-write enable (RMW) - Enables read-modify-write cycles for sub-word writes to 16- bit 32-bit areas with common write strobe (no byte write strobe).
5 : 4	RAM width (RAM WIDTH) - Sets the data width of the RAM area ("00"=8, "01"=16, "1X"=32).
3 : 2	RAM write waitstates (RAM WRITE WS) - Sets the number of wait states for RAM write cycles ("00"=0, "01"=1, "10"=2, "11"=3).
1 : 0	RAM read waitstates (RAM READ WS) - Sets the number of wait states for RAM read cycles ("00"=0, "01"=1, "10"=2, "11"=3).

106.14.3Memory configuration register 3 (MCFG3)

MCFG3 is contains the reload value for the SDRAM refresh counter.

TABLE 2095. 0x08 - MCFG3 - Memory Configuration register 3

31 27	26 12	11 0
RESERVED	SDRAM REFRESH RELOAD VALUE	RESERVED

31: 27	RESERVED
26: 12	SDRAM refresh counter reload value (SDRAM REFRESH RELOAD VALUE)
11: 0	RESERVED

The period between each AUTO-REFRESH command is calculated as follows:

$t_{REFRESH}$ = ((reload value) + 1) / SYSCLK

106.14.4Power-Saving Configuration Register

*Table 2096.*0x0C - MCFG4 - Power-Saving configuration register

31	30	29	28 24	23 20	19	18 16	15 7	6	5	4	3	2 0
ME	CE	EM	RESERVED	tXSR	R	PMODE	RESERVED		DS		TCSR	PASR
*	*	0	0	0b111	0	0	0		0		0	0
rw	rw	rw	r	rw	r	rw	r		rw		rw	rw

31	Mobile SDRAM functionality enabled. ‘1’ = Enabled (support for Mobile SDRAM), ‘0’ = disabled (support for standard SDRAM)
30	Clock enable (CE). This value is driven on the CKE inputs of the SDRAM. Should be set to ‘1’ for correct operation. This register bit is read only when Power-Saving mode is other then none.
29	EMR. When set, the LOAD-COMMAND-REGISTER command issued by the SDRAM command field in MCFG2 will be interpret as a LOAD-EXTENDED-COMMAND-REGISTER command.
28: 24	Reserved
23: 20	SDRAM tXSR timing. tXSR will be equal to field-value system clocks. (Read only when Mobile SDR support is disabled).
19	Reserved
18: 16	Power-Saving mode (Read only when Mobile SDR support is disabled). “000”: none “001”: Power-Down (PD) “010”: Self-Refresh (SR) “101”: Deep Power-Down (DPD)
15: 7	Reserved
6: 5	Selectable output drive strength (Read only when Mobile SDR support is disabled). “00”: Full “01”: One-half “10”: One-quarter “11”: Three-quarter
4: 3	Reserved for Temperature-Compensated Self Refresh (Read only when Mobile SDR support is disabled). “00”: 70ªC “01”: 45ªC “10”: 15ªC “11”: 85ªC
2: 0	Partial Array Self Refresh (Read only when Mobile SDR support is disabled). “000”: Full array (Banks 0, 1, 2 and 3) “001”: Half array (Banks 0 and 1) “010”: Quarter array (Bank 0) “101”: One-eighth array (Bank 0 with row MSB = 0) “110”: One-sixteenth array (Bank 0 with row MSB = 00)

106.15 Vendor and device identifiers

The core has vendor identifier 0x04 (ESA) and device identifier 0x00F. For description of vendor and device identifier see GRLIB IP Library User’s Manual.

106.16 Implementation

106.16.1Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User’s Manual). The core makes use of synchronous reset and resets a subset of its internal registers. See the documentation for the *syncrst* VHDL generic for information on asynchronous reset affecting external signals.

106.16.2Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

106.17 Configuration options

Table 2097 shows the configuration options of the core (VHDL generics).

***Table 2097.*Configuration options**

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0x1FFFFFFF.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.	0 - 16#FFF#	16#E00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space. Default I/O area is 0x20000000 - 0x2FFFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR1 defining I/O address space.	0 - 16#FFF#	16#E00#
ramaddr	ADDR field of the AHB BAR2 defining RAM address space. Default RAM area is 0x40000000-0x7FFFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR2 defining RAM address space.	0 -16#FFF#	16#C00#
paddr	ADDR field of the APB BAR configuration registers address space.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR configuration registers address space.	0 - 16#FFF#	16#FFF#
wprot	RAM write protection.	0 - 1	0
invclk	Inverted clock is used for the SDRAM.	0 - 1	0
fast	Enable fast SDRAM address decoding.	0 - 1	0
romasel	log2(PROM address space size) - 1. E.g. if size of the PROM area is 0x20000000 romasel is log2(2^29)-1 = 28.	0 - 31	28
sdrasel	log2(RAM address space size) - 1. E.g if size of the RAM address space is 0x40000000 sdrasel is log2(2^30)-1= 29.	0 - 31	29
srbanks	Number of SRAM banks.	0 - 5	4
ram8	Enable 8-bit PROM and SRAM access.	0 - 1	0
ram16	Enable 16-bit PROM and SRAM access.	0 - 1	0
sden	Enable SDRAM controller.	0 - 1	0
sepbus	SDRAM is located on separate bus.	0 - 1	1
sdbits	32 or 64 -bit SDRAM data bus.	32, 64	32
sdlsb	Can be used to shift address lines used for SDRAM. Leave at default value.	-	2
oepol	Select polarity of drive signals for data pads. 0 = active low, 1 = active high.	0 - 1	0
syncrst	When syncrst is set to 0 then the output registers of the core have asynchronous reset. When syncrst is set to 1 only synchronous reset is used within the core and the output signals are instead gated with the reset signal.	0 - 1	0
pageburst	Line burst read of length 8 when 0, page burst read when 1, programmable read burst type when 2.	0 - 2	0
scantest	Enable scan test support (connects test oen to output enables, handles asynchronous reset case).	0 - 1	0
mobile	Enable Mobile SDRAM support 0: Mobile SDRAM support disabled 1: Mobile SDRAM support enabled but not default 2: Mobile SDRAM support enabled by default 3: Mobile SDRAM support only (no regular SDR support)	0 - 3	0

106.18 Signal descriptions

Table 2098 shows the interface signals of the core (VHDL ports).

***Table 2098.*Signal descriptions**

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
MEMI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Bus ready strobe	Low
	BEXCN	Input	Bus exception	Low
	WRN[3:0]	Input	SRAM write enable feedback signal	Low
	BWIDTH[1:0]	Input	Sets the reset value of the PROM data bus width field in the MCFG1 register	High
	SD[31:0]	Input	SDRAM separate data bus	High
MEMO	ADDRESS[31:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data	-
	SDDATA[63:0]	Output	Sdram memory data	-
	RAMSN[4:0]	Output	SRAM chip-select	Low
	RAMOEN[4:0]	Output	SRAM output enable	Low
	IOSN	Output	Local I/O select	Low
	ROMSN[1:0]	Output	PROM chip-select	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0].	Low
	MBEN[3:0]	Output	Byte enable: MBEN[0] corresponds to DATA[31:24], MBEN[1] corresponds to DATA[23:16], MBEN[2] corresponds to DATA[15:8], MBEN[3] corresponds to DATA[7:0].	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus.Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0].	Low/High
	VBDRIVE[31:0]	Output	Vectored I/O-pad drive signals.	Low/High
	SVBDRIVE[63:0]	Output	Vectored I/O-pad drive signals for separate sdram bus.	Low/High
	READ	Output	Read strobe	High
	SA[14:0]	Output	SDRAM separate address bus	High
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-

Table 2098. **Signal descriptions**

Signal name	Field	Type	Function	Active
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
WPROT	WPROTHIT	Input	Unused	-
SDO	SDCASN	Output	SDRAM column address strobe	Low
	SDCKE[1:0]	Output	SDRAM clock enable	High
	SDCSN[1:0]	Output	SDRAM chip select	Low
	SDDQM[7:0]	Output	SDRAM data mask: DQM[7] corresponds to DATA[63:56], DQM[6] corresponds to DATA[55:48], DQM[5] corresponds to DATA[47:40], DQM[4] corresponds to DATA[39:32], DQM[3] corresponds to DATA[31:24], DQM[2] corresponds to DATA[23:16], DQM[1] corresponds to DATA[15:8], DQM[0] corresponds to DATA[7:0].	Low
	SDRASN	Output	SDRAM row address strobe	Low
	SDWEN	Output	SDRAM write enable	Low

* see GRLIB IP Library User's Manual

106.19 Library dependencies

Table 2099 shows libraries used when instantiating the core (VHDL libraries).

Table 2099. **Library dependencies**

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals	Memory bus signals definitions
		Components	SDMCTRL component
ESA	MEMORYCTRL	Component	Memory controller component declaration

106.20 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined on the example designs port map and connected to the memory controller. System clock and reset are generated by GR Clock Generator and Reset Generator.

Memory controller decodes default memory areas: PROM area is 0x0 - 0x1FFFFFFF, I/O-area is 0x20000000-0x3FFFFFFF and RAM area is 0x40000000 - 0x7FFFFFFF. SDRAM controller is enabled. SDRAM clock is synchronized with system clock by clock generator.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
```

```
library esa;
use esa.memoryctrl.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;

    -- memory bus
    address  : out   std_logic_vector(27 downto 0); -- memory bus
    data     : inout std_logic_vector(31 downto 0);
    ramsn    : out   std_logic_vector(4 downto 0);
    ramoen   : out   std_logic_vector(4 downto 0);
    rwen     : inout std_logic_vector(3 downto 0);
    romsn    : out   std_logic_vector(1 downto 0);
    iosn     : out   std_logic;
    oen      : out   std_logic;
    read     : out   std_logic;
    writen   : inout std_logic;
    brdyn    : in    std_logic;
    bexcn    : in    std_logic;
-- sdram i/f
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0) -- optional sdram data
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo : sdram_out_type;

  signal wprot : wprot_out_type;  -- dummy signal, not used
  signal clkm, rstn : std_ulogic; -- system clock and reset

-- signals used by clock and reset generators
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;

  signal gnd : std_ulogic;

begin

  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                                tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  -- Memory controller
  mctrl0 : mctrl generic map (srbanks => 1, sden => 1)
```

```
      port map (rstn, clkm, memi, memo, ahbsi, ahbso(0), apbi, apbo(0), wprot, sdo);

  -- memory controller inputs not used in this configuration
  memi.brdyn <= '1'; memi.bexcn <= '1'; memi.wrn <= "1111";
  memi.sd <= sd;

  -- prom width at reset
  memi.bwidth <= "10";

  -- I/O pads driving data memory bus data signals
  datapads : for i in 0 to 3 generate
      data_pad : iopadv generic map (width => 8)
      port map (pad => data(31-i*8 downto 24-i*8),
                o => memi.data(31-i*8 downto 24-i*8),
                en => memo.bdrive(i),
                i => memo.data(31-i*8 downto 24-i*8));
  end generate;

  -- connect memory controller outputs to entity output signals
  address <= memo.address; ramsn <= memo.ramsn; romsn <= memo.romsn;
  oen <= memo.oen; rwen <= memo.wrn; ramoen <= "1111" & memo.ramoen(0);
  sa <= memo.sa;
  writen <= memo.writen; read <= memo.read; iosn <= memo.iosn;
  sdcke <= sdo.sdcke; sdwen <= sdo.sdwen; sdcsn <= sdo.sdcsn;
  sdrasn <= sdo.rasn; sdcasn <= sdo.casn; sddqm <= sdo.dqm;
end;
```

107 MDIO_CTRL - MDIO Serial Bus Controller

107.1 Overview

The MDIO controller is an MDIO bus master with built in clock generation. It also provides the following features:

- Support for 32 PHYs on one single MDIO interface.
- Automated auto negotiation configuration of selected PHYs.
  - Initial setup ran after reset.
  - Periodic checking of auto negotiation results.
- Interface compatible with GRETH_GBIT.
- Software issued MDIO transactions via APB register file.
- Handling and synchronization of an external interrupt.
- Build-time configurable output and input delays.

The figure below illustrates the flow of data and interfaces within and to the outside of the MDIO controller.

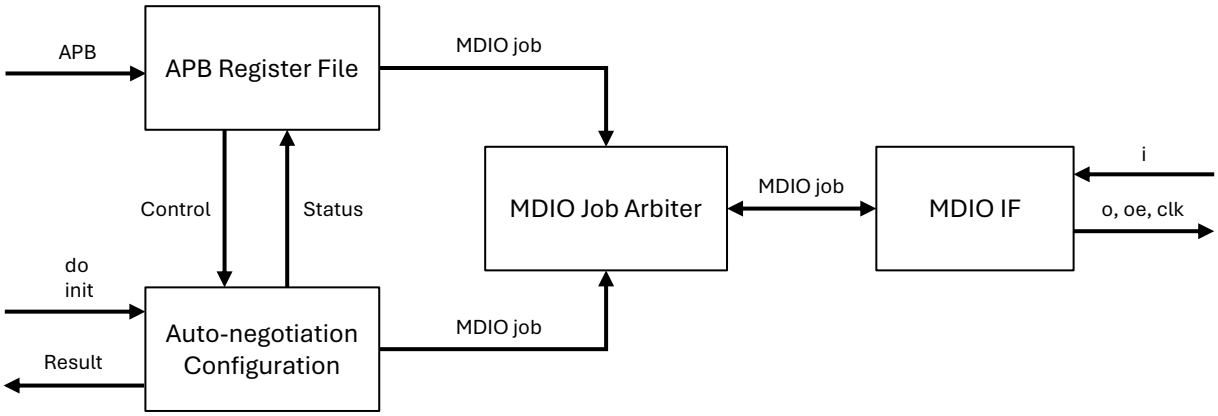


Figure 296. MDIO_CTRL block diagram

107.2 APB Register File

The APB register file contains the logic needed for the APB interface. It offers various status registers and some control registers. Additionally, it offers an interface to create MDIO transactions using the MDIO job register.

107.3 Auto Negotiation Configuration

The auto negotiation block performs automated reset and enabling of auto negotiation for the PHYs specified by the phy_init_mask generic. If the negotiation can be completed the results are gathered and speed and duplex parameters generated, saved and output on the results interface. The auto negotiation configuration block operates in two different modes: initialization and periodic.

107.3.1 Initialization

The initialization mode of the configuration block runs once after reset, if the corresponding signal is asserted when the reset is lifted. Which PHYs the initialization tries to initialize is decided with the *phy_init_mask* VHDL generic.

The block tries to reset and enable auto negotiation. The number of times the block tries to do this, before giving up on the PHY and going to the next one, is decided by the *phy_init_retry* build-time parameter.

If the PHY is successfully reset, the initialization checks for the auto negotiation results. If the PHY and its partner PHY has been able to complete the operation, the results are read out. The speed and duplex parameters are calculated and then published on the results interface. The parameters are also available in the register file. If the PHY lacks the capability of auto negotiation, the initialization will publish a 10 MBit/s half duplex result.

The whole process is repeated for each PHY index indicated by the generic *phy_init_mask*. After the initialization process is completed, the configuration block tries to enter the periodic mode.

107.3.2 Periodic

When the configuration block enters the periodic mode, a register bit in the control register is checked. If the bit is not set the configuration block will enter an idle state.

The periodic mode will cycle through the PHYs that were successfully initialized and support auto negotiation. The process is the same as the latter half of the initialization mode (i.e. after reset and auto negotiation capability check). The periodic mode will be active until the register bit is unset. The results of the periodic checking will also be published on the results interface.

107.3.3 Auto-negotiation results

The auto negotiation interface bundles the signals related to the automated configuration functionality.

The *perform_startup_init* signal controls whether the automated configuration process will be ran. If it is asserted with a '1' during the release of the core reset, the process will start, otherwise it will be skipped.

The *aneg_results* vector contains the different speed settings calculated as a result of the automated configuration process. Bit 0 indicates whether the link is 10 Mbit/s or 100 Mbit/s. This bit will be 0 if the *gbit* indicator is asserted. Bit 1 indicates whether the link negotiated is a Gbit link or not, if asserted then bit 0 will be '0' as a result. Bit 2 indicates whether the link is full duplex or not, a '1' means that the link has negotiated full duplex.

The *aneg_valid* vector contains one bit per PHY, where the bit index corresponds to the PHY address. An asserted bit in the vector qualifies the *aneg_results* for the PHY corresponding to the asserted bit's index. The bit will stay asserted for one clock cycle and the results vector is only valid when a bit in the valid vector is asserted. The results should therefore be stored by the recipient.

107.4 MDIO Job Arbiter

The MDIO job arbiter block is responsible for forwarding the MDIO jobs. It ensures that only one job is running at a time and that the results of the job ends up with the issuer of the job. The jobs are prioritized as follows:

1 Initialization configuration jobs

2 APB jobs

3 Periodic configuration jobs

107.5 MDIO IF

The communication over the MDIO bus is handled by the MDIO IF. It acts as the bus master and clock generator for the bus. When a job request is received, from the job arbiter, the MDIO IF will create the corresponding transaction on the MDIO bus. The results, in the case of a read transactions, are reported back to the job arbiter and the MDIO IF starts listening for another job. The switching of

the data output can be delayed at a clock cycle granularity. The same goes for the sampling of the input data. Both are controlled by VHDL generics. The clock period of the generated clock is also controlled by a VHDL generic.

An output enable is provided to interface the data input and output signals to a tri-state buffer. The polarity of the output enable can be controlled by the output enable polarity generic.

107.6 APB Issued MDIO Jobs

MDIO transactions can be created via the APB interface. There is an MDIO job register which is used for both control and status. A job is created when writing to the register. The data field is required for write transactions as it is the source of the data to be transmitted to the specified PHY's register. In the case of a read transaction the read data will be placed in the data field when the done status bit is asserted. A new write to the register should not be performed until the current APB issued job has been completed. For information about the individual fields and layout of the MDIO job register, please refer to the register map section.

107.7 Interrupt generation

The MDIO controller contains an interrupt handler for a single source MDIO interrupt.

The interrupt signal is synchronized and the first detected flank is deemed as the active flank. Care should therefore be taken to ensure that the input interrupt is not asserted when the core leaves reset. If an active flank is encountered, the interrupt handler will assert its output interrupt and set the corresponding status bit in the register file. The interrupt can be cleared by writing to the status register. The interrupt handling can also be disabled by clearing the IRQ enable bit in the general control register.

107.8 Implementation

107.8.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package, please refer to the GRLIB User’s Manual for details.

The core will add reset for all registers, except synchronization registers, if the GRLIB config package setting grlib_sync_reset_enable_all is set.

The core will use asynchronous reset for all registers if the GRLIB config package setting grlib_async_reset_enable is set.

107.9 Registers

The core is programmed through registers mapped into APB address space. The register map is divided into two segments: the general MDIO controller registers and PHY-specific registers.

*Table 2100.*MDIO_CTRL registers

APB address offset	Register
0x00	MDIO Status Register
0x04	MDIO Control Register
0x08	APB MDIO Job Register
0x0C	Init Bit Mask Register
0x80 + 4*(PHY index/address)	PHY Specific Registers

107.9.1 MDIO Status Register

Table 2101. 0x00 - STS - MDIO Status register

31 … 5	4	3	2	1	0
RESERVED	ST		P	B	I
0	0		-	0	0
r	r		r	r	r

4: 3 Configuration state (ST). The current state of the configuration block state machine.
0x0: Initialization
0x1: Periodic
0x2: Done

2 Perform init configuration (P). A mirror of the input signal.

1 MDIO busy (B). 1 if the MDIO interface is busy with a transaction.

0 Active interrupt (I). 1 if an interrupt is active. Can be cleared by writing the status register.

107.9.2 MDIO Control Register

Table 2102. 0x04 - CTRL - MDIO Control register

31 … 2	1	0
RESERVED	P	I
0	1	1
r	rw	rw

1 Periodic configuration enable (P). Set the field to 1 if the configuration block should perform periodic checking of the link auto negotiation results. Once disabled the function can't be enabled again unless a reset of the core is performed.

0 Interrupt enable (I). Set the field to 1 if incoming interrupts should be output on the IRQ output and the corresponding status bit should be set.

107.9.3 APB MDIO Job Register

Table 2103. 0x08 - JOB - APB MDIO Job Register

31 16	15 11	10 6	5	4 3	2	1	0
DATA	REG	PHY	W	RES	L	D	A
0	0	0	0	00	0	0	0
rw	rw	rw	rw	r	r	r	r

31: 16 Data (DATA). The data to be either written or read from the specified PHY and register. A write transaction will source its data from this field while a read transaction will place the read data in this field. Qualified by the Job Done and Link Failure fields.

A transaction can be initiated by writing to this register when the APB MDIO job is not busy (Job Active is 0).

For read transactions, the data read from the PHY will be placed in this field when the Job Done field is asserted by the core via a 1.

15: 11 Reg address (REG). The address of the PHY register the transaction is intended for.

10: 6 PHY address (PHY). The address of the PHY the transaction is intended for.

5 Write not read (W). Controls whether the transaction is a read transaction or a write transaction. A 1 will result in a write transaction, while a 0 will result in a read transaction.

2 Link failure (L). Set if a read transaction failed due to the PHY not reacting in accordance with the MDIO protocol. Acts as a disqualifier of the read data, trumps job done.

1 Job done (D). Set if an APB MDIO job is completed. Acts as a qualifier for the read data in case of a read transaction. Cleared when the APB MDIO job register is read.

0 Job active (A). Set if an APB MDIO job is currently active (either to be issued or transaction in action).

107.9.4 Init Bit Mask Register

*Table 2104.*0x0C - MSK - Init Bit Mask Register

31 0
MSK
-
r

31:0 Initialization build-time parameter bit mask (MSK). It contains the value of the *phy_init_mask* VHDL generic. The bit mask controls which PHYs are automatically initialized by the configuration block when the reset is released. By extension it also controls which of the PHYs are subject to the periodic configuration.

107.9.5 PHY Specific Register

Table 2105. 0x80 + 4* (PHY index/address) - PHY - PHY Specific Registers

31 6	5	4 3	2 0
RESERVED	AM	AS	IS
0	0	00	000
r	r	r	r

5 Auto-negotiation Mode (AM): 0 - Half duplex, 1 - Full duplex

4: 3 Auto-negotiation Speed (AS): 00b - 10 MBit/s, 01b - 100 MBit/s, 10b - 1000 Mbit/s

Table 2105. 0x80 + 4* (PHY index/address) - PHY - PHY Specific Registers

2:	0	Initialization status (IS). 0x0: Not performed. Skipped due to mask bit not set or init not performed at all. 0x1: Running. Currently running. 0x2: Done. Performed successfully. 0x3: Failed. Auto negotiation could not be enabled. 0x4: Timeout. A read transaction resulted in a link failure or the PHY could not be reset within the retry limit. 0x5: Deferred. Auto negotiation was not performed, most likely due to a missing Ethernet link.

107.10 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x0DF. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

107.11 Configuration options

Table 2106 shows the configuration options of the core (VHDL generics).

*Table 2106.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Index of the interrupt line.	0 - NAHBIRQ-1	0
mdio_clk_divisor	MDIO clock period. Sets the divisor value use to generate the MDIO clock. The MDIO clock frequency will be clk/(2*(mdio_clk_divisor+1)).		1
oe_polarity	Output-enable polarity. It controls the active signal level for the MDIO output enable. A polarity of '1' will result in the MDIO controller using the level '1' to assert active output from the MDIO interface.	0 - 1	1
mdio_output_delay	MDIO output switching delay. It controls the delay of the output switching with regard to the rising edge of the MDIO clock. The delay is specified in core clock cycles. The delay must be greater or equal to 1, but care should be taken that the delay does not spill into the next MDIO clock cycle. According to 802.3, the output must be stable at a minimum of 10 ns before and after the rising edge of the MDIO clock.		1
mdio_input_delay	MDIO input sampling delay. It controls the delay of the input sampling by the specified number of core clock cycles. The delay must be greater or equal to 2, but care should be taken that the delay does not spill into the next MDIO clock cycle. According to 802.3, the PHY output delay can be anywhere between 0 and 300 ns. The MDIO controller must therefore be able to delay the sampling of the input in relation to the rising edge of the MDIO clock.	2 - 2147483647	2

*Table 2106.*Configuration options

Generic name	Function	Allowed range	Default
phy_init_mask	PHY initialization mask. It defines which PHYs the auto negotiation configuration block will try to initialize. A '1' will tell the configuration to try to initialize the PHY with the address corresponding to the bit index. A '1' at bit position 20 will lead to the initialization of the PHY responding to address 20.	0 - 16#FFFFFF#	0
phy_reset_retry_count	PHY initialization reset retry count. During the initialization mode of the configuration, the PHY might be unresponsive. If this is detected during the reset phase of the initialization mode, the core will retry the operation as many times as the value of this build-time parameter. If the retry count is exceeded the PHY will be skipped and will not be included in the periodic checking.		0

107.12 Signal descriptions

Table 2107 shows the interface signals of the core (VHDL ports).

*Table 2107.*Signal definitions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AHB reset	Low
CLK	N/A	Input	AHB clock	-
MDIO_CLK	N/A	Output	MDIO clock	-
MDIO_I	N/A	Input	MDIO data input	-
MDIO_O	N/A	Output	MDIO data output	-
MDIO_OE	N/A	Output	Output enable	-
MDIO_IRQ	N/A	Input	MDIO Interrupt	High
PEFORM_STARTUP_INIT	N/A	Input	Initialize PHY:S after reset	High
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
ANEG_VALID	N/A	Output	Corresponding PHY (auto negotiation	-
ANEG_RESULTS	N/A	Output	Results from auto negotiation	-

107.13 Library dependencies

Table 2108 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2108.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions

108 MEMSCRUB - AHB Memory Scrubber and Status Register

108.1 Overview

The memory scrubber monitors an AMBA AHB bus for accesses triggering an error response, and for correctable errors signaled from fault tolerant slaves on the bus. The core can be programmed to scrub a memory area by reading through the memory and writing back the contents using a locked read-write cycle whenever a correctable error is detected. It can also be programmed to initialize a memory area to known values.

The memory scrubber core is largely backwards compatible with the AHBSTAT core, and can replace it in many cases. Unlike AHBSTAT, the scrubber's registers are accessed through the AMBA AHB bus.

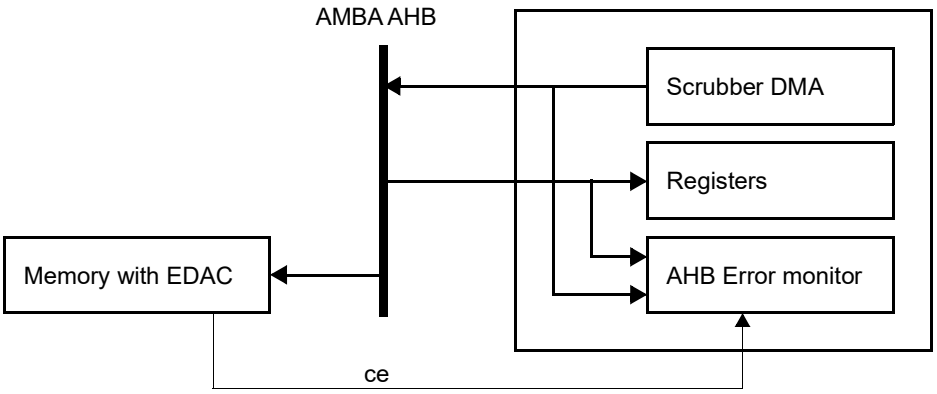


Figure 297. Memory scrubber block diagram

108.2 Operation

108.2.1 Errors

All AMBA AHB bus transactions are monitored and current HADDR, HWRITE, HMASTER and HSIZE values are stored internally. When an error response (HRESP = "01") is detected, an internal counter is increased. When the counter increments from "threshold" to "threshold+1" (where "threshold" is a user-programmable value), the status and address register contents are frozen and the New Error (NE) bit is set to one. At the same time an interrupt is generated, as described hereunder.

The default threshold is zero and enabled on reset so the first error on the bus will generate an interrupt.

Note that many of the fault tolerant units containing EDAC signal an uncorrectable error as an AMBA error response, so that it can be detected by the processor as described above.

108.2.2 Correctable errors

Not only error responses on the AHB bus can be detected. Many of the fault tolerant units containing EDAC have a correctable error signal which is asserted each time a correctable error is detected. When such an error is detected, the effect will be the same as for an AHB error response. The only difference is that the Correctable Error (CE) bit in the status register is set to one when a correctable error is detected. Correctable and uncorrectable errors use separate counters and threshold values.

When the CE bit is set, the interrupt routine can acquire the address containing the correctable error from the failing address register and correct it. When it is finished it resets the CE bit and the monitoring becomes active again. Interrupt handling is described in detail hereunder.

The correctable error signals from the fault tolerant units should be connected to the *scrubi.cerror* input signal vector of the AHB status register core, which is or-ed internally and if the resulting signal is asserted, it will have the same effect as an AHB error response.

108.2.3 Scrubbing

The memory scrubber can be commanded to scrub a certain memory area, by writing a start and end address to the scrubbers start/end registers, followed by writing “00” to the scrub mode field and ‘1’ to the scrub enable bit in the scrubber control register.

After starting, the core will proceed to read the memory region in bursts. The burst size is fixed (set by the *burstlen* generic) and typically tuned to match the cache-line size or native block size of the slave. When a correctable error is detected, the scrubber performs a locked read-write cycle to correct the error, and then resumes the scrub operation.

If the correctable error is detected in the middle of a burst, the following read in the burst is completed before the read-write cycle begins. The core can handle the special case where that access also had a correctable error within the same locked scrub cycle.

If an uncorrectable error is detected, that location is left untouched.

Note that the status register functionality is running in parallel with the scrubber, so correctable and uncorrectable errors will be logged as usual. To prevent double logging, the core masks out the (expected) correctable error arising during the locked correction cycle.

To allow normal access to the bus, the core sleeps for a number of cycles between each burst. The number of cycles can be adjusted in the config register.

If the ID bit is set in the config register, the core will interrupt when the complete scrub is done.

108.2.4 Scrubber error counters

The core keeps track of the number of correctable errors detected during the current scrub run and the number of errors detected during processing of the current “count block”. The size of the count block is a fixed power of two equal or larger than the burst length (set by the *countlen* generic).

The core can be set up to interrupt when the counters exceed given thresholds. When this happens, the NE bit, plus one of the SEC/SBC bits, is set in the status register. An interrupt will only be generated when a counter increments from “threshold” to “threshold+1”. Additional increments when a counter is already larger than its threshold value will not generate interrupts.

108.2.5 External start and clear

If the ES bit is set in the config register, the scrub enable bit is set automatically when the start input signal goes high. This can be used to set up periodic scrubbing.

The external input signal clrerr can be used to clear the global error counters. If this is connected to a timer, it is possible to count errors that have occurred within a specific unit of time. This signal can be disabled through the EC bit in the config register.

108.2.6 Memory regeneration

The regeneration mode performs the same basic function as the scrub mode, but is optimised for the case where many (or all) locations have correctable errors.

In this mode, the whole memory area selected is scrubbed using locked read/write bursts.

If an uncorrectable error is encountered during the read burst, that burst block is processed once again using the regular scrub routine, and the regeneration mode resumes on the following block. This avoids overwriting uncorrectable error locations.

108.2.7 Initialization

The scrubber can be used to write a pre-defined pattern to a block of memory. This is often necessary on EDAC memory before it can be used.

Before running the initialization, the pattern to be written to memory should be written into the scrubber initialization data register. The pattern has the same size as the burst length, so the corresponding number of writes to the initialization data register must be made.

108.2.8 Interrupts

The interrupt is generated on the line selected by the *hirq* VHDL generic. The interrupt is connected to the interrupt controller to inform the processor of the event.

After an interrupt is generated, either the NE bit or the DONE bit in the status register is set, to indicate which type of event caused the interrupt.

The normal procedure is that an interrupt routine handles the error with the aid of the information in the status registers. When it is finished it resets the NE bit and error counters in the AHB status register or the DONE bit in the scrubber status register, and the monitoring becomes active again. Error interrupts can be generated for both AMBA error responses and correctable errors as described above.

108.2.9 Mode switching

Switching between scrubbing and regeneration modes can be done on the fly during a scrub by modifying the MODE field in the scrubber configuration register. The mode change will take effect on the following scrub burst.

If the address range needs to be changed, then the core should be stopped before updating the registers. This is done by clearing the SCEN bit, and waiting for the ACTIVE bit in the status register to go low. An exception is when making the range larger (i.e. increasing the end address or decreasing the start address), as this can be done on the fly.

108.2.10Dual range support

The scrubber can work over two non-overlapping memory ranges. This feature is enabled by writing the start/end addresses of the second range into the scrubber's second range start/end registers and setting the SERA bit in the configuration register. The two address ranges should not overlap.

108.3 Registers

The core is programmed through registers mapped into an I/O region in the AHB address space. Only 32-bit single-accesses to the registers are supported.

Table 2109. Memory scrubber registers

AHB address offset	Registers
0x00	AHB Status register
0x04	AHB Failing address register
0x08	AHB Error configuration register
0x0C	Reserved
0x10	Scrubber status register
0x14	Scrubber configuration register
0x18	Scrubber range low address register
0x1C	Scrubber range high address register
0x20	Scrubber position register
0x24	Scrubber error threshold register
0x28	Scrubber initialization data register
0x2C	Scrubber second range start address register
0x30	Scrubber second range end address register

108.3.1 AHB Status Register

Table 2110. 0x00 - AHBS - AHB Status register

31 22	21 14	13	12	11	10	9	8	7	6 3	2 0
CECNT	UECNT	DONE	RES	SEC	SBC	CE	NE	HWRITE	HMASTER	HSIZE
0	0	0	0	0	0	0	0	NR	NR	NR
rw	rw	r	r	rw	rw	rw	rw	r	r	r

31: 22 CECNT: Global correctable error count

21: 14 UECNT: Global uncorrectable error count

13 DONE: Task completed. (read-only)

This is a read-only copy of the DONE bit in the scrubber status register.

12 RESERVED

11 SEC: Scrubber error counter threshold exceeded. Asserted together with NE.

10 SBC: Scrubber block error counter threshold exceeded. Asserted together with NE.

9 CE: Correctable Error. Set if the detected error was caused by a correctable error and zero otherwise.

8 NE: New Error. Deasserted at start-up and after reset. Asserted when an error is detected. Reset by writing a zero to it.

7 The HWRITE signal of the AHB transaction that caused the error.

6: 3 The HMASTER signal of the AHB transaction that caused the error.

2: 0 The HSIZE signal of the AHB transaction that caused the error

108.3.2 AHB Failing Address Register

Table 2111. 0x04 - AHBFAR - AHB Failing address register

31 0
AHB FAILING ADDRESS
NR
r

31: 0 The HADDR signal of the AHB transaction that caused the error.

108.3.3 AHB Error Configuration Register

Table 2112. 0x08 - AHBERC - AHB Error configuration register

31 22	21 14	13 2	1	0
CORRECTABLE ERROR COUNT THRESHOLD	UNCORR. ERROR COUNT THRESH.	RESERVED	CECTE	UECTE
0	0	0	0	0
rw	rw	r	rw	rw

31: 22 Interrupt threshold value for global correctable error count

21: 14 Interrupt threshold value for global uncorrectable error count

13: 2 RESERVED

1 CECTE: Correctable error count threshold enable

0 UECTE: Uncorrectable error count threshold enable

108.3.4 Scrubber Status Register

Table 2113. 0x10 - STAT - Scrubber status register

31 22	21 14	13	12 5	4 1	0
SCRUB RUN ERROR COUNT	BLOCK ERROR COUNT	DONE	RESERVED	BURSTLEN	ACTIVE
0	0	0	0	*	0
r	r	wc	r	r	r

31: 22	Number of correctable errors in current scrub run (read-only)
21: 14	Number of correctable errors in current block (read-only)
13	DONE: Task completed. Needs to be cleared (by writing zero) before a new task completed interrupt can occur.
12: 5	RESERVED
4: 1	Burst length in 2-log of AHB bus cycles; “0000”=1, “0001”=2, “0010”=4, “0011”=8, ...
0	Current state: 0=Idle, 1=Running (read-only)

108.3.5 Scrubber Configuration Register

*Table 2114.*0x14 - CONFIG - Scrubber configuration register

31 16	15 8	7	6	5	4	3 2	1	0
RESERVED	DELAY	IRQD	EC	SERA	LOOP	MODE	ES	SCEN
0	0	0	0	0	0	0	0	0
r	rw	rw	r	rw	rw	rw	rw	rw

31: 16	RESERVED
15: 8	Delay time between processed blocks, in cycles
7	Interrupt when scrubber has finished
6	External clear counter enable
5	Second memory range enable
4	Loop mode, restart scrubber when run finishes
3: 2	Mode (00=Scrub, 01=Regenerate, 10=Initialize, 11=Undefined)
1	External start enable
0	Enable

108.3.6 Scrubber Range Low Address Register

Table 2115. 0x18 - RANGEL - Scrubber range low address register

31 0
SCRUBBER RANGE LOW ADDRESS
0
rw*

31: 0	The lowest address in the range to be scrubbed The address bits below the burst size alignment are constant ‘0’

108.3.7 Scrubber Range High Address Register

Table 2116. 0x1C - RANGEH - Scrubber range high address register

31	0
SCRUBBER RANGE HIGH ADDRESS	
0	
rw*	

31: 0 The highest address in the range to be scrubbed

The address bits below the burst size alignment are constant ‘1’

108.3.8 Scrubber Position Register

Table 2117. 0x20 - POS - Scrubber position register

31	0
SCRUBBER POSITION	
0	
rw	

31: 0 The current position of the scrubber while active, otherwise zero.

The address bits below the burst size alignment are constant ‘0’

108.3.9 Scrubber Error Threshold Register

Table 2118. 0x24 - ERROR - Scrubber error threshold register

31 22	21 14	13 2	1	0
RECT	BECT	RESERVED	RECTE	BECTE
0	0	0	0	0
rw	rw	r	rw	rw

31: 22	Interrupt threshold value for current scrub run correctable error count
21: 14	Interrupt threshold value for current scrub block correctable error count
13: 2	RESERVED
1	RECTE: Scrub run correctable error count threshold enable
0	BECTE: Scrub block correctable error count threshold enable

108.3.10Scrubber Initialization Data Register

Table 2119. 0x28 - INIT - Scrubber initialization data register (write-only)

31 0
SCRUBBER INITIALIZATION DATA
-
w

31: 0	Part of data pattern to be written in initialization mode. A write operation assigns the first part of the buffer and moves the rest of the words in the buffer one step.

108.3.11Scrubber Second Range Low Address Register

Table 2120. 0x2C - RANGEL2 - Scrubber second range low address register

31 0
SCRUBBER RANGE LOW ADDRESS
0
rw*

31: 0	The lowest address in the second range to be scrubbed (if SERA=1) The address bits below the burst size alignment are constant ‘0’

108.3.12Scrubber Second Range High Address Register

Table 2121. 0x30 - RANGEH2 - Scrubber second range high address register

31 0
SCRUBBER RANGE HIGH ADDRESS
0
rw*

31: 0	The highest address in the second range to be scrubbed (if SERA=1) The address bits below the burst size alignment are constant ‘1’

108.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x057. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

108.5 Implementation

108.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

108.5.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

108.6 Configuration options

Table 2122 shows the configuration options of the core (VHDL generics).

*Table 2122.*Configuration options

Generic	Function	Allowed range	Default
hmindex	AHB master index	0 - NAHBMST-1	0
hsindex	AHB slave index	0 - NAHBSLV-1	0
ioaddr	AHB slave register area address	0 - 16#FFF#	0
iomask	AHB slave register area address mask	0 - 16#FFF#	16#FFF#
hirq	Interrupt line driven by the core	0 - 16#FFF#	0
nftslv	Number of FT slaves connected to the cerror vector	1 - NAHBSLV-1	3
memwidth	Width of accesses to scrubbed memory slave in bits	32, 64, ..., 1024	AHBDW
burstlen	Length of burst accesses to scrubbed memory slave	2, 4, 8, 16, ...	2
countlen	Length of blocks used for block error count	burstlen x (1,2,4,8 ...)	8

108.7 Signal descriptions

Table 2123 shows the interface signals of the core (VHDL ports).

*Table 2123.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signal	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
SCRUBI	CERROR	Input	Correctable Error Signals	High
	CLRCOUNT	Input	Clear global error counters	High
	START	Input	External start signal	High

* see GRLIB IP Library User's Manual

108.8 Library dependencies

Table 2124 shows libraries used when instantiating the core (VHDL libraries).

*Table 2124.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MISC	Component	Component declaration

108.9 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory scrubber. There are three Fault Tolerant units with EDAC connected to the scrubber's *cerror* vector. The connection of the different memory controllers to external memory is not shown.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;
    --other signals
    ....
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo, sdo2: sdctrl_out_type;

  signal sdi : sdctrl_in_type;

  signal aramo : ahbram_out_type;

  -- correctable error vector
  signal scrubi : memscrub_in_type;

begin

  -- AMBA Components are defined here ...

  -- AHB Memory Scrubber and Status Register
```

```
  mscrub0 : memscrub
    generic map(hmindex => 4, hsindex => 5, ioaddr => 16#C00#,
                hirq => 11, nftslv => 3);
    port map(rstn, clkm, ahbmi, ahbmo(4), ahbsi, ahbso(5), scrubi);

  scrubi.start <= '0'; scrubi.clrcount <= '0';
  scrubi.cerror(3 to NAHBSLV-1) <= (others => '0');

  --FT AHB RAM
  a0 : ftahbram
    generic map(hindex => 1, haddr => 1, tech => inferred,  kbytes => 64,
                pindex => 4, paddr => 4, edacen => 1, autoscrub => 0,
                errcnt => 1, cntbits => 4)
    port map(rst, clk, ahbsi, ahbso(1), apbi, apbo(4), aramo);

  scrubi.cerror(0) <= aramo.ce;

  -- SDRAM controller
  sdc : ftsdctrl
    generic map (hindex => 3, haddr => 16#600#, hmask => 16#F00#, ioaddr => 1,
                 fast => 0, pwron => 1, invclk => 0, edacen => 1, errcnt => 1,
                 cntbits => 4)
    port map (rstn, clk, ahbsi, ahbso(3), sdi, sdo);

  stati.cerror(1) <= sdo.ce;

  -- Memory controller
  mctrl0 : ftsrctrl
    generic map (rmw => 1, pindex => 10, paddr => 10, edacen => 1, errcnt => 1,
                 cntbits => 4)
    port map (rstn, clk, ahbsi, ahbso(0), apbi, apbo(10), memi, memo, sdo2);

  scrubi.cerror(2) <= memo.ce;

end;
```

109 MMA - Memory Mapped AMBA bridge

109.1 Overview

The bridge acts as a slave on the memory bus and provides a master interface on the AMBA AHB bus. A read/write access issued towards the slave memory bus interface for the defined memory range is transferred to the AHB master interface and issued on the AMBA bus.

109.2 Operation

109.2.1 Read

Only 32-bit read accesses is allowed for the slave memory interface. When a read access is detected and decoded by the bridge, the bus ready signal is combinatorially deasserted. When corresponding AMBA access is done, the bus ready signal is asserted and the data is driven on the memory bus. When output enable / chip select signal is detected deasserted the data bus is tri-stated.

109.2.2 Write

Only 32-bit write accesses is allowed for the slave memory interface. When a write access is detected and decoded by the bridge, the bus ready signal is combinatorially deasserted. The bus ready signal is reasserted as soon as the write data is stored by the bridge. The bridge can store a write access when the last access has completed on the AMBA bus.

109.3 Default configuration

The default configuration is set by the defcfg signal. The configuration is arranged in the following order:

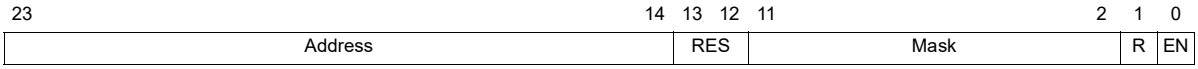
Table 2125.

DEFCFG bits	Configuration
407 - 380	Reconfiguration area
379 - 356	Config for CSN[9]
355 - 332	Config for CSN[8]
331 - 308	Config for CSN[7]
307 - 284	Config for CSN[6]
283 - 260	Config for CSN[5]
259 - 236	Config for CSN[4]
235 - 212	Config for CSN[3]
211 - 188	Config for CSN[2]
187 - 164	Config for CSN[1]
163 - 140	Config for CSN[0]
139 - 126	Address map for CSN[9]
125 - 112	Address map for CSN[8]
111 - 98	Address map for CSN[7]
97 - 84	Address map for CSN[6]
83 - 70	Address map for CSN[5]
69 - 56	Address map for CSN[4]

Table 2125.

DEFCFG bits	Configuration
55 - 42	Address map for CSN[3]
41 - 28	Address map for CSN[2]
27 - 14	Address map for CSN[1]
13 - 0	Address map for CSN[0]

Table 2126. Config for CSN



27: 8 Base memory address for CSN

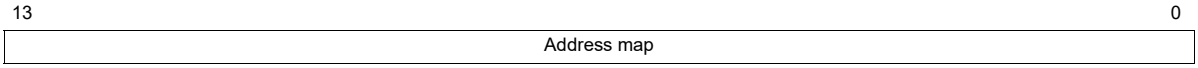
13: 12 Reserved

11: 2 Address mask

1 Reserved

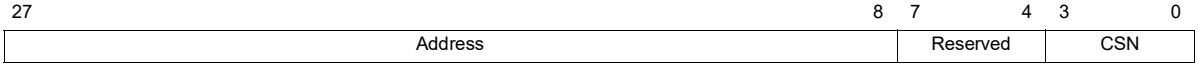
0 CSN enable

Table 2127. Address for CSN



13: 0 Accesses to this CSN are using this address map as bit[31:18] on the AHB bus

Table 2128. Reconfiguration area



27: 8 Base address for the reconfiguration area

7: 4 Reserved

3: 0 The reconfiguration area is located at this CSN

109.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x07F. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

109.5 Implementation

109.5.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

109.5.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

109.6 Configuration options

Table 2129 shows the configuration options of the core (VHDL generics).

*Table 2129.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
oepol	Output enable polarity	0 - 1	0
filter	Memory bus signal filtering/sampling	1	1
csnnum	Number of chip selects	1 - 10	1
config	Implement reconfiguration	0 - 1	0

109.7 Signal descriptions

Table 2130 shows the interface signals of the core (VHDL ports).

*Table 2130.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
SCLK	N/A	Input	Not used	-
MMAI	ADDRESS[31:0]	Input	Memory address	High
	CSN[9:0]	Input	Chip select	Low
	DATA[31:0]	Input	Memory data	-
	OEN	Input	Output enable	Low
	WRITEN	Input	Write strobe	Low
	WREN[3:0]	Input	Write enable: WREN[0] corresponds to DATA[31:24], WREN[1] corresponds to DATA[23:16], WREN[2] corresponds to DATA[15:8], WREN[3] corresponds to DATA[7:0].	Low
	READ	Input	Read strobe	High
MMAO	DATA[31:0]	Output	Memory data	High
	DATAOEN[31:0]	Output	Memory data output enable	High/Low
	BRDYN	Output	Bus ready strobe	Low
	BRDYOEN	Output	Bus ready strobe output enable	High/Low
	BEXCN	Output	Bus exception	Low
	BEXCOEN	Output	Bus exception output enable	High/Low
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
DEFCFG[407:0]		Input	Default configuration	-

* see GRLIB IP Library User's Manual

109.8 Library dependencies

Table 2131 shows libraries used when instantiating the core (VHDL libraries).

*Table 2131.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MMA_PKG	Component	Component declaration

109.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
```

```
use gaisler.misc.all;

entity mctrl_ex is
  port (
     clk : in std_ulogic;
     rstn : in std_ulogic;
    -- memory bus
     address  : in     std_logic_vector(27 downto 0);
     data     : inout std_logic_vector(31 downto 0);
     csn      : in     std_logic_vector(0 downto 0);
     rwen     : in     std_logic_vector(3 downto 0);
     oen      : in     std_logic;
     read     : in     std_logic;
     writen   : in     std_logic;
     brdyn    : out    std_logic;
     bexcn    : out    std_logic;
   --other signals
  ....
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal mmai : mma_in_type;
  signal mmao : mma_out_type;


begin

-- MMA core
mma0 : mma
generic map(
  hindex => 0,
  oepol => 0,
  filter => 1,
  csnnum => 1,
  config => 0)
port map(
  rst => rst,
  clk => clk,
  sclk => '0',
  mmai => mmai,
  mmao => mmao,
  ahbmi => ahbmi,
  ahbmo => ahbmo(0),
  defcfg => mma_cfgi);

mma_cfgi <= mma_cfg(
  csn0 => mma_csn_cfg(x"2340", x"340", x"FC0", "00", '0', '1'));

-- I/O pads driving data memory bus data signals
  datapads : for i in 0 to 3 generate
      data_pad : iopadv generic map (width => 8)
      port map (pad => data(31-i*8 downto 24-i*8),
                o => mmai.data(31-i*8 downto 24-i*8),
                en => mmao.dataoen(31-i*8 downto 24-i*8),
                i => mmao.data(31-i*8 downto 24-i*8));
  end generate;

  -- connect memory controller outputs to entity output signals
  mmai.address <= address; mmai.csn(0) <= csn(0);
  mmai.oen <= oen; mmai.wren <= rwen;
  mmai.writen <= writen; mmai.read <= read; mmai.iosn <= iosn;
  brdyn <= mmao.brdyn; bexcn <= mmao.bexcn;
end;
```

110 MUL32 - Signed/unsigned 32x32 multiplier module

110.1 Overview

The multiplier module is highly configurable module implementing 32x32 bit multiplier. Multiplier takes two signed or unsigned numbers as input and produces 64-bit result. Multiplication latency and hardware complexity depend on multiplier configuration. Variety of configuration option makes it possible to configure the multiplier to meet wide range of requirements on complexity and performance.

For DSP applications the module can be configured to perform multiply & accumulate (MAC) operation. In this configuration 16x16 multiplication is performed and the 32-bit result is added to 40-bit value accumulator.

110.2 Operation

The multiplication is started when '1' is samples on MULI.START on positive clock edge. Operands are latched externally and provided on inputs MULI.OP1 and MULI.OP2 during the whole operation. The result appears on the outputs during the clock cycle following the clock cycle when MULO.READY is asserted if multiplier if 16x16, 32x8 or 32x16 configuration is used. For 32x32 configuration result appears on the output during the second clock cycle after the MULI.START was asserted.

Signal MULI.MAC shall be asserted to start multiply & accumulate (MAC) operation. This signal is latched on positive clock edge. Multiplication is performed between two 16-bit values on inputs MULI.OP1[15:0] and MULI.OP2[15:0]. The 32-bit result of the multiplication is added to the 40-bit accumulator value on signal MULI.ACC to form a 40-bit value on output MULO.RESULT[39:0]. The result of MAC operation appears during the second clock cycle after the MULI.MAC was asserted.

110.3 Implementation

110.3.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers if the GRLIB config package setting *grlib_async_reset_enable* is set.

110.3.2 Complexity

Table 2132 shows hardware complexity in ASIC gates and latency for different multiplier configurations.

*Table 2132.*Multiplier latencies and hardware complexity

Multiplier size (multype)	Pipelined (pipe)	Latency (clocks)	Approximate area (gates)
16x16	1	5	6 500
16x16	0	4	6 000
32x8	-	4	5 000
32x16	-	2	9 000
32x32	-	1	15 000

110.4 Configuration options

Table 2133 shows the configuration options of the core (VHDL generics).

*Table 2133.*Configuration options

Generic	Function	Allowed range	Default
tech	Multiplier technolofy selection. If set to 0 the multipliers will be inferred by the synthesis tool. Use this option if your synthesis tool i capable of inferring efficient multiplier implementation.	0 to NTECH-1	0
multype	Size of the multiplier that is actually implemented. All configuration produce 64-bit result with different latencies. 0 - 16x16 bit multiplier 1 - 32x8 bit multiplier 2 - 32x16 bit multiplier 3 - 32x32 bit multiplier	0 to 3	0
pipe	Used in 16x16 bit multiplier configuration. Adds a pipeline register stage to the multiplier. This option gives better timing but adds one clock cycle to latency.	0 to 1	0
mac	Enable multiply & accumulate operation. Use only with 16x16 multiplier option with no pipelining (*pipe* = 0)	0 to 1	0
arch	Multiplier structure 0: Inferred by synthesis tool 1: Generated using Module Generators from NTNU 2: Using technology specific netlists (techspec). Only supported for RTAX-D FPGAs. Other technologies will assert a simulation error. 3: Using Synopsys DesignWare (DW02_mult and DW_mult_pipe)	0 to 3	0
scantest	Enable scan test support. Only required if GRLIB has been changed to use asynchronous reset.	0 - 1	0

110.5 Signal descriptions

Table 2134 shows the interface signals of the core (VHDL ports).

*Table 2134.*Signal declarations

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
HOLDN	N/A	Input	Hold	Low
MULI	OP1[32:0]	Input	Operand 1 OP1[32] - Sign bit. OP1[31:0] - Operand 1 in 2's complement format	High
	OP2[32:0]		Operand 2 OP2[32] - Sign bit. OP2[31:0] - Operand 2in 2's complement format	High
	FLUSH		Flush current operation	High
	SIGNED		Signed multiplication	High
	START		Start multiplication	High
	MAC		Multiply & accumulate	High
	ACC[39:0]		Accumulator. Accumulator value is held externally.	High
MULO	READY	Output	Result is ready during the next clock cycle for 16x16, 32x8 and 32x16 configurations. Not used for 32x32 configuration or MAC operation.	High
	NREADY		Not used	-
	ICC[3:0]		Condition codes ICC[3] - Negative result (not used in 32x32 conf) ICC[1] - Zero result (not used in 32x32 conf) ICC[1:0] - Not used	High
	RESULT[63:0]		Result. Available at the end of the clock cycle if MULO.READY was asserted in previous clock cycle. For 32x32 configuration the result is available during second clock cycle after the MULI.START was asserted.	High
TESTEN	N/A	Input	Test enable (only used together with async. reset)	High
TESTRST	N/A	Input	Test reset (only used together with async. reset)	Low

110.6 Library dependencies

Table 2135 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2135.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	ARITH	Signals, component	Signals, component declaration

110.7 Component declaration

The core has the following component declaration.

```
component mul32
```

```
generic (
    infer   : integer := 1;
    multype : integer := 0;
    pipe    : integer := 0;
    mac     : integer := 0
);
port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    holdn   : in  std_ulogic;
    muli    : in  mul32_in_type;
    mulo    : out mul32_out_type
);
end component;
```

110.8 Instantiation

This example shows how the core can be instantiated.

The module is configured to implement 16x16 pipelined multiplier with support for MAC operations.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use gaisler.arith.all;

.
.
.

signal muli  : mul32_in_type;
signal mulo  : mul32_out_type;

begin

mul0 : mul32 generic map (infer => 1, multype => 0, pipe => 1, mac => 1)
    port map (rst, clk, holdn, muli, mulo);


end;
```

111 MULTLIB - High-performance multipliers

111.1 Overview

The GRLIB.MULTLIB VHDL-library contains a collection of high-performance multipliers from the Arithmetic Module Generator at Norwegian University of Science and Technology. 32x32, 32x8, 32x16, 16x16 unsigned/signed multipliers are included. 16x16-bit multiplier can be configured to include a pipeline stage. This option improves timing but increases latency with one clock cycle.

111.2 Configuration options

Table 2136 shows the configuration options of the core (VHDL generics).

*Table 2136.*Configuration options

Generic	Function	Allowed range	Default
mulpipe	Include a pipeline stage (0 -pipelining disabled, 1 - pipelining enabled)	0 - 1	0

111.3 Signal descriptions

Table 2137 shows the interface signals of the core (VHDL ports).

*Table 2137.*Signal descriptions

Signal name	Type	Function	Active
CLK (16x16 multiplier only)	Input	Clock	-
HOLDN (16x16 multiplier only)	Input	Hold. When active, the pipeline register is not updates	Low
X[16:0] (16x16 mult) X[32:0] (32x8 mult) X[32:0] (32x16 mult) X[32:0] (32x32 mult)	Input	Operand 1. MBS bit is sign bit.	High
Y[16:0] (16x16 mult) Y[8:0] (32x8 mult) Y[16:0] (32x16 mult) Y[32:0] (32x32 mult)	Input	Operand 2. MSB bit is sign bit.	High
P[33:0] (16x16 mult) P[41:0] (32x8 mult) P[49:0] (32x16 mult) P[65:0] (32x32 mult)		Result. Two MSB bits are sign bits.	High

111.4 Library dependencies

Table 2138 shows libraries used when instantiating the core (VHDL libraries).

*Table 2138.*Library dependencies

Library	Package	Imported unit	Description
GRLIB	MULTLIB	Component	Multiplier component declarations

111.5 Component declaration

The core has the following component declaration.

```
component mul_33_33
  port (
    x    : in  std_logic_vector(32 downto 0);
    y    : in  std_logic_vector(32 downto 0);
    p    : out std_logic_vector(65 downto 0)
  );
end component;
```

111.6 Implementation

111.6.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

111.7 Instantiation

This example shows how the core can be instantiated.

The core is configured to implement 16x16 pipelined multiplier with support for MAC operations.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.multlib.all;

.
.

signal op1, op2 : std_logic_vector(32 downto 0);
signal prod : std_logic_vector(65 downto 0);

begin

m0 : mul_33_33
         port map (op1, op2, prod);

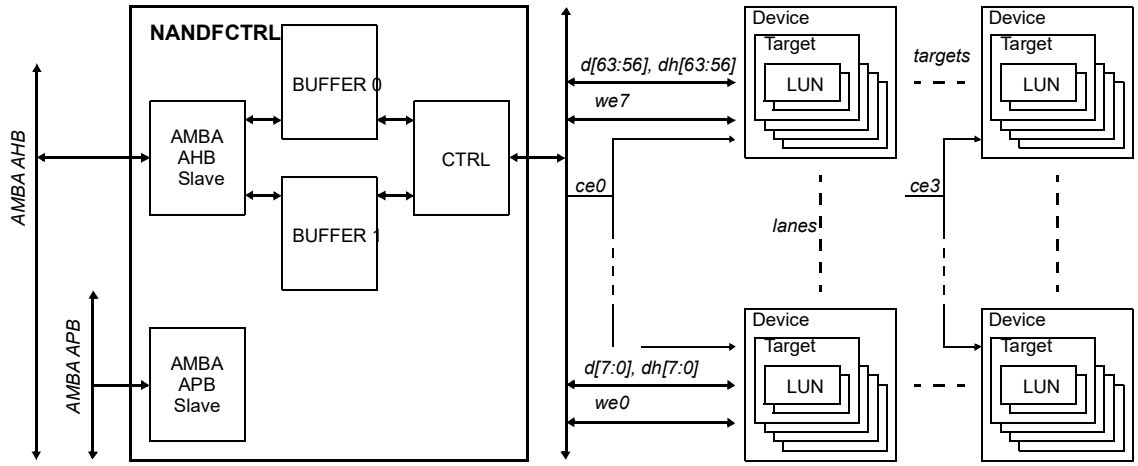
end;
```

112 NANDFCTRL - NAND Flash Memory Controller

112.1 Overview

The NAND Flash Memory Controller (NANDFCTRL) core provides a bridge between external NAND flash memory and the AMBA bus. The memory controller is an Open NAND Flash Interface (ONFI) 2.2 command compliant core (see exceptions below) and it can communicate with multiple parallel flash memory devices simultaneously, where each device in turn can consist of up to four individually addressable targets, one target addressed at a time. The core is configured through a set of AMBA APB registers, described in section 112.4, and data is written to / read from the flash memory by accessing internal buffers mapped over AMBA AHB.

This document mainly describes the NANDFCTRL core's functionality. For details about the actual flash memory interface, flash memory architecture and ONFI 2.2 command set please refer to the *Open NAND Flash Interface specification, revision 2.2*, hereafter called the ONFI 2.2 specification.



Note: One device might have more than one target, using one chip enable signal for each target. This will reduce the number devices that can be placed horizontally in the figure. All devices (and the internal targets) placed vertically in the figure belong to the same chip enable signal, with individual write enable signals controlling each 8-bit/16-bit data lane.

Figure 298. Block diagram

112.2 Operation

112.2.1 System overview

A block diagram of the core can be seen in figure 298. Features and limitations of the core are listed below:

- All commands defined in the ONFI 2.2 standard are supported, except Synchronous Reset and Interleaved Read.
- The core does not implement support for the source synchronous data interface, only asynchronous data interface.
- The core does not place any other limitation on the device architecture other than those specified in the ONFI 2.2 standard. For example, the core does not need to know how many LUNs, blocks, or pages a connected flash memory device has. (See the ONFI 2.2 specification for information about LUNs, blocks, and pages.)
- Multiple parallel data lanes are supported, which gives the possibility to read / write several flash memory devices at the same time.

- To support interleaving of flash memory accesses and AMBA accesses and give greater throughput, two buffers for reading / writing flash memory data are implemented.
- The data interface timing can either be fixed (set at implementation time) or programmable through AMBA APB registers. With fixed data interface timing support for ONFI timing modes 0 - 5 can be implemented and then switched between during run-time. Programmable timing interface allows for clock frequencies unknown at implementation time, as well as custom timing. It is highly recommended to use programmable timing as the additional delays (like I/O timings) might require custom timing options. Read section 112.5 (Timing Modes) for more information.
- The core does not implement any wear-leveling or bad block management.
- The core does not implement any direct access to flash memory devices from the AMBA AHB bus. Instead accesses are performed through two temporary buffers by initiating command and data transfers through control registers via the AMBA APB bus.
- The temporary buffers are mapped into AMBA AHB memory space. Note that the buffer addresses are not directly mapped to the flash memory, but are mapped specific addressing registers. This allows large amount of flash memory to be address, exceeding the 4 Gbyte address space of the AMBA bus.

112.2.2 Internal buffer structure

The number of buffers implemented in the core depends on whether or not the core is implemented with separate buffers for each parallel 8-bit/16-bit data lane or not. This is indicated by the *sepb* field in the *Capability register*. How many data lanes the core implements can be found by reading the *nlane* field of the *Capability register* and adding one. If separate buffers are used then for each data lane there are four different buffers implemented. Two buffers are used for data that are read from / written to any page area in the flash memory device (hereafter called *page buffer*), and the other two are used for data read from / written to any page's spare area (hereafter called *spare buffer*). The size of each page buffer (in bytes) is $2^{\wedge(pbits+1)}$, where pbits is the value of the *pbits* field in the *Capability register*. The size of each spare buffer (in bytes) is $2^{\wedge(sbits+1)}$, where sbits is the value of the *sbits* field in the *Capability register*. If separate buffers aren't used, then the core implements one set of the above mentioned four buffers and uses them for all data lanes.

One page buffer and one spare buffer for each data lane (or all lanes if separate buffers aren't used) are grouped together into what in this document is called *buffer 0*. The other set of page buffers and spare buffers are grouped into *buffer 1*. For example, if the core has support for eight data lanes with separate buffers, and the page buffers are 4096 bytes, and the spare buffers are 256 bytes, then buffer 0 and buffer 1 will each be 32768 + 2048 bytes large. Buffer 0 and buffer 1 are associated with their own set of control registers, described in section 112.4.

Note that support for buffer 1 is optional. Whether or not support for buffer 1 is implemented is indicated by the *b1en* bit in the *Capability register*.

All buffers are mapped into AMBA AHB address space, and the core supports two different mapping schemes, with their own designated AMBA AHB address space:

- For the first address map, called *Consecutive address map*, the first part of the assigned AHB memory area is mapped to the page buffer corresponding to the first 8-bit/16-bit data lane, followed by the page buffer corresponding to the second 8-bit/16-bit data lane etc. After all the page buffers the spare buffers follow in the same manner. See table 2139 for an example with 4096 bytes page buffer and 256 bytes spare buffer and table 2140 for an example with 2048 bytes page buffer and 128 bytes spare buffer.
- For the second address map, called *By page address map*, the first part of the assigned AHB memory area is mapped to the page buffer corresponding to the first 8-bit/16-bit data lane, followed by the spare buffer corresponding to the same data lane. After that follows the page buffer and spare buffer pairs for all other data lanes. Note that since the spare buffers normally are much

smaller than the page buffers there is normally a gap between the last address of a spare buffer and the first address of the next page buffer. The size of the gap is page buffer size - spare buffer size. See table 2141 for an example with 4096 bytes page buffer and 256 bytes spare buffer and see table 2142 for an example with 2048 bytes page buffer and 128 bytes spare buffer.

- See table 2143 for an example which illustrates both *Consecutive address map* and *page address map* for 8192 bytes page buffer and 2048 bytes spare buffer with two lanes, separate buffers and buffer 1 is not implemented.

Note that the page and spare buffer areas might be larger than the size of the page and spare memory implemented in the actual flash memory device. Thus, there might be gaps in the addressing schemes. If the EDAC is implemented (indicated by the *edac* bit in the *Capability register*) then the buffers have an additional AHB address map. Read and write accesses to the additional AHB address space will trigger EDAC operation. This additional address map is also shown in table 2139 and table 2140.

Table 2139. Buffer memory map, consecutive addressing. Example with 4096 B page size, 256 B spare size, separate buffers, both buffer 0 and buffer 1 implemented, and with EDAC.

AMBA AHB offset	address	Contents	AMBA AHB offset	address	Contents
Normal address space (no EDAC operation)			**Normal address space (no EDAC operation)**		
0x00000	0x00FFF	Buffer 0, 4096 byte page data, lane 0	0x10000	0x10FFF	Buffer 1, 4096 byte page data, lane 0
0x01000	0x01FFF	Buffer 0, 4096 byte page data, lane 1	0x11000	0x11FFF	Buffer 1, 4096 byte page data, lane 1
0x02000	0x02FFF	Buffer 0, 4096 byte page data, lane 2	0x12000	0x12FFF	Buffer 1, 4096 byte page data, lane 2
0x03000	0x03FFF	Buffer 0, 4096 byte page data, lane 3	0x13000	0x13FFF	Buffer 1, 4096 byte page data, lane 3
0x04000	0x04FFF	Buffer 0, 4096 byte page data, lane 4	0x14000	0x14FFF	Buffer 1, 4096 byte page data, lane 4
0x05000	0x05FFF	Buffer 0, 4096 byte page data, lane 5	0x15000	0x15FFF	Buffer 1, 4096 byte page data, lane 5
0x06000	0x06FFF	Buffer 0, 4096 byte page data, lane 6	0x16000	0x16FFF	Buffer 1, 4096 byte page data, lane 6
0x07000	0x07FFF	Buffer 0, 4096 byte page data, lane 7	0x17000	0x17FFF	Buffer 1, 4096 byte page data, lane 7
0x08000	0x080FF	Buffer 0, 256 byte spare data, lane 0	0x18000	0x180FF	Buffer 1, 256 byte spare data, lane 0
0x08100	0x081FF	Buffer 0, 256 byte spare data, lane 1	0x18100	0x181FF	Buffer 1, 256 byte spare data, lane 1
0x08200	0x082FF	Buffer 0, 256 byte spare data, lane 2	0x18200	0x182FF	Buffer 1, 256 byte spare data, lane 2
0x08300	0x083FF	Buffer 0, 256 byte spare data, lane 3	0x18300	0x183FF	Buffer 1, 256 byte spare data, lane 3
0x08400	0x084FF	Buffer 0, 256 byte spare data, lane 4	0x18400	0x184FF	Buffer 1, 256 byte spare data, lane 4
0x08500	0x085FF	Buffer 0, 256 byte spare data, lane 5	0x18500	0x185FF	Buffer 1, 256 byte spare data, lane 5
0x08600	0x086FF	Buffer 0, 256 byte spare data, lane 6	0x18600	0x186FF	Buffer 1, 256 byte spare data, lane 6
0x08700	0x087FF	Buffer 0, 256 byte spare data, lane 7	0x18700	0x187FF	Buffer 1, 256 byte spare data, lane 7
EDAC address space (EDAC operation on page buffers)			**EDAC address space (EDAC operation on page buffers)**		
0x20000	0x20FFF	Buffer 0, 4096 byte page data, lane 0	0x30000	0x30FFF	Buffer 1, 4096 byte page data, lane 0
0x21000	0x21FFF	Buffer 0, 4096 byte page data, lane 1	0x31000	0x31FFF	Buffer 1, 4096 byte page data, lane 1
0x22000	0x22FFF	Buffer 0, 4096 byte page data, lane 2	0x32000	0x32FFF	Buffer 1, 4096 byte page data, lane 2
0x23000	0x23FFF	Buffer 0, 4096 byte page data, lane 3	0x33000	0x33FFF	Buffer 1, 4096 byte page data, lane 3
0x24000	0x24FFF	Buffer 0, 4096 byte page data, lane 4	0x34000	0x34FFF	Buffer 1, 4096 byte page data, lane 4
0x25000	0x25FFF	Buffer 0, 4096 byte page data, lane 5	0x35000	0x35FFF	Buffer 1, 4096 byte page data, lane 5
0x26000	0x26FFF	Buffer 0, 4096 byte page data, lane 6	0x36000	0x36FFF	Buffer 1, 4096 byte page data, lane 6
0x27000	0x27FFF	Buffer 0, 4096 byte page data, lane 7	0x37000	0x37FFF	Buffer 1, 4096 byte page data, lane 7
0x28000	0x280FF	Buffer 0, 256 byte spare data, lane 0	0x38000	0x380FF	Buffer 1, 256 byte spare data, lane 0
0x28100	0x281FF	Buffer 0, 256 byte spare data, lane 1	0x38100	0x381FF	Buffer 1, 256 byte spare data, lane 1
0x28200	0x282FF	Buffer 0, 256 byte spare data, lane 2	0x38200	0x382FF	Buffer 1, 256 byte spare data, lane 2
0x28300	0x283FF	Buffer 0, 256 byte spare data, lane 3	0x38300	0x383FF	Buffer 1, 256 byte spare data, lane 3
0x28400	0x284FF	Buffer 0, 256 byte spare data, lane 4	0x38400	0x384FF	Buffer 1, 256 byte spare data, lane 4
0x28500	0x285FF	Buffer 0, 256 byte spare data, lane 5	0x38500	0x385FF	Buffer 1, 256 byte spare data, lane 5
0x28600	0x286FF	Buffer 0, 256 byte spare data, lane 6	0x38600	0x386FF	Buffer 1, 256 byte spare data, lane 6
0x28700	0x287FF	Buffer 0, 256 byte spare data, lane 7	0x38700	0x387FF	Buffer 1, 256 byte spare data, lane 7

Table 2140. Buffer memory map, consecutive addressing. Example with 2048 B page size, 128 B spare size, separate buffers, both buffer 0 and buffer 1 implemented, using sing lane, and with EDAC.

AMBA AHB offset	address	Contents	AMBA AHB offset	address	Contents
Normal address space (no EDAC operation)			**Normal address space (no EDAC operation)**		
0x0000	0x007FF	Buffer 0, 2048 byte page data, lane 0	0x1000	0x17FF	Buffer 1, 2048 byte page data, lane 0
0x0800	0x0087F	Buffer 0, 128 byte spare data, lane 0	0x1800	0x187F	Buffer 1, 128 byte spare data, lane 0
EDAC address space (EDAC operation on page buffer)			**EDAC address space (EDAC operation on page buffer)**		
0x2000	0x27FF	Buffer 0, 2048 byte page data, lane 0	0x3000	0x37FF	Buffer 1, 2048 byte page data, lane 0
0x2800	0x287F	Buffer 0, 128 byte spare data, lane 0	0x3800	0x387F	Buffer 1, 128 byte spare data, lane 0

Table 2141. Buffer memory map, by page addressing. Example with 4096 B page size, 256 B spare size, separate buffers, both buffer 0 and buffer 1 implemented, and with EDAC.

AMBA AHB offset	address	Contents	AMBA AHB offset	address	Contents
Normal address space (no EDAC operation)			**Normal address space (no EDAC operation)**		
0x00000	0x00FFF	Buffer 0, 4096 byte page data, lane 0	0x10000	0x10FFF	Buffer 1, 4096 byte page data, lane 0
0x01000	0x010FF	Buffer 0, 256 byte spare data, lane 0	0x11000	0x110FF	Buffer 1, 256 byte spare data, lane 0
0x02000	0x02FFF	Buffer 0, 4096 byte page data, lane 1	0x12000	0x12FFF	Buffer 1, 4096 byte page data, lane 1
0x03000	0x030FF	Buffer 0, 256 byte spare data, lane 1	0x13000	0x130FF	Buffer 1, 256 byte spare data, lane 1
0x04000	0x04FFF	Buffer 0, 4096 byte page data, lane 2	0x14000	0x14FFF	Buffer 1, 4096 byte page data, lane 2
0x05000	0x050FF	Buffer 0, 256 byte spare data, lane 2	0x15000	0x150FF	Buffer 1, 256 byte spare data, lane 2
0x06000	0x06FFF	Buffer 0, 4096 byte page data, lane 3	0x16000	0x16FFF	Buffer 1, 4096 byte page data, lane 3
0x07000	0x070FF	Buffer 0, 256 byte spare data, lane 3	0x17000	0x170FF	Buffer 1, 256 byte spare data, lane 3
0x08000	0x08FFF	Buffer 0, 4096 byte page data, lane 4	0x18000	0x18FFF	Buffer 1, 4096 byte page data, lane 4
0x09000	0x090FF	Buffer 0, 256 byte spare data, lane 4	0x19000	0x190FF	Buffer 1, 256 byte spare data, lane 4
0x0A000	0x0AFFF	Buffer 0, 4096 byte page data, lane 5	0x1A000	0x1AFFF	Buffer 1, 4096 byte page data, lane 5
0x0B000	0x0B0FF	Buffer 0, 256 byte spare data, lane 5	0x1B000	0x1B0FF	Buffer 1, 256 byte spare data, lane 5
0x0C000	0x0CFFF	Buffer 0, 4096 byte page data, lane 6	0x1C000	0x1CFFF	Buffer 1, 4096 byte page data, lane 6
0x0D000	0x0D0FF	Buffer 0, 256 byte spare data, lane 6	0x1D000	0x1D0FF	Buffer 1, 256 byte spare data, lane 6
0x0E000	0x0EFFF	Buffer 0, 4096 byte page data, lane 7	0x1E000	0x1EFFF	Buffer 1, 4096 byte page data, lane 7
0x0F000	0x0F0FF	Buffer 0, 256 byte spare data, lane 7	0x1F000	0x1F0FF	Buffer 1, 256 byte spare data, lane 7
EDAC address space (EDAC operation on page buffer)			**EDAC address space (EDAC operation on page buffer)**		
0x20000	0x20FFF	Buffer 0, 4096 byte page data, lane 0	0x30000	0x30FFF	Buffer 1, 4096 byte page data, lane 0
0x21000	0x210FF	Buffer 0, 256 byte spare data, lane 0	0x31000	0x310FF	Buffer 1, 256 byte spare data, lane 0
0x22000	0x22FFF	Buffer 0, 4096 byte page data, lane 1	0x32000	0x32FFF	Buffer 1, 4096 byte page data, lane 1
0x23000	0x230FF	Buffer 0, 256 byte spare data, lane 1	0x33000	0x330FF	Buffer 1, 256 byte spare data, lane 1
0x24000	0x24FFF	Buffer 0, 4096 byte page data, lane 2	0x34000	0x34FFF	Buffer 1, 4096 byte page data, lane 2
0x25000	0x250FF	Buffer 0, 256 byte spare data, lane 2	0x35000	0x350FF	Buffer 1, 256 byte spare data, lane 2
0x26000	0x26FFF	Buffer 0, 4096 byte page data, lane 3	0x36000	0x36FFF	Buffer 1, 4096 byte page data, lane 3
0x27000	0x270FF	Buffer 0, 256 byte spare data, lane 3	0x37000	0x370FF	Buffer 1, 256 byte spare data, lane 3
0x28000	0x28FFF	Buffer 0, 4096 byte page data, lane 4	0x38000	0x38FFF	Buffer 1, 4096 byte page data, lane 4
0x29000	0x290FF	Buffer 0, 256 byte spare data, lane 4	0x39000	0x390FF	Buffer 1, 256 byte spare data, lane 4
0x2A000	0x2AFFF	Buffer 0, 4096 byte page data, lane 5	0x3A000	0x3AFFF	Buffer 1, 4096 byte page data, lane 5
0x2B000	0x2B0FF	Buffer 0, 256 byte spare data, lane 5	0x3B000	0x3B0FF	Buffer 1, 256 byte spare data, lane 5
0x2C000	0x2CFFF	Buffer 0, 4096 byte page data, lane 6	0x3C000	0x3CFFF	Buffer 1, 4096 byte page data, lane 6
0x2D000	0x2D0FF	Buffer 0, 256 byte spare data, lane 6	0x3D000	0x3D0FF	Buffer 1, 256 byte spare data, lane 6
0x2E000	0x2EFFF	Buffer 0, 4096 byte page data, lane 7	0x3E000	0x3EFFF	Buffer 1, 4096 byte page data, lane 7
0x2F000	0x2F0FF	Buffer 0, 256 byte spare data, lane 7	0x3F000	0x3F0FF	Buffer 1, 256 byte spare data, lane 7

Table 2142. Buffer memory map, by page addressing. Example with 2048 B page size, 128 B spare size, separate buffers, both buffer 0 and buffer 1 implemented, using sing lane, and with EDAC.

AMBA AHB offset	address	Contents
Normal address space (no EDAC operation)		
0x0000	0x007FF	Buffer 0, 2048 byte page data, lane 0
0x0800	0x0087F	Buffer 0, 128 byte spare data, lane 0
EDAC address space (EDAC operation on page buffer)		
0x2000	0x27FF	Buffer 0, 2048 byte page data, lane 0
0x2800	0x287F	Buffer 0, 128 byte spare data, lane 0

AMBA AHB offset	address	Contents
Normal address space (no EDAC operation)		
0x1000	0x17FF	Buffer 1, 2048 byte page data, lane 0
0x1800	0x187F	Buffer 1, 128 byte spare data, lane 0
EDAC address space (EDAC operation on page buffer)		
0x3000	0x37FF	Buffer 1, 2048 byte page data, lane 0
0x3800	0x387F	Buffer 1, 128 byte spare data, lane 0

Table 2143. Buffer memory map, by consecutive addressing and by page addressing. Example with 8192B page size, 2048 B spare size, separate buffers, only buffer 0 implemented, and with EDAC.

CONSECUTIVE ADDRESSING

AMBA AHB offset	address	Contents
Normal address space (no EDAC operation)		
0x00000	0x01FFF	Buffer 0, 8192 byte page data, lane 0
0x02000	0x03FFF	Buffer 0, 8192 byte page data, lane 1
0x04000	0x047FF	Buffer 0, 2048 byte spare data, lane 0
0x04800	0x04FFF	Buffer 0, 2048 byte spare data, lane 1
EDAC address space (EDAC operation on page buffer)		
0x10000	0x11FFF	Buffer 0, 8192 byte page data, lane 0
0x12000	0x13FFF	Buffer 0, 8192 byte page data, lane 1
0x14000	0x147FF	Buffer 0, 2048 byte spare data, lane 0
0x14800	0x14FFF	Buffer 0, 2048 byte spare data, lane 1

PAGE ADDRESSING

AMBA AHB offset	address	Contents
Normal address space (no EDAC operation)		
0x00000	0x01FFF	Buffer 0, 8192 byte page data, lane 0
0x02000	0x027FF	Buffer 0, 2048 byte spare data, lane 0
0x04000	0x05FFF	Buffer 0, 8192 byte page data, lane 1
0x06000	0x067FF	Buffer 0, 2048 byte spare data, lane 1
EDAC address space (EDAC operation on page buffer)		
0x10000	0x11FFF	Buffer 0, 8192 byte page data, lane 0
0x12000	0x127FF	Buffer 0, 2048 byte spare data, lane 0
0x14000	0x15FFF	Buffer 0, 8192 byte page data, lane 1
0x16000	0x167FF	Buffer 0, 2048 byte spare data, lane 1

112.2.3 Data interface timing

The ONFI timing parameters that the core explicitly handles are:

- tCCS - Change Column setup time
- tADL - ALE to data loading time
- tCS - CE setup time
- tRP - RE pulse width
- tREH - RE high hold time (starting from REV. 2)
- tRR - Ready to RE low (data only)
- tWP - WE pulse width
- tWB - WE high to SR[6] low
- tRHW - RE high to we WE low
- tWH - WE high hold time
- tWHR - WE high to RE low
- tWW - WP transition to WE low

All other timing requirements are either fulfilled through design, or are handled by the flash memory devices. See the ONFI specification for details about the different timing parameters.

The data interface timing can be either fixed (set at implementation time) or programmable to allow both system clock frequencies unknown at implementation time as well as custom timing parameters.

When the timing is fixed all timing parameters are calculated on implementation time based on the specified system clock frequency. When programmable timing is used the timing parameters are programmed through AMBA APB registers, described in section 112.4. The registers power-up / reset values are set at implementation time.

Note that the timing parameter tCCS is always programmable, even if fixed timing is used for all other parameters. This is because the ONFI specification says that after initialization is complete, the value for tCCS specified in the flash memory's parameter page should be used.

112.2.4 Accessing the NAND flash memory devices

The steps that need to be taken to access (i.e. send an ONFI 2.2 command to) the flash memory devices are:

1. Make sure that the chosen buffer is not busy by checking the *run* and *bsy* bits in the *Buffer control / status* register. If both the *run* bit and *bsy* bits are set then the core is currently executing a command associated with that buffer, and the buffer can not be used. If only the *bsy* bit is set then the core is done executing a command but the corresponding data buffer and control bits are still write protected. Software then needs to clear the *bsy* bit by writing '1' to it.

2. If the command requires a row address to be sent, write it to the *Buffer row address register*. Otherwise this step can be skipped.

3. If the command requires a column address to be sent, write it to the *coladdr* field of the *Buffer column address register*. Bits 7:0 of the *coladdr* field also need to be written with the one byte address used for SET FEATURES, GET FEATURES, READ ID, READ UNIQUE ID, and READ PARAMETER PAGE commands. Otherwise this step can be skipped.

4. Write the command value to the *Buffer command register*, and possibly set the control bits *sel*, *cd*, or *sc2* if needed. See table 2151 in section 112.4 for a description of these bits.

5. If the command should include a data phase then set the size of the data by writing to the *size* field of the *Buffer column address register*. This step is not necessary for the SET FEATURES, GET FEATURES, READ STATUS, and READ STATUS ENHANCED commands since they always have a fixed size data phase.

6. If data should be written to the flash memory devices then write this data to the corresponding buffers. Note that the core uses the *coladdr* field in the *Buffer column address register* to index into the buffers, which means that data that should be written with an offset into a flash memory page (i.e. column address is not zero) need to be written with the same offset into the buffers. The exception is the commands mentioned in step 3. They always read from the beginning of the buffers. This step can be skipped if no data are to be written.

7. Select which data lanes and targets the command should be sent to, if an interrupt should be generated when the command is finished, and start execution by writing to the *lanesel*, *targsel*, *irqmsk*, and *exe* bits in the *Buffer control / status register*. See table 2152 in section 112.4 for a description of these bits.

Once command execution has been started software can monitor the *run* bit in the *Buffer control / status register* (or wait for an interrupt if the core was configured to generate one) to learn when the command is finished. If data was read from the flash memory then it can be found in the buffers. Note that the core uses the *coladdr* field in the *Buffer column address register* to index into the buffers, which means that data that was read from a flash memory page with an offset (i.e. column address is not zero) was written into the buffers with the same offset. The exception are the commands mentioned in step 3. They always place their data in the beginning of the buffers.

112.2.5 Endianness

The core supports both little-endian and big-endian systems. The endianness can be configured using the settings in the GRLIB configuration package (see GRLIB User's Manual). Maximum 32-bit wide

accesses are supported when configured as little-endian. Only 32-bit wide accesses are supported when configured as big-endian.

112.3 EDAC

112.3.1 EDAC operation

The optional NAND Flash Memory controller EDAC automatically encode and decode data being stored in the NAND Flash memory with an error correcting code. The *edac* bit in the *Capability* register shows if the EDAC is implemented or not. If implemented, the EDAC is enabled by setting the *edacen* bit in the *Core control* register. Optional AHB error responses can be generated upon an uncorrectable error by setting the *ahberren* bit in the same register. See section 112.4 for more information.

The checksum of the code is calculated and stored for each word that is written, via the AMBA interface, into the page data part of the buffer, and automatically stored in parallel in the spare data part of the buffer. For each word of data written, one byte is stored. Note that data must be written into the EDAC part of the AHB address space. Data written to the non EDAC part of the address space does not trigger EDAC operation. See section 112.2.2 for information about the buffer memory map. Halfword and byte writes are not supported when the EDAC is enabled.

Since the number of spare bytes normally is less than what is sufficient to protect the full page data, the page data size is limited by the programmable *lpaddr* field in the *Core control* register. The *lpaddr* field defines how much user data (i.e. page data) is to be used, with the remaining part of the page data and spare data being used for checksums. Note that the spare data buffer portion is therefore made bigger than the actual spare data in the NAND Flash memory. When the NAND Flash memory is written, the data will be fetched first from the page data buffer up to *lpaddr*, and then the rest will be fetched from the spare data buffer. Errors can be detected in the underlying buffer memory when reading them as part of the write operation to the NAND Flash memory.

The reverse applies to when data is fetched from the NAND Flash memory during read. When the buffer contents are read out, via the AMBA interface, each read page data word is automatically corrected using the corresponding spare buffer checksum byte. Two levels of errors can occur, errors stemming from the underlying buffer memory protection, or errors stemming from the EDAC protecting the NAND Flash memory contents. All error flags are described in the *Core status* register.

It is possible to write or read additional dummy words to the page data buffer past the LPADDR address. This can be used to handle any surplus spare data bytes.

The memory contents are protected by means of a Bose Chaudhuri Hocquenghem (BCH) type of code. It is a Quad Error Correction/Quad Error Detection (QEC/QED) code.

The data symbols are 4-bit wide, represented as GF(2^4). The has the capability to detect and correct a single symbol error anywhere in the codeword.

112.3.2 Code

The code has the following definition:

- there are 4 bits per symbol; most significant has index 3 (to the left), least significant has index 0
- there are 17 symbols per codeword, of which 2 symbols represent the checksum;
- the code is systematic;
- the code can correct one symbol error per codeword;
- the field polynomial is

$$f(x) = x^4 + x + 1$$

- all multiplications are performed as Galois Field multiplications over the above field polynomial

- all additions/subtractions are performed as Galois Field additions (i.e. bitwise exclusive-or)

Note that only 8 of the 17 symbols are used for data, 2 symbols are used for the checksum, and the reset are not used, the code is thus shortened by 7 symbols.

112.3.3 Encoding

- a codeword is defined as 17 symbols:

  $[c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15}, c_{16}]$

  where c_0 to c_{14} represent information symbols and c_{15} to c_{16} represent check symbols.
- c_{15} is calculated as follows

$$c_{15} = \sum_{0}^{14} (k_i \times c_i)$$

- c_{16} is calculated as follows

$$c_{16} = \sum_{0}^{14} c_i$$

- where the constant vector k is defined as:

  k_0=0xF, k_1=0xE, .., k_{14}=0x1 (one can assume k_{15}=0x1 and k_{16}=0x1 for correction purposes)
- bit 1 of c_{15}, and bits 3 and 1 of c_{16} are inverted after encoded

112.3.4 Decoding

- the corrupt codeword is defined as 17 symbols:

  $[r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}]$
- the corrupt codeword can also be defined as 17 uncorrupt symbols and an error:

  $[c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15}, c_{16}] + [e_x]$

  where the error is defined as e_x, e being the unknown magnitude and
  x being the unknown index position in the codeword
- recalculated checksum rc_0 is calculated as follows (k_i is as defined above, x being the unknown index)
- recalculated rc_1 is calculated as follows
- syndrome s_0 is calculated as follows
- syndrome s_1 is calculated as follows, which gives the magnitude (not applicable to c_{15} and c_{16})

$$rc_0 = \sum_{0}^{14}(k_i \times r_i) = \sum_{0}^{14}(k_i \times c_i) + (k_x \times e_x)$$

$$rc_1 = \sum_{0}^{14} r_i = \sum_{0}^{14} c_i + e_x$$

$$s_0 = rc_0 + r_{15} = \sum_{0}^{14}(k_i \times r_i) + \sum_{0}^{14}(k_i \times c_i) = k_x \times e_x$$

$$s_1 = rc_1 + r_{16} = \sum_{0}^{14} r_i + \sum_{0}^{14} c_i = e_x$$

- in case s_0 and s_1 are both non-zero, to located the error in range c_0 to c_{14}, multiply error magnitude e_x with each element of the constant vector defined above:

$$k_i e_x = k_i \times s_1 = k_i \times e_x \qquad i = [0,14]$$

- search the resulting vector to find the element matching syndrome s_0, the resulting index i points to the error location (applicable only to i in [0, 14])

$$k_i e_x \Leftrightarrow k_i \times e_x$$

- finally perform the correction (applicable only to i in [0, 14])

$$c_i = r_i - s_1 = r_i - e_x = (c_i - e_x) \times e_x = c_i - e_x + e_x = c_i$$

- when s_0 is zero and s_1 is non-zero, the error is located in checksum r_{15}, no correction is necessary
- when s_1 is zero and s_0 is non-zero, the error is located in checksum r_{16}, no correction is necessary
- when s_0 and s_1 are both zero, no error has been detected, no correction is necessary
- bit 1 of r_{15}, and bits 3 and 1 of r_{16} are inverted before decoded

112.3.5 EDAC Handling

EDAC handling on NANDFCTRL is based on hardware/software co-design. There is no auto-write or auto-read of EDAC bytes. The software has to ensure that EDAC bytes are writen or read from the NAND device.

For example lets assume lpaddr is set to 1023 meaning the data that is going to be protected is 1024 bytes and total EDAC size is 256 bytes. In this case, if the full protected page data is going to be written or read the size of operation should be set to 1024+256=1280 bytes. The lpaddr will ensure that edac bits are written to or read from spare buffer. If the operation is done on a size which is smaller than the page then the EDAC has to be read/write with the correct offset with a second operation. For example if bytes 4-8 needs to be written, after writing 4 bytes from column address 4 another operation needs to be done for 1 byte from column address 1024+(4/4)=1025 this will ensure that the byte which protects the word between byte 4-8 is written to NAND device. Same applies for read operation. The second EDAC operation can be done simply with the command change column.

112.4 Registers

The core is programmed through registers mapped into APB address space. Vendor and device identifier

*Table 2144.*NANDFCTRL registers

APB address offset	Register
0x00	Core control register
0x04	Core status register
0x08	Interrupt pending register
0x0C	Capability register
0x10	Buffer 0 row address register
0x14	Buffer 0 column address register
0x18	Buffer 0 command register
0x1C	Buffer 0 control / status register
0x20*	Buffer 1 row address register
0x24*	Buffer 1 column address register
0x28*	Buffer 1 command register
0x2C*	Buffer 1 control / status register
0x30	Programmable timing register 0
0x34**	Programmable timing register 1
0x38**	Programmable timing register 2
0x3C**+	Programmable timing register 3
0x40++	Extended capability register
0x44++	Extended Buffer 0 target select register
0x48++	Extended Buffer 1 target select register

* Only present if buffer 1 registers are implemented. Indicated by *b1en* bit in *Capability register*.

** Only present if programmable timing is implemented. Indicated by *prgt* bit in *Capability register*.

+ Present starting from the revision 2 of the core.

++Present starting from the revision 3 of the core.

Note: The Buffer 0 and Buffer 1 registers are identical, and therefore only one set of tables describing the registers are presented below.

Table 2145. 0x00 - CTRL - Core control register

31	16
LPADDR	
0xFFFF	
rw	

15 13	12	11	10	9	8	7	6	5	4 3 2	1	0
R	DW	CMDO	R	AHBERREN	EDACEN	R	EDO	ABORT	TMODE	WP	RST
0	0	*	0	0	0	0	0	0	0	1	0
r	rw*	rw	r	rw	rw*	r	rw*	rw*	rw*	rw	rw

31:16 Last page address (LPADDR) - This field should be set to the last addressable byte in a flash memory page, not including the spare area. The core uses these bits to know when to switch to the internal buffers for the page's spare area. For example: If the flash memory devices have a page size of 4096 bytes (and an arbitrary sized spare area for each page) this field should be set to 0xFFF (= 4095). The actual number of bits used for this field depends on the size of the implemented buffers. The number of bits can be found by reading the *pbits* field of the *Capability register* and adding one. Reset value: 0xF..F

15:13 Reserved (R) - Always reads zero.

12 Data width (DW) - Sets the default data lane width. 0 = Core uses 8-bit data lanes. 1 = Core uses 16-bit data lanes. This can be overridden for individual commands by setting the *dwo* bit in the *Buffer command* register. This bit is only available if the *dw16* bit in the *Capability register* is 1. Reset value 0.

11 Command bit order (CMDO) - When this bit is set to 0 the ONFI command bytes are mapped to the core's data lane(s) as follows: Cmd bit 0 -> Data lane bit 7, Cmd bit 1 -> Data lane bit 6., and so on.

When this bit is set to 1 the commands are mapped as follows: Cmd bit 0 -> Data lane bit 0, Cmd bit 1 -> Data lane bit 1, and so on. Reset value equals the value of the *cmdo* bit in the *Capability register*.

10 Reserved (R) - Always reads zero.

9 AHB error response generation (AHBERREN) - If this bit is set then the core will generate an AMBA error response if the EDAC detects an uncorrectable error, or if the fault tolerance logic for the internal buffers report and uncorrectable error, upon and AHB read. If this bit is not set when an uncorrectable error is detected, the core's output signal *error* is set high for one clock cycle instead. This bit is only present when the EDAC is implemented or when the internal buffers are implemented with either byte parity and DMR or only byte parity. The *ft* bits in the *Capability register* shows which fault tolerance that is implemented, and the *edac* bit in the *Capability register* shows if EDAC is implemented. Reset value 0.

8 EDAC enable (EDACEN) - When this bit is set the EDAC is enabled. This bit is only present when the EDAC is implemented. The *edac* bit in the *Capability register* shows if EDAC is implemented or not. Reset value 0.

7 Reserved (R) - Always reads zero.

6 EDO data output (EDO) - If programmable timing is implemented (indicated by the *prgt* field in the *Capability register*) then this bit should be set if EDO data output cycles should be used. See ONFI 2.2 Specification for more information. If programmable timing is not implemented then this bit is read-only. When programmable timing is not implemented the EDO is only enabled when timing mode 5 is selected and tm5_edoen generic is set to 1.

5 Command abort (ABORT) - This bit can be set to 1 to abort a command that for some reason has put the core in a dead lock waiting for the *rb* input signal to go high. This could happen for example if a program or erase command was issued while the memory was in write protect mode. This bit is automatically cleared by the core. Reset value 0. Only available if the *rev* field in the *Capability register* > 0, otherwise always 0.

4:2 Timing mode (TMODE) - If programmable timing is not implemented (indicated by the *prgt* field in the *Capability register*) then writing this field changes the core's internal timing mode. See ONFI 2.2 Specification for more information on the different timing modes. Note that in order to change timing mode on the flash memory devices a SET FEATURES command needs to be issued. This is not done automatically when writing these bits. Timing mode 0 i always supported. Additional supported timing modes is indicated by the *tm[5:1]* bits in the *Capability register*. Should not be written while a command is in progress. Note that if this field is written with a value matching a timing mode that is not supported, then the core will operate in timing mode 0 (even though this field still changes to the invalid value). If programmable timing is implemented then this field is not present. Reset value: 0b000

Table 2145. 0x00 - CTRL - Core control register

1 Write protect (WP) - When this bit is set to 1 the core puts the flash memory devices in write protect mode by asserting the *wp* signal. In write protect mode, the memories won't respond to PROGRAM or ERASE commands. If the core is active when software writes this bit there is a delay before the actual write protect signal goes low. Software can use the *wp* field in the *Core status register* to see when the signal has changed value. Reset value: 1

0 Software reset (RST) - If software writes this bit to 1 the core is reset, and a RESET (0xFF) command is issued to all targets on all attached flash memory devices. The only difference between a software reset and a hardware reset / power up is that the core does not reset it's *tmode* field (described above) nor the *Programmable timing registers* during software reset. The reason for this is that the flash memory devices does not change timing mode after receiving a RESET command. This bit is cleared automatically. Reset value: 0

Table 2146. 0x04 - STAT - Core status register (read only)

31	24	23	22	21	20	19	18	17	16
R		FTNERRFLAGS		R		FTAERRFLAGS		R	
0		0		0		0		0	
r		wc*		r		wc*		r	

15	12	11	9	8	4	3	2	1	0
EERRFLAGS		R		STATE		R		WP	RDY
0		0		0		0		1	0
wc*		r		r		r		r	r

31:24 Reserved (R) - Always reads zero.

23:22 Fault tolerance error flags on NAND side (FTNERRFLAGS) - The bits in this field indicates the following errors:

Bit 22: Uncorrectable error in buffer 0.

Bit 23: Uncorrectable error in buffer 1.

If the fault tolerance logic for the internal buffers indicates an error when the buffers are read during a NAND flash write operation the corresponding error flag in this register is set. These bits can be cleared by writing a 1 to them. The error flag for each buffer is also automatically cleared when an AHB write access occurs to that buffer (independent of address). These bits are only present when the internal buffers are implemented with either byte parity and DMR or only byte parity. The *ft* bits in the *Capability register* shows which fault tolerance that is implemented. Reset value 0.

21:20 Reserved (R) - Always reads zero.

19:18 Fault tolerance error flags on AHB side (FTAERRFLAGS) - The bits in this field indicates the following errors:

Bit 18: Uncorrectable error in buffer 0.

Bit 19: Uncorrectable error in buffer 1.

If the fault tolerance logic for the internal buffers indicates an error when the buffers are read over AHB the corresponding error flag in this register is set. These bits can be cleared by writing a 1 to them. The error flag for each buffer is also automatically cleared when an AHB read access occurs to that buffer with an address which is at offset 0 in the page data buffer for any data lane. For example, if the core is configured with four data lanes and 4096 byte large page buffers, then an AHB read access with offsets 0x0000, 0x1000, 0x2000, and 0x3000 (with consecutive address scheme) would reset the error flag for buffer 0. These bits are only present when the internal buffers are implemented with either byte parity and DMR or only byte parity. The *ft* bits in the *Capability register* shows which fault tolerance that is implemented.

When these bits get set then the core's output signal *err* is also set high for one clock cycle, if the *ahberren* bit in the *Core control* register is not set.

Reset value 0.

17:16 Reserved (R) - Always reads zero.

Table 2146. 0x04 - STAT - Core status register (read only)

15:12	EDAC error flags (EERRFLAGS) - The different bits indicate the following errors: Bit 12: Correctable error in buffer 0. Bit 13: Correctable error in buffer 1. Bit14: Uncorrectable error in buffer 0. Bit 15: Uncorrectable error in buffer 1. If the EDAC detects an error while an internal buffer is being read over AHB the corresponding error flag in this register is set. These bits can be cleared by writing a 1 to them. The error flags for each buffer are also automatically cleared when an AHB read access occurs to that buffer with an address which is at offset 0 in the page data buffer for any data lane. For example, if the core is configured with four data lanes and 4096 byte large page buffers, then an AHB read access with offsets 0x0000, 0x1000, 0x2000, and 0x3000 (with the consecutive address scheme) would reset the error flag for buffer 0. These bits are only present when the EDAC is implemented. The *edac* bit in the *Capability register* shows if EDAC is implemented or not. When the correctable error status bits get set then the core's output signal *err* is also set high for one clock cycle. The *err* output is also set for one cycle if the error flags for uncorrectable errors get set and the *ahberren* bit in the *Core control* register is not set. Reset value 0x0.
11:9	Reserved (R) - Always reads zero.
8:4	Core state - Shows the core's internal state. Implemented for debugging purposes. 0 = reset, 1-2 = idle, 3-8 = command state, 9-11 = address state, 12-15 = data in state, 16-18 = data out state, 19-31 = unused.
3:2	Reserved (R) - Always reads zero.
1	Write protect (WP) - Shows if the flash memory devices are in write protect mode or not. 0 = Not in write protect mode. 1 = In write protect mode. Reset value: 1
0	Core ready (RDY) - After a power up / reset this bit will be cleared. Once the core is done with it's reset procedure (waiting for ready signal from all flash memory devices and issuing a RESET command) this bit is set to 1. Reset value: 0

Table 2147. 0x08 - IPEND - Interrupt pending register

31 … 2	1	0
R	B1IRQ	B0IRQ
0	0	0
r	wc	wc

31:2	Reserved (R) - Always reads zero.
1	Buffer 1 interrupt (B1IRQ) - This bit is set to one when an interrupt linked to buffer 1 is generated. Software can clear this bit by writing 1 to it. Only present if the *b1en* bit in the *Capability register* indicates that buffer 1 is implemented. Reset value: 0
0	Buffer 0 interrupt (B0IRQ) - This bit is set to one when an interrupt linked to buffer 0 is generated. Software can clear this bit by writing 1 to it. Reset value: 0

Table 2148. 0x0C - CAP - Capability register (read only)

15 14	13 10	9	8	7	6	5	4 2	1 0
SBITS	PBITS	TM5	TM4	TM3	TM2	TM1	NLANES	NTARGS
*	*	*	*	*	*	*	*	*
r	r	r	r	r	r	r	r	r

31 28	27	26	25	24	23 22	21	20	19	18	17 16
REV	R	PRGT	SEPB	CMDO	FT	R	B1EN	EDAC	DW16	SBITS
*	0	*	*	*	*	0	*	*	*	‘
r	r	r	r	r	r	r	r	r	r	r

31:28 Revision (REV) - Indicates the revision of the core.

27 Reserved (R) - Always reads zero.

26 Programmable timing (PRGT) - 0 = Data interface timing is set at implementation time and not programmable. 1 = Data interface timing is programmable (only reset values are set at implementation time). Reflects value of VHDL generic *progtime*.

25 Separate buffers (SEPB) - Indicates if the buffers for the data lanes are shared or separate. 0 = All data lanes share buffers. 1 = All data lanes have their own buffers. When 0 a command with a data in phase should only be issued to one target and lane. A command with a data out phase can still be issued to several lanes but with the limitation that the same data will be sent to all lanes. When set to 1 several lanes can be read at the same time, and lanes can be written with individual data simultaneously. Reflects value of VHDL generic *sepbufs*.

24 Command bit order (CMDO) - Indicates the reset value for the *cmdo* bit in the *Core control register*. Reflects value of VHDL generic *cmdorder*.

23:22 Fault tolerant buffers (FT) - These bits indicate if the internal buffers in the core is implemented with fault tolerance. 0b00 = no fault tolerance, 0b01 = Byte parity DMR, 0b10 = TMR, 0b11 = Byte parity, no DMR. Reflects value of VHDL generic *ft*.

21 Reserved (R) - Always reads zero.

20 Buffer 1 enabled (B1EN) - If this bit is 1 then the core implements both buffer 0 and buffer 1, otherwise only buffer 0 is implemented. See section 112.2.2 for more information about the buffer structure. Reflects value of VHDL generic *b1en*.

19 EDAC support (EDAC) - If this bit is 1 then the core implements error detection and correction on data read from the NAND flash memory. See section 112.3 for details. Reflects value of VHDL generic *edac*.

18 16-bit memory support (DW16) - If this bit is 0 the core only support 8-bit memories. If this bit is 1 the core support both 8-bit and 16-bit memories. Only available if the *rev* field > 0, otherwise always 0. Reflects value of VHDL generic *dwidth16*.

17:14 Spare area buffer address bits (SBITS) - This field indicates how many address bits that are implemented for the buffers for the pages spare area. Add one to the value of this field to get the number of bits. The size of the buffers are $2^{\wedge(SBITS+1)}$ Reflects value of VHDL generic *sbufsize*.

13:10 Page buffer address bits (PBITS) - This field indicated how many address bits that are implemented for the page buffers. Add one to the value of this field to get the number of bits. The size (in bytes) of the buffers are $2^{\wedge(PBITS+1)}$ Reflects value of VHDL generic *pbufsize*.

9 Timing mode 5 support (TM5) - If the core does not support programmable timing (indicated by *prgt* bit described above) then this bit is 1 if the core supports timing mode 5. If programmable timing is implemented, or if the core does not support timing mode 5 then this bit is 0. Reflects value of VHDL generic *tm5*.

8 Timing mode 4 support (TM4) - If the core does not support programmable timing (indicated by *prgt* bit described above) then this bit is 1 if the core supports timing mode 4. If programmable timing is implemented, or if the core does not support timing mode 4then this bit is 0. Reflects value of VHDL generic *tm4*.

7 Timing mode 3 support (TM3) - If the core does not support programmable timing (indicated by *prgt* bit described above) then this bit is 1 if the core supports timing mode 3. If programmable timing is implemented, or if the core does not support timing mode 3then this bit is 0. Reflects value of VHDL generic *tm3*.

Table 2148. 0x0C - CAP - Capability register (read only)

6	Timing mode 2 support (TM2) - If the core does not support programmable timing (indicated by *prgt* bit described above) then this bit is 1 if the core supports timing mode 2. If programmable timing is implemented, or if the core does not support timing mode 2then this bit is 0. Reflects value of VHDL generic *tm2*.
5	Timing mode 1 support (TM1) - If the core does not support programmable timing (indicated by *prgt* bit described above) then this bit is 1 if the core supports timing mode 1. If programmable timing is implemented, or if the core does not support timing mode 1 then this bit is 0. Reflects value of VHDL generic *tm1*.
4:2	Number of flash memory devices (NLANES) - This field indicates how many 8-bit/16-bit data lanes (minus one) the core can access, i.e. number of write enable signals. A write enable signal can be connected to one or more targets (i.e. one or more flash memory devices). Reflects value of VHDL generic *nlanes*.
1:0	Number of targets per device (NTARGS) - This field indicates how many individual targets (minus one) the core can access, i.e. number of chip enable signals. A flash memory device can have one or more targets, each with an individual chip enable signal. Reflects value of VHDL generic *ntargets*. Starting from Revision 3, core can support up to 32 number of targets. Since this field can only encode up to four targets, if the number of targets is higher than four this field will show four. In order to get correct number of targets when the number of targets is higher than four the extended capability register with offset 0x40 must be used.

Table 2149. 0x10,0x20 - ROW - Buffer row address register

31 24	23 0
R	ROWADDR
0	NR
r	rw*

31:24	Reserved (R) - Always reads zero.
23:0	Row address (ROWADDR) - This field sets the three byte row address, which is used to address LUNs, blocks and pages. As described in the ONFI 2.2 specification the least significant part of the row address is the page address, the middle part block address, and the most significant part is the LUN address. Exactly how many bits that are used for each part of the address depends on the architecture of the flash memory. Software needs to write this field prior to issuing any command that has an address phase that includes the row address. The core ignores this field if the command doesn't use the row address. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.

Table 2150. 0x14,0x24 - COL - Buffer column address register

31 16	15 0
SIZE	COLADDR
NR	NR
rw*	rw*

31:16	Command data size (SIZE) - If a command has a data out or data in phase then software needs to set this field to the size of the data that should be read / written. Software does not need to set this field for the commands SET FEATURES, GET FEATURES, READ STATUS, or READ STATUS ENHANCED their data phases are always the same size. The core also ignores this field if the command issued doesn't have a data phase, as for example BLOCK ERASE. The actual number of bits used for this field depends on the size of the implemented buffers. The number of bits can be found by reading the *pbits* field of the *Capability register* and adding one. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.
15:0	Column address (COLADDR) - This field sets the two byte column address, which is used to address into a flash memory page. See the ONFI 2.2 specification for more information about column address. Software needs to write this field for those commands that have an address phase that includes the column address, as well as for those special commands that only have a one byte address phase (SET FEATURES; GET FEATURES; READ ID, READ UNIQUE ID, and READ PARAMETER PAGE). The core uses this field as an offset into the buffers when reading / writing data. The exception is the one byte address commands mentioned above, which always store their data in the beginning of the buffers. This field is ignored by the core if the command only uses the row address. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.

*Table 2151.*0x18, 0x28 - CMD - Buffer command register

31 21	20	19	18	17	16	15 8	7 0
RESERVED	DWO	R	SEL	SC2	CD	CMD2	CMD1
0	0	0	NR	NR	NR	NR	NR
r	rw*	r	rw*	rw*	rw*	rw*	rw*

31:21 Reserved (R) - Always reads zero.

20 Data width override (DWO) - Set this bit to 1 to override the *dw* bit in the *Core control register* for the current command. If this bit is 0, then the *dw* bit in the *Core control register* decides whether to use an 8-bit or 16-bit data lane for the current command. This bit is only available if the *dw16* bit in the *Capability register* is 1. Reset value 0.

19 Reserved (R) - Always reads zero.

18 Command select (SEL) - This bit is used to select between commands that have the same opcode in the first command cycle. This applies to the CHANGE WRITE COLUMN and COPYBACK PROGRAM commands, which both have a first command byte with the value of 0x85. If a COPYBACK PROGRAM is to be executed, this bit should be set to 0. If a CHANGE WRITE COLUMM command is to be executed, this bit should be set to 1. If the *rev* field in the *Capability register* > 0 then this bit is also used to select between a PAGE PROGRAM and a SMALL DATA MOVE (with opcode 0x80). If the SMALL DATA MOVE should be executed then this bit should be set to 1, otherwise it should be set to 0. The core ignores this bit for all other commands. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.

17 Skip second command phase (SC2) - If this bit is set and a program command (PAGE PROGRAM, COPYBACK PROGRAM, or CHANGE WRITE COLUMN) is being executed the core skips the second command phase for that command and jumps back to idle state once all data has been written. This is done in order to support the CHANGE WRITE COLUMN command, which needs to be executed in between the first and second phase of the program command. If a CHANGE WRITE COLUMN command is required during a PAGE PROGRAM, the PAGE PROGRAM command is issued with SC2 set to 1. After that when issuing a CHANGE WRITE COLUMN command if more than one CHANGE WRITE COLUMN commands needs to be issued all the ones apart from the last one is issued with SC" set to 0. The last CHANGE WRITE COLUMN command is issued with SC2 set to '1' this way the PAGE PROGRAM command is ended. See the ONFI 2.2 specification for details on the CHANGE WRITE COLUMN command.

*Starting from revision 4 of the core it is possible to use skip command phase if both CMD1 and CMD2 is set to READ ("00"). In this case the core will issue a single READ ("00") command. This can be used to generate the first "00" command for a two-plane read which exists in some of the NAND devices.

This bit is ignored by the core for all other commands. . This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.

16 Common data (CD) - Sometimes, for example for the SET FEATURES command, it is desirable to send the same data on all data lanes. If this is the case, software can write the data to send in the buffer corresponding to the first data lane and then set to this bit to 1. When the core executes the command it will then send the same data on all lanes without the need for software to fill all the corresponding buffers. Needs to be set to 0 if individual data should be send to the devices. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.

15:8 Second command phase (CMD2) - If the command to execute is a two byte (two phase) command then software should write the second byte of the command to this field. The core ignores this field for commands that only have one command phase. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.

7:0 First command phase (CMD1) - Software should write this field with the first byte of the command to execute. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.

*Table 2152.*0x1C,0x2C - BCS - Buffer control / status register

31 27	26	25	24	23 16	15 8	7 4	3 2	1	0
R	INV	BSY	RUN	R	LANESEL	TARGSEL[3:0]	R	IRQM	EXE
0	0	0	0	0	NR	NR	0	1	0
r	rw	wc	r	r	rw*	rw*	r	rw*	rw*

31:27 Reserved (R) - Always reads zero.

26 Invalid command (INV) - This bit is set to one if an invalid command is written to the *Buffer command register* when the *exe* bit is written. This bit is cleared automatically once a new command is started. Reset value: 0

25 Buffer busy (BSY) - Core sets this bit to 1 when a command is being executed. Once the command is done (*run* bit is cleared) software can clear this bit by writing 1 to it. While this bit is set it prevents software from writing to the buffer. This bit is write clear, but only when the *run* bit is zero. Reset value: 0

24 Command running (RUN) - Core sets this bit to 1 when a command is being executed, and clears it again automatically once the command is done. While this bit is set it prevents software from accessing the buffer. Reset value: 0

23:16 Reserved (R) - Always reads zero.

15:8 Data lane select (LANESEL) - The core uses this field to select which of the connected 8-bit/16-bit data lanes to send the command to (which write enable (WE) signals to assert). (A write enable signal can be connected to one or more flash memory devices but only one of the targets is selected at a time.) The least significant bit in this field corresponds to the first connected data lane (WE(0)) etc. The actual number of bits implemented equals the *nlanes* field in the *Capability register* plus one. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.

7:4 Target select (TARGSEL)[3:0] - The core uses this field to select which targets from [3:0] to send the command to (which chip enable (CE) signals to assert). The least significant bit in this field corresponds to the first target (CE(0)) etc. (A chip enable signal can be connected to one or more flash memory devices, all on different 8-bit/16-bit memory lanes. One or more chip enable signals can be connected to a flash memory device, depending on how many targets the device implements.) The actual number of bits implemented equals the *ntargs* field in the *Capability register* plus one. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero.

***Important note:** Starting from Revision 3 the core supports up to 32 targets. When the core is configured to have up to four targets there is no additional action needed. But when the core is configured to use more than four targets the control of targets starting from the fifth one and onwards are controlled by an extended targsel register. If the core is configured to use more than four targets please read the section " 0x44 / 0x48- Extended Buffer target select register" below carefully.

3:2 Reserved (R) - Always reads zero.

1 Interrupt mask (IRQM) - If this bit is set to 1 an interrupt will be generated when the command linked to the corresponding buffer has been executed. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero. Reset value: 1

0 Execute command (EXE) - When software writes this bit to 1 the core sends the command programmed into the corresponding *Buffer command register* to the lanes and targets selected by the *lanesel* and *targsel* field in this register. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero. Reset value: 0

*Table 2153.*0x30 - TIM0 - Programmable timing register 0

15 9	8 0
tCS	tCCS
*	*
rw*	rw*

31 30	29 24	23 16
R	tRP	tRHW
0	*	*
r	rw*	rw*

31:30 Reserved (R) - Always reads zero.

*Table 2153.*0x30 - TIM0 - Programmable timing register 0

29:24 RE pulse width (tRP) - Length of tRP in clock cycles, minus one. See ONFI 2.2 specification for more information. Field only present if programmable timing is implemented. Indicated by *prgt* bit in *Capability* register. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.

23:16 RE high to WE low (tRHW) - Length of tRHW in clock cycles, minus one. See ONFI 2.2 specification for more information. Field only present if programmable timing is implemented. Indicated by *prgt* bit in *Capability* register. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.

15:9 CE setup time (tCS) - Length of tCS in clock cycles, minus one. See ONFI 2.2 specification for more information. Field only present if programmable timing is implemented. Indicated by *prgt* bit in *Capability* register. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.

8:0 Change Column setup time (tCCS) - Length of tCCS in clock cycles, minus one. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.

Table 2154. 0x34 - TIM1 - Programmable timing register 1 (only present if programmable timing is implemented, which is indicated by *prgt* bit in *Capability* register)

15	14 8	7 5	4 0
R	tWHR	R	tWH
0	*	0	*
r	rw*	r	rw*

31 30 29 24	23 22	21 16
tWB	R	tWP
*	0	*
rw*	r	rw*

31:24 WE high to SR[6] low (tWB) - Length of tWB in clock cycles, minus one.ee ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.

23:22 Reserved (R) - Always reads zero.

21:16 WE pulse width (tWP) - Length of tWP in clock cycles, minus one. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.

15 Reserved (R) - Always reads zero.

14:8 WE high to RE low (tWHR) - Length of tWHR in clock cycles, minus one. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.

7:5 Reserved (R) - Always reads zero.

4:0 WE high hold time (tWH) - Length of tWH in clock cycles, minus one. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.

*Table 2155.*0 x38 - TIM2 - Programmable timing register 2 (only present if programmable timing is implemented, which is indicated by *prgt* bit in *Capability* register)

15	14 8	7 6	5 0
R	tWW	R	tRR
0	*	0	*
r	rw*	r	rw*

31 24	23 16
R	tADL
0	*
r	rw*

*Table 2155.*0 x38 - TIM2 - Programmable timing register 2 (only present if programmable timing is implemented, which is indicated by *prgt* bit in *Capability* register)

31:24	Reserved (R) - Always reads zero.
23:16	ALE to data loading time (tADL) - Length of tADL in clock cycles, minus one. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.
15	Reserved (R) - Always reads zero.
14:8	WP transition to WE low (tWW) - Length of tWW in clock cycles, minus one. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.
7:6	Reserved (R) - Always reads zero.
5:0	Ready to RE low (tRR) - Length of tRR in clock cycles, minus one. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.

Table 2156. 0x3C - TIM3 - Programmable timing register 3 (only present if programmable timing is implemented and core revision is bigger or equal to 2, which is indicated by *prgt* bit and *REV* bits in *Capability* register)

31 6	5 0
R	tREH
0	*
r	rw*

31:6	Reserved (R) - Always reads zero.
5:0	RE high hold time (tREH) - Length of tREH in clock cycles, minus one. For backward compatibility purposes the value of tREH is set to the value of tRP when TIM0 register is updated. As a result if the required tREH value is different than tRP, then the TIM3 has to be updated after TIM0 is updated. It should be noted that if tREH value is set something different than tRP, the tRC (tRP+tREH) timing should be met. See ONFI 2.2 specification for more information. Reset value calculated from VHDL generic *sysfreq* to match value for timing mode 0.

Table 2157. 0x40 - Extended capability register

31 5	4 0
R	NTARGS
0	*
r	r

31:5	Reserved (R) - Always reads zero.
4:0	Number of targets per device (NTARGS) - This field indicates how many individual targets the core can access, i.e. number of chip enable signals. A flash memory device can have one or more targets, each with an individual chip enable signal. Reflects value of VHDL generic *ntargets*.

Table 2158. 0x44 / 0x48- Extended Buffer target select register

31 4	3 0
TARGSEL[31:4]	R
NR	0
rw*	R

Table 2158. 0x44 / 0x48- Extended Buffer target select register

31:4	Target select (TARGSEL) [31:4] - This register should be used if the number of targets are larger than four. The core uses this field to select which targets from [31:4] to send the command to (which chip enable (CE) signals to assert). The least significant bit in this field corresponds to the fifth target (CE(4)) etc. (A chip enable signal can be connected to one or more flash memory devices, all on different 8-bit/16-bit memory lanes. One or more chip enable signals can be connected to a flash memory device, depending on how many targets the device implements.) The actual number of bits implemented equals the *ntargs-4* field in the *Capability register* plus one. This field can only be written if the *bsy* bit in the *Buffer control / status register* is zero. ***Important note:** This register is added on revision 3 in order to be able to use more than four targets. This register should be used together with 0x1C,0x2C - BCS - Buffer control / status register (for the corresponding buffer). When using this register, first the required target configuration for [31:4] should be written to this register. After that the target configuration for [3:0] should be set while writing to BCS - Buffer control / status register(bitfield[7:4] of the corresponding buffer). Since the targetsel field can not be changed after a command is executed it is highly recommended that to write this register first before writing to command register in order to avoid software mistakes.
[3:0]	Reserved (R) - Always reads zero.

Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x059. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

112.5 Timing Modes

It is highly recommended to use programmable timing to be able to customize timing, which is needed when additional timing delays are needed to be accounted like I/O delays. In addition it gives the flexibility to correct timing errors by software. See ONFI specification 2.2 for more details about timing.

112.5.1 Timing Register Values when Programmable Timing is Disabled

This section shows how the timing counter values are calculated when programmable timing is disabled. It should be noted that this values are calculated without taking additional delays into account, for example the delay from the NANDFCTRL core outputs to NANDFLASH DEVICE or vice versa. Hence it is highly recommended to use programmable timing, to mitigate additional delays and manage custom timing.

The values are floored if there is a remainder after the division. The following calculations show which values are used instead of the values that are defined in the timing registers when programmable timing is not enabled.

CLOCK_PERIOD = `1000000/sysfreq (sysfreq is a generic)`

tWW = 100/CLK_PERIOD (TM0,TM1,TM2,TM3,TM4,TM5)

tCCS = 500/CLK_PERIOD (TM0,TM1,TM2,TM3,TM4,TM5)

tADL = 200/CLK_PERIOD (TM0); 100/CLK_PERIOD (TM1); 100/CLK_PERIOD (TM2); 100/CLK_PERIOD (TM3); 70/CLK_PERIOD (TM4); 70/CLK_PERIOD (TM5);

tCS = 70/CLK_PERIOD (TM0); 35/CLK_PERIOD (TM1); 25/CLK_PERIOD (TM2); 25/CLK_PERIOD (TM3); 20/CLK_PERIOD (TM4); 15/CLK_PERIOD (TM5);

tRHW = 200/CLK_PERIOD (TM0); 100/CLK_PERIOD (TM1); 100/CLK_PERIOD (TM2); 100/CLK_PERIOD (TM3); 100/CLK_PERIOD (TM4); 100/CLK_PERIOD (TM5);

tRP = 50/CLK_PERIOD (TM0); 30/CLK_PERIOD (TM1); 25/CLK_PERIOD (TM2); 20/CLK_PERIOD (TM3); 20/CLK_PERIOD (TM4); 16/CLK_PERIOD (TM5);

tREH = 50/CLK_PERIOD (TM0); 20/CLK_PERIOD (TM1); 15/CLK_PERIOD (TM2); 10/CLK_PERIOD (TM3); 10/CLK_PERIOD (TM4); 7/CLK_PERIOD (TM5);

tWH = 30/CLK_PERIOD (TM0); 15/CLK_PERIOD (TM1); 15/CLK_PERIOD (TM2); 10/CLK_PERIOD (TM3); 10/CLK_PERIOD (TM4); 7/CLK_PERIOD (TM5);

tWHR = 120/CLK_PERIOD (TM0); 80/CLK_PERIOD (TM1); 80/CLK_PERIOD (TM2); 60/CLK_PERIOD (TM3); 60/CLK_PERIOD (TM4); 60/CLK_PERIOD (TM5);

tWP = 50/CLK_PERIOD (TM0); 25/CLK_PERIOD (TM1); 17/CLK_PERIOD (TM2); 15/CLK_PERIOD (TM3); 12/CLK_PERIOD (TM4); 10/CLK_PERIOD (TM5);

tWB = 200/CLK_PERIOD (TM0); 100/CLK_PERIOD (TM1); 100/CLK_PERIOD (TM2); 100/CLK_PERIOD (TM3); 100/CLK_PERIOD (TM4); 100/CLK_PERIOD (TM5);

tRR = 40/CLK_PERIOD (TM0); 20/CLK_PERIOD (TM1); 20/CLK_PERIOD (TM2); 20/CLK_PERIOD (TM3); 20/CLK_PERIOD (TM4); 20/CLK_PERIOD (TM5);

If TM5 mode is intended to be used when programmable timing is disabled, the user has to calculate if the *tRC* period is going to be less than 30 ns. If it less than 30 ns the tm5_edoen generic shall be set to 1 according to ONFI 2.2 specification. In other cases tm5_edoen should be set to 0.

112.5.2 tROH timing

tROH timing is equal to tRHW timing. Since minimum tRHW timing is equal to maximum tROH timing when minimum tRHW timing is met the tROH timing is met implicitly.

112.5.3 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers. The Flash interface signals have asynchronous reset.

112.5.4 Core instantiation

The core maps all usage of RAM on the *syncram* (or *syncramft* if *ft* generic is not set to 0) component from the technology mapping library (TECHMAP). The size of the instantiated RAM is determined by the *pbufsize, sbufsize, nlanes,* and *sepbufs* generics. Fault tolerance - byte parity DMR or TMR - can be added to the RAM by setting the *ft* generic to 1 or 2.

Note that both the *ft* and *edac* generics need to be set to 0 unless the core is used together with the fault tolerant GRLIB.

The core implements one interrupt, mapped by means of the *pirq* VHDL generic.

112.5.5 Scan test support

The VHDL generic *scantest* enables scan test support. If the core has been implemented with scan test support and the *testen* input signal is high, the core will:

- disable the internal RAM blocks when the *scanen* signal is asserted.
- use the *testoen* signal as output enable signal.
- use the *testrst* signal as the reset signal for those registers that are asynchronously reseted.

The *testen*, *scanen*, *testrst*, and *testoen* signals are routed via the AHB slave interface.

112.6 Implementation

112.6.1 Endianness

The cores endianness can be configured using the settings in the GRLIB configuration package (see GRLIB User's Manual).

112.7 Configuration options

Table 2159 shows the configuration options of the core (VHDL generics).

*Table 2159.*Configuration options

Generic name	Function	Allowed range	Default
hsindex	AHB slave index	0 - NAHBSLV-1	0
haddr0	AHB slave address for BAR 0. See section 112.2.2 for explanation of address scheme.	0 - 16#FFF#	16#000#
haddr1	AHB slave address for BAR 1. See section 112.2.2 for explanation of address scheme.	0 - 16#FFF#	16#001#
hmask0	AHB slave address mask for BAR 0. If set to zero, BAR 0 is disabled. See section 112.2.2 for explanation of address scheme.	0 - 16#FFF#	16#FFF#
hmask1	AHB slave address mask for BAR 1. If set to zero, BAR 1 is disabled. See section 112.2.2 for explanation of address scheme.	0 - 16#FFF#	16#FFF#
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	APB irq number.	0 - NAHBIRQ-1	0
memtech	Memory technology used for the buffers.	0 - NTECH	inferred
sysfreq	System clock frequency in kHz.	1 - 1 000 000	50000
ntargets	Number of targets = Number of chip select signals connected to the core. A flash memory device can have one or more targets.	1 - 32	2
nlanes	Number of 8-bit/16-bit data lanes = Number of write enable signals. A write enable signal can be connected to one or more targets (i.e. one or more flash memory devices).	1 - 8	8
dwidth16	0 = Core implements 8-bit data lanes. 1 = Core implements 16-bit data lanes.	0 - 1	0
pbufsize	Size of each page buffer (in bytes). One or two page buffers are implemented for each connected flash memory device (depends on value of generic *buflen*.). Generic needs to be set to a value equal to or greater than the flash memory device's page area. Value also needs to be a power of two.	8, 16, 32 ... 32768	4096
sbufsize	Size of each spare area buffer (in bytes). One or two spare buffers are implemented for each connected flash memory device (depends on value of generic *buflen*.). Generic needs to be set to a value equal to or greater than the flash memory device's page spare area. Value also needs to be a power of two.	8, 16, 32 ... 32768	256
tm1	Enable support for timing mode 1 in the case with fixed data interface timing (*progtime* generic set to 0). This generic has no effect if the *progtime* generic is set to 1.	0 - 1	0

*Table 2159.*Configuration options

Generic name	Function	Allowed range	Default
tm2	Enable support for timing mode 2 in the case with fixed data interface timing (*progtime* generic set to 0). This generic has no effect if the *progtime* generic is set to 1.	0 - 1	0
tm3	Enable support for timing mode 3 in the case with fixed data interface timing (*progtime* generic set to 0). This generic has no effect if the *progtime* generic is set to 1.	0 - 1	0
tm4	Enable support for timing mode 4 in the case with fixed data interface timing (*progtime* generic set to 0). This generic has no effect if the *progtime* generic is set to 1.	0 - 1	0
tm5	Enable support for timing mode 5 in the case with fixed data interface timing (*progtime* generic set to 0). This generic has no effect if the *progtime* generic is set to 1.	0 - 1	0
tm5_edoen	Enable EDO data output cycles when timing mode 5 is used in the case fixed data interface timing (progtime generic set to 0). This generic has no effect if the *progtime* generic is set to 1. This generic is only available if the core revision is bigger or equal to 2.	0 - 1	0
nsync	Number of synchronization registers on R/B input. (Data input is always synchronized through one set of registers.)	0 - 3	2
ft	This generic determines if fault tolerance should be added to the buffers. 0 = no fault tolerance, 1 = Byte parity DMR, 2 = TMR. 3 = Byte parity, no DMR, 5 = Technology specific protection. Note that this generic needs to be set to 0 if the core is used together with the GPL version of GRLIB, since that version does not include any fault tolerance.	0 - 3, 5	0
oepol	Polarity of pad output enable signal.	0 - 1	0
scantest	Enable scan test support.	0 - 1	0
edac	Enable EDAC support. See section 112.3.1 for EDAC information. Note that this generic needs to be set to 0 if the core is used together with the GPL version of GRLIB, since that version does not include any fault tolerance.	0 - 1	0
cmdorder	Sets the default way the ONFI command bytes are mapped to the core's data lane(s). When set to 0 the commands are mapped as follows: Cmd bit 0 -> Data lane bit 7, Cmd bit 1 -> Data lane bit 6., and so on. When this bit is set to 1 the commands are mapped as follows: Cmd bit 0 -> Data lane bit 0, Cmd bit 1 -> Data lane bit 1, and so on.	0 - 1	1
sepbufs	When set to 0 all data lanes share the same internal memory buffers. When set to 1 all data lanes have their own internal memory buffers. When set to 0 a command with a data in phase should only be issued to one target and lane at the time. A command with a data out phase can still be issued to several lanes and targets but with the limitation that the same data will be sent on all lanes. When set to 1 several lanes can be read at the same time, and lanes can be written with individual data simultaneously.	0 - 1	1

*Table 2159.*Configuration options

Generic name	Function	Allowed range	Default
progtime	When set to 0 the data interface timing for the different timing modes are set at implementation time from the *sysfreq* generic. When set to 1 the data interface timing is programmable through APB registers (reset values are set at implementation time from the *sysfreq* generic). Maximum system frequency when this generic is set to 1 is 1 GHz.	0 - 1	0
buf1en	When this generic is set to 1 then both buffer 0 and buffer 1 are implemented. If set to 0, only buffer 0 is implemented. See section 112.2.2 for more information about the buffer structure.	0 - 1	1

112.8 Signal descriptions

Table 2160 shows the interface signals of the core (VHDL ports).

*Table 2160.*Signal descriptions

Signal name	Field	Type	Function	Active
rst	N/A	Input	Reset	Logical 0
clk	N/A	Input	Clock	-
apbi	*	Input	APB slave input signals	-
apbo	*	Output	APB slave output signals	-
ahbsi	*	Input	AHB slave input signals	-
ahbso	*	Output	AHB slave output signals	-
nandfi	rb	Input	Ready/Busy signal	-
	di(63:0) [2]	Input	Data input (used both for 8-bit and 16-bit lanes)	-
	dih(63:0) [2]	Input	Data input (upper byte for 16-bit lanes)	-
nandfo	ce(3:0) [3]	Output	Chip enable	Logical 0
	we(7:0) [4]	Output	Write enable	Logical 0
	do(63:0) [2]	Output	Data output (used both for 8-bit and 16-bit lanes)	-
	doh(63:0)[2]	Output	Data output (upper byte for 16-bit memories)	-
	cle	Output	Command latch enable	Logical 1
	ale	Output	Address latch enable	Logical 1
	re	Output	Read enable	Logical 0
	wp	Output	Write protect	Logical 0
	err	Output	EDAC / Buffer error on AHB access	Logical 1
	oe	Output	Output enable	[5]

* see GRLIB IP Library User's Manual

[2] The actual number of data input/output signals used depends on core configuration. Eight bits are used for each lane.

[3] The core drives one chip select signal for each target, i.e. on or more attached flash memory devices.

[4] The core drives one write enable signal for each 8-bit/16-bit data lane, i.e. one or more attached flash memory devices.

[5] The polarity of the output enable signal is implementation dependent.

112.9 Signal definitions and reset values

The signals and their reset values are described in table 2161.

*Table 2161.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
rb	Input	Ready/Busy signal	-	-
d(63:0) [1]	Input/Output	Data (used both for 8-bit and 16-bit lanes)	-	-
dh(63:0) [1]	Input/Output	Data (upper byte for 16-bit lanes)	-	-
ce(3:0) [2]	Output	Chip enable	Logical 0	Logical 1
we(7:0) [3]	Output	Write enable	Logical 0	Logical 1
cle	Output	Command latch enable	Logical 1	Logical 0
ale	Output	Address latch enable	Logical 1	Logical 0
re	Output	Read enable	Logical 0	Logical 1
wp	Output	Write protect	Logical 0	Logical 1
err	Output	EDAC / Buffer error on AHB access	Logical 1	Logical 0
oe	Output	Output enable	[4]	[4]

[1] The actual number of data input/output signals used depends on core configuration. Eight bits are used for each attached flash memory device.

[2] The core drives one chip select signal for each target, i.e. on or more attached flash memory devices.

[3] The core drives one write enable signal for each 8-bit data lane, i.e. on or more attached flash memory devices.

[4] The polarity of the output enable signal is implementation dependent.

112.10 Library dependencies

Table 2162 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2162.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MEMCTRL	Signals, component	Component declaration

112.11 Timing

The timing waveforms and timing parameters are shown in figure 299 and are defined in table 2163.

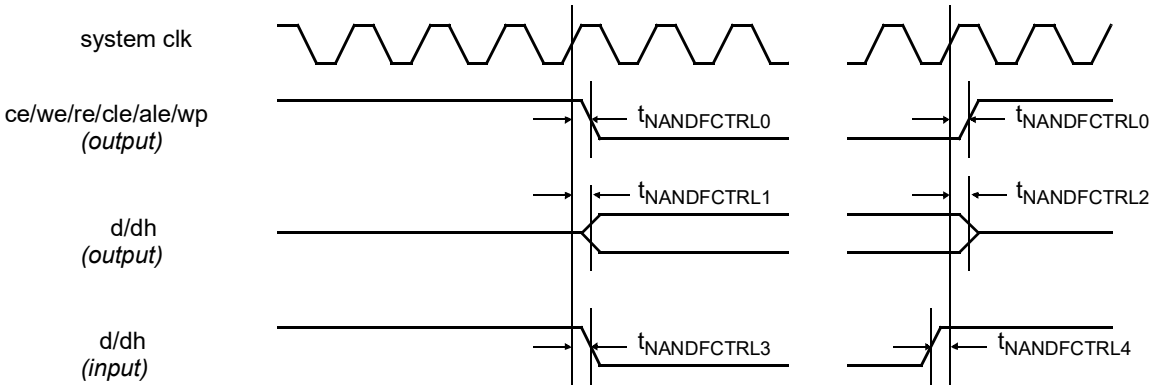


Figure 299. Timing waveforms

*Table 2163.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{NANDFCTRL0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns
$t_{NANDFCTRL1}$	clock to non-tri-state delay	rising *clk* edge	TBD	TBD	ns
$t_{NANDFCTRL2}$	clock to tri-state delay	rising *clk* edge	TBD	TBD	ns
$t_{NANDFCTRL3}$	input to clock hold	rising *clk* edge	TBD	-	ns
$t_{NANDFCTRL4}$	input to clock setup	rising *clk* edge	TBD	-	ns

Note: The *rb* input is re-synchronized internally. The signal does not have to meet any setup or hold requirements.

112.12 Instantiation

This example shows how the core can be instantiated. The instantiated core has all its generics, except *hsindex, pindex*, *paddr*, and *pirq* at their default values. The impact of the generics can be seen in table 2159.

```
library ieee, grlib, gaisler;
use ieee.std_logic_1164.all;
use grlib.amba.all;
use gaisler.nandpkg.all;

entity nandfctrl_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;
    nandf_d : inout std_logic_vector(63 downto 0);
    nandf_dh : inout std_logic_vector(63 downto 0);
    nandf_rb : in std_ulogic;
    nandf_ce : out std_logic_vector(1 downto 0);
    nandf_we : out std_logic_vector(7 downto 0);
    nandf_re : out std_ulogic;
    nandf_cle : out std_ulogic;
    nandf_ale : out std_ulogic;
    nandf_wp : out std_ulogic;
    nandf_err : out std_ulogic
    );
end;

architecture rtl of nandfctrl_ex is
```

```
  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);


  -- NANDFCTRL signals
  signal nandfo : nandfctrl_out_type;
  signal nandfi : nandfctrl_in_type;

begin

  -- AMBA Components are instantiated here
  ...

  -- NANDFCTRL core
  nand0 : nandfctrl
    generic map (hsindex => 1, pindex => 10, paddr => 10, pirq => 10)
    port map (rstn, clk, apbi, apbo(10), ahbsi, ahbso(1), nandfi, nandfo);

  -- Pads for NANDFCTRL core
  nandf_d : iopadv generic map (tech => padtech, width => 64)
                   port map (nandf_d, nandfo.do, nandfo.oe, nandfi.di);
  nandf_dh : iopadv generic map (tech => padtech, width => 64)
                   port map (nandf_dh, nandfo.doh, nandfo.oe, nandfi.dih);
  nandf_rb : inpad generic map (tech => padtech)
                   port map (nandf_rb, nandfi.rb);
  nandf_ce : outpadv generic map (tech => padtech, width => 2)
                   port map (nandf_ce, nandfo.ce);
  nandf_we : outpadv generic map (tech => padtech, width => 8)
                   port map (nandf_we, nandfo.we);
  nandf_re : outpad generic map (tech => padtech)
                   port map (nandf_re, nandfo.re);
  nandf_cle : outpad generic map (tech => padtech)
                   port map (nandf_cle, nandfo.cle);
  nandf_ale : outpad generic map (tech => padtech)
                   port map (nandf_ale, nandfo.ale);
  nandf_wp : outpad generic map (tech => padtech)
                   port map (nandf_wp, nandfo.wp);
  nandf_err : outpad generic map (tech => padtech)
                   port map (nandf_err, nandfo.err);

end;
```

113 NANDFCTRL2 - NAND Flash Memory Controller with Direct Memory Access

113.1 Overview

Note: This IP core manual describes a NAND flash memory controller which is under development. Some features are described as not implemented and will be implemented over time together with additional extensions. See individual sections.

The NANDFCTRL2 core is a memory controller for NAND flash memory devices. The core is designed to operate with Open NAND Flash Interface (ONFI) 4.0 flash memory devices. It can communicate with multiple parallel NAND flash memory devices. The core provides an AMBA AHB master to perform data write and read accesses. Configuration through generics supports a vast variation of memory devices including support for up to 64 individually addressable targets via chip enable signals, 32 ready-busy signals and 16 channels. For write accesses, it is possible to issue the same command to several targets and channels at once by selecting multiple targets or channels.

The core supports EDAC using a configurable BCH code implementation, supporting up to 60-bit correcting capability. When using EDAC, NANDFCTRL2 will automatically split the NAND flash memory area into data area, spare area, and flag area. The core also supports automatic data randomization. Randomization and EDAC may be independently enabled or disabled.

The core supports up to 3 different NAND flash memory configurations and up to 2 ECC configurations simultaneously, configured through generics. In addition to generic configuration, configuration of NANDFCRTL2 is done through a set of registers, accessible through the AHB and APB interfaces.

An access to a NAND flash memory device through NANDFCTRL2 is done using a descriptor which holds necessary information to perform an access. These descriptors may be transferred to the core using either the APB or AHB interface. Multiple accesses can be setup in AHB area memory via descriptors in a linked list. This allows NANDFCTRL2 to operate more autonomously and fetch the descriptors one by one using the AHB interface.

Other key features of the core are

- Time out monitoring using the ready-busy input signals.
- A user-controlled external digital output signal, which may be used for e.g. SEFI handling.

A PHY should typically be used between the controller and the flash memory devices.

For details about the actual NAND flash memory interface, NAND flash memory architecture, and ONFI 4.0 command set, please refer to the *Open NAND Flash Interface specification, revision 4.0*, hereafter called the ONFI 4.0 specification.

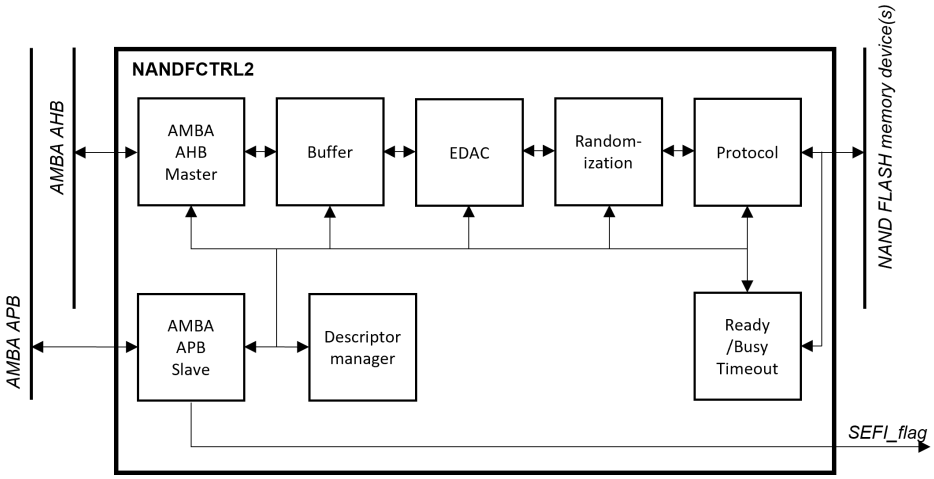


Figure 300. Block diagram

113.1.1 Limitations of current revision and planned features

The core is under development and available with limitation in functionality and features as described in the following list:

- All mandatory ONFI commands are implemented and supported as well as some vendor-specific commands. See section 113.1.2 for details.
- SDR data interface, 8-bit data bus, timing mode 0-5 is supported, with EDO.
- NV-DDR2 and NV-DDR3 data interface, 8-bit data bus, timing mode 0-10 is supported.
- NV-DDR data interface is not supported.
- The alternative way of using the Read Status command continually pulsing RE_n or leaving RE_n low is not supported.
- NAND flash memory die data bus size of 16 bits is not supported.
- **The current version of the core requires the clock frequency of clk_sys to be larger or equal to clk_core.**
- **The current version of the core requires the cycle time of DQS and RE_n (tDSC and tRC) to be at least 2 core clock periods (8 phy clock periods) if using the NV-DDR2/3 data interface. This limitation does not apply to the SDR (asynchronous) data interface.**
- **Scatter and gather have the following limitations:**

  - All fragments must have a size which is a multiple of the AHB bus width.

  - The first fragment must start at column address = 0.

  - The first fragment must not have exactly the size of a chunk.

  - If a contains exactly two fragments, the second one must not cross a chunk border.

  All of the scatter and gather related limitations will be removed in future releases.

113.1.2 Supported ONFI and vendor-specific commands

All mandatory ONFI commands and some optional commands supported in this revision of the IP as summarized in the table below.

Table 2164. Implemented ONFI commands for NANDFCTRL2.

Command	Optional/ Mandatory	1st Cycle	2nd Cycle	Supported
Read	M	00h	30h	YES
Read Multi-plane	O	00h	32h	YES
Copyback Read	O	00h	35h	YES
Change Read Column	M	05h	E0h	YES
Change Read Column Enhanced	O	06h	E0h	YES
Read Cache Random	O	00h	31h	YES
Read Cache Sequential	O	31h		YES
Read Cache End	O	3Fh		YES
Block Erase	M	60h	D0h	YES
Block Erase Multi-plane	O	60h	D1h	YES
Read Status	M	70h		YES
Read Status Enhanced	O	78h		YES
Page Program	M	80h	10h	YES
Page Program Multi-plane	O	80h	11h	YES
Page Cache Program	O	80h	15h	YES
Copyback Program	O	85h	10h	YES
Copyback Program Multi-plane	O	85h	11h	YES
Small Data Move	O	85h	11h	YES
Change Write Column	M	85h		YES
Change Row Address	O	85h		YES
Read ID	M	90h		YES
Volume Select	O	E1h		YES
ODT Configure	O	E2h		YES
Read Parameter Page	M	ECh		YES
Read Unique ID	O	EDh		YES
Get Features	O	EEh		YES
Set Features	O	EFh		YES
LUN Get Features	O	D4h		YES
LUN Set Features	O	D5h		YES
ZQ Calibration Short	O	D9h		YES
ZQ Calibration Long	O	F9h		YES
Reset LUN	O	FAh		YES
Synchronous Reset	O	FCh		YES
Reset	M	FFh		YES

In addition to the ONFI command set the core also implements support for some vendor-specific commands of the UT81NDQ512G8T NAND flash memory device. These are listed in the table below. Refer to the data sheet of the mentioned device for more information about these commands.

Table 2165.

Command	1st Cycle	2nd Cycle	Comment
SLC Mode Enable	DAh		
SLC Mode Disable	DFh		
Read Single Soft Bit	00h	34h	
Read Page with Soft Information	33h	30h	Requires the feature 97h to be set
Soft information readout	36h		
Hard Reset	FDh		

113.2 Physical interface, data interface and timing modes

113.2.1 Physical interface

NANDFCTRL2 should typical be used together with a PHY, refer to the NANDFCTRL2 PHY section of this manual. An instantiation example of NANDFCTRL2 together with a PHY is included in the end of this NANDFCTRL2 chapter.

113.2.2 Interface configurability

The pin interface of the core can be configured to support many different memory configurations as well as several devices in parallel. This is done through the three generics *nrofce*, *nrofrb* and *nrofch*. *nrofce* sets the number of CE_N signals. *nrofrb* sets the number of RB_N and SEFI_FLAG signals. *nrofch* controls the other signals (DQ(7:0), DQS, DQ_OE, DQS_OE, CLE, ALE, RE_N, WE_N and WP_N).

There are also control, status and scantest signals connected to the PHY where ddr_mde, ddr_valid, datai_valid are set by *nrofch* and ifsel, t_dqsh, t_dqsl, t_reh, t_rp, abort, skew_ctrl_rx, skew_ctrl_tx, phy_ready and scantest signals are shared for all channels in the PHY.

This gives the user the possibility to configure the core for a specific system including:

- Memory package configuration.
- Number of memory devices on a PCB design.
- PCB configuration, including possible pin-sharing of one or more signals.

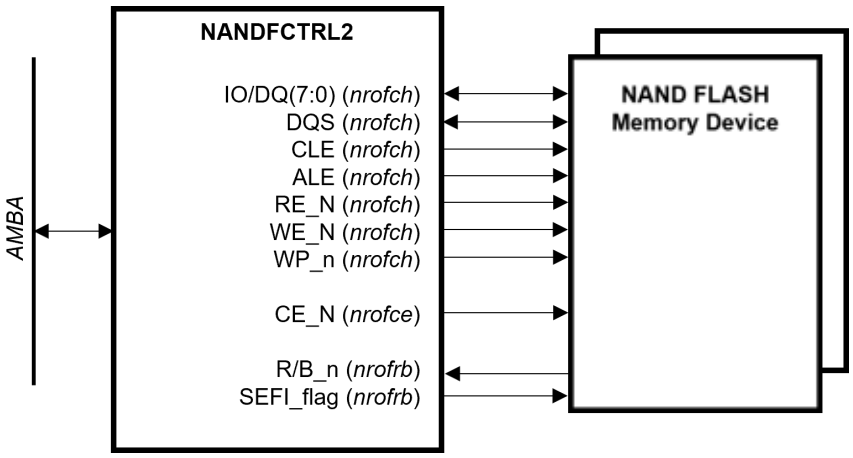


Figure 301. Interfaces

Some NAND flash memory devices does not populate all signals in direct relation to the *nrofch* generic. One example is a device using 8 RE_n and 8 WE_N signals but 1 ALE, 1 CLE, 1 WP and 1 8-bit data bus. In this case *nrofch* should be set to the higher value and the controller should enable the *channel mode* bit in the *Core control register 0*. Channel mode is described in more detail in section 113.5.

113.2.3 Data interfaces and timing modes

NANDFCTRL2 supports the data interfaces NV-DDR2, NV-DDR3 and SDR configured by the *IFSEL* field in the *Core control 0 register*.

Timing mode 0-5, including Extended data output (EDO) are supported in SDR. Timing mode 0-10 are supported in NV-DDR2 and NV-DDR3. Timing mode is configured through a number of programmable timing parameters. Those timing parameters are fully programmable via the *Programmable timing registers (0-11)* and EDO is controlled by the *EDO* field in the *Core control 0 register.* The actual time is derived from these timing registers and the frequency of the core clock and the PHY clock. The dependence of the PHY clock is only applicable when running NV-DDR2 or NV-DDR3 data interface.

The programmable timing registers are mapped by name to the corresponding timing parameters in ONFI 4.0. The user must ensure that the timing parameter registers are set according to the constraints in ONFI 4.0, chapter 4 and the NAND flash memory device.

113.2.4 Data interface SDR

The ONFI 4.0 timing parameters which should be programmed by the user are listed in the table below. All timing parameters are derived from the frequency of the core clock when configured to SDR.

Table 2166. Timing parameters in SDR

Timing Parameter	Description	Clock	Comment
tCS	CE_n setup time	Core clock	Should be configured to the value of tCS3.
tWW	WP_n transition to WE	Core clock	
tRR	Ready to RE_n low	Core clock	
tWB	WE_n high to SR[6] low	Core clock	
tRHW	RE_n high to WE_n low	Core clock	
tWHR	WE_n high to RE_n low	Core clock	
tADL	ALE to data loading time	Core clock	
tCCS	Change Column setup time	Core clock	
tREH	RE_n high hold time	Core clock	
tRP	RE_n pulse width	Core clock	
tWH	WE_n high hold time	Core clock	
tWP	WE_n pulse width	Core clock	
tVDLY	Delay prior to issuing the next command after a new Volume is selected using the Volume Select command	Core clock	
tCS_VOL	CE_n setup time for Volume Select command.	Core clock	This timing parameter is required for some flash devices that have a longer CE_N setup time than tCS/tCS3 for the Volume select command. In case the target flash device doesn't specify the Volume select setup time specific this parameter should be configured to the same value as tCS/tCS3. Not that this is not an ONFI timing parameter.

In addition the timing parameters needs to comply with the following requirements:

- tCS should always be configured to the value of tCS3, as specified in ONFI 4.0 errata 005.
- tRP should always be configured to a higher value than tREA when EDO is disabled.
- tWH and tWP needs to be configured to comply with the requirement of tWC.
- tREH and tRP needs to be configured to comply with the requirement of tRC.

If the timing parameters above form a valid configuration, other timing requirements are either fulfilled through design, or handled by the NAND flash memory devices. See the ONFI 4.0 specification for details about the different timing parameters.

113.2.5 Data interface NV-DDR2 and NV-DDR3

The ONFI 4.0 timing parameters which should be programmed by the user are listed in the table below. In NV-DDR2 and NV-DDR3 mode the timing parameters are derived from either the core clock or the phy clock, as indicated in the same table.

Table 2167. Timing parameters in NV-DDR2 and NV-DDR3

Timing Parameter	Description	Clock	Comment
tCS	CE_n setup time	Core clock	Should be configured to the value of tCS3.
tWW	WP_n transition to WE	Core clock	
tRR	Ready to RE_n low	Core clock	
tWB	WE_n high to SR[6] low	Core clock	
tRHW	RE_n high to WE_n low	Core clock	
tWHR	WE_n high to RE_n low	Core clock	
tADL	ALE to data loading time	Core clock	
tCCS	Change Column setup time	Core clock	
tREH	RE_n high hold time	PHY clock	Used by PHY.
tRP	RE_n pulse width	PHY clock	Used by PHY.
tWH	WE_n high hold time	Core clock	
tWP	WE_n pulse width	Core clock	
tCALS	CLE and ALE setup time	Core clock	Should be configured to tCALS2 if ODT is enabled.
tCSD	ALE, CLE, WE_n hold time from CE_n high	Core clock	
tCHZ	CE_n high to output Hi-Z	Core clock	
tWPRE	DQS write preamble	Core clock	Should be configured to tWPRE2 if ODT is enabled.
tRPST	Read postamble	Core clock	
tDQSRH	DQS hold time after (RE_n low or RE_t/RE_c crosspoint)	Core clock	This parameter is equal to the number of clock cycles and not minus as other timing parameters, see the register section for more information.
tWPST	DQS write postamble	Core clock	
tWPSTH	DQS write postamble hold time	Core clock	
tDQSL	DQS low level width	PHY clock	Used by PHY.
tDQSH	DQS high level width	PHY clock	Used by PHY.
tVDLY	Delay prior to issuing the next command after a new Volume is selected using the Volume Select command	Core clock	
tCS_VOL	CE_n setup time for Volume Select command.	Core clock	This timing parameter is required for some flash devices that have a longer CE_N setup time than tCS/tCS3 for the Volume select command. In case the target flash device doesn't specify the Volume select setup time specific this parameter should be configured to the same value as tCS/tCS3. Not that this is not an ONFI timing parameter.

In addition the timing parameters needs to comply with the following requirements:

- tCS should always be configured to the value of tCS3, as specified in ONFI 4.0 errata 005.

- tWH and tWP needs to be configured to comply with the requirement of tWC.
- tREH and tRP needs to be configured to comply with the ONFI requirement of tRC. tRC needs to be configured to an integer value of core clock periods.
- tDQSL and tDQSH needs to be configured to comply with the ONFI requirement of tDSC. tDSC needs to be configured to an integer value of core clock periods.
- **The current version of the core requires the cycle time of DQS and RE_n (tDSC and tRC) to be at least 2 core clock periods (8 phy clock periods) if using the NV-DDR2/3 data interface. This limitation does not apply to the SDR (asynchronous) data interface.**

If the timing parameters above form a valid configuration, other timing requirements are either fulfilled through design, or handled by the NAND flash memory devices. See the ONFI 4.0 specification for details about the different timing parameters.

113.3 AMBA interface and accessing the flash memory device

The core features an AMBA AHB master with DMA and an AMBA APB slave interface. The APB registers hold all configuration and status bits of the core.

113.3.1 Descriptor

An access to a NAND flash memory device through the core is done by setting up and triggering a *descriptor*. A descriptor is the information which is used and expected to be updated per access. Their names are prefaced with the word *Descriptor*.

There are two methods of doing this. The first method is by setting up a linked list of descriptors in an AHB memory area and letting the AHB master of NANDFCTRL2 fetch them over the AHB interface.

The other method is to configure an individual descriptor by setting the appropriate register fields through the APB interface.

113.3.2 Linked list mode

When using linked list mode, the descriptors are realised as a linked list in the host system's AHB area memory. The key benefit of using this mode is to offload the host processor, and to do accesses back-to-back without manual operations with the CPU between them.

To set up the controller to use a linked list, the following information must be transferred over the APB interface

- The *LLM* bit in the *CTRL0 register* shall be 1.
- The *Linked list pointer registers* shall point to the first descriptor in the linked list

When these are set and the rest of the descriptor is configured, setting the *Descriptor trigger* bit will start the execution of the linked list.

The information in a single descriptor must be arranged in the AHB memory according to the following table.

*Table 2168.*Memory offset and layout of a descriptor block.

Address offset	Register
0x00	**Descriptor command register**
0x04	**Descriptor target select 0 register**
0x08	**Descriptor target select 1 register**
0x0C	**Descriptor channel select register**
0x10	**Descriptor ready/busy select register**
0x14	**Descriptor row & control register**
0x18	**Descriptor column & size register**
0x1C	**Descriptor status register**
0x20	**Descriptor ECC status register**
0x24, 0x28, 0x2C	**Reserved addresses**
0x30	**Descriptor data pointer low register**
0x34	**Reserved address**
0x38	**Descriptor linked list pointer low register**
0x3C	**Reserved address**

Each descriptor has an *EN* bit, which is used to signal whether a descriptor has yet been processed. The handshake mechanism is such that after the host system has set the *EN* bit to '1', it will not change any fields in that descriptor. Conversely, when the *EN* bit is '0', NANDFCTRL2 will not change that descriptor. In practice, this gives the following behaviour.

If an individual descriptor has the *EN* bit in the descriptor command register configured to '1', NANDFCTRL2 will execute that descriptor and fetch the next one at the AHB memory location specified in the linked list pointers. If the *EN* bit in a descriptor is '0', the controller does not execute that descriptor and considers the linked list to be terminated. Upon completion of a descriptor, including communication with the NAND flash memory device and possible data transfer, the core will update the status word of the descriptor in memory, and set the EN bit to '0'.

For an example of setting up a linked list, see section 113.13.2

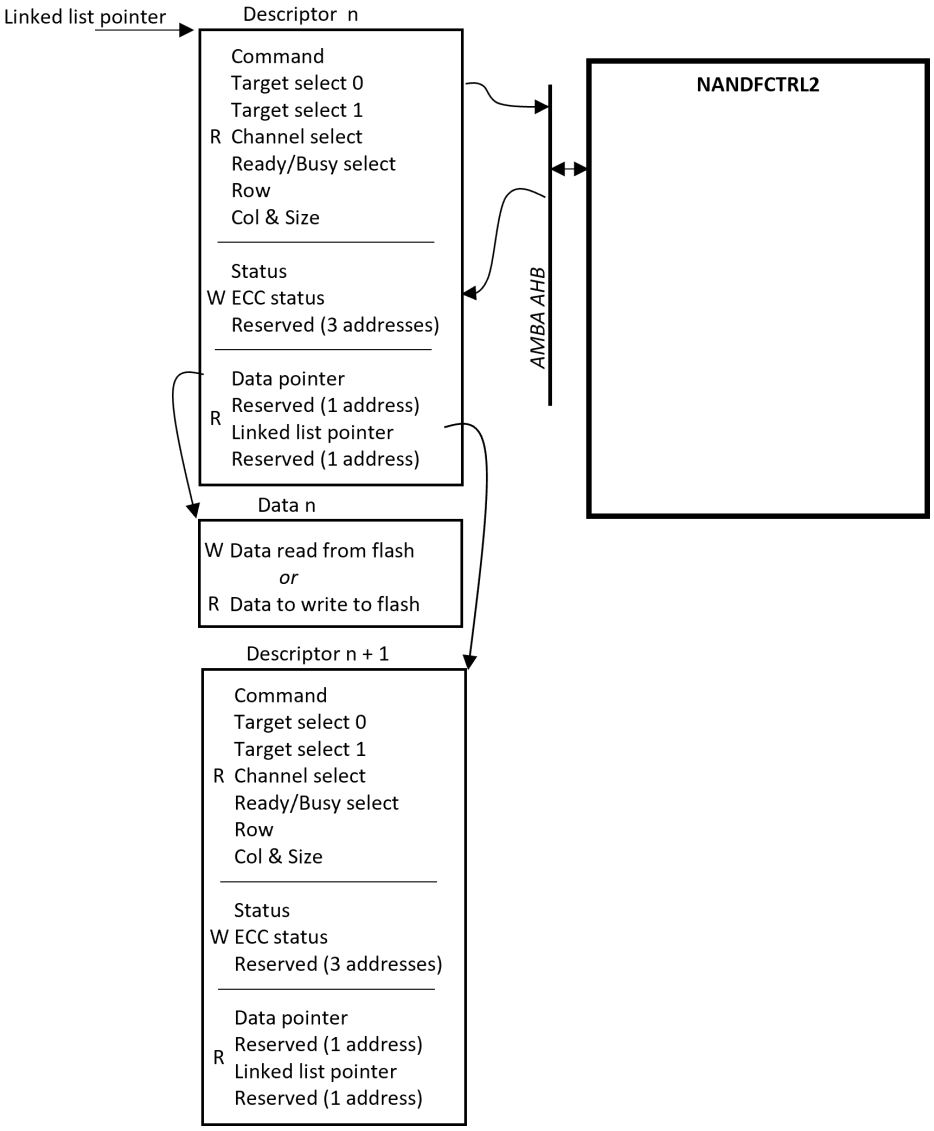


Figure 302. Example of a linked list

113.3.3 Register mode

When using register mode, a descriptor is realised by setting the registers whose names start with *Descriptor* using the APB interface. This mode is chosen by setting the *LLM* bit in the *CTRL0 register* to 0.

After the descriptor has been set to its appropriate values, the descriptor is triggered by setting the *Descriptor trigger* to 1. After the descriptor has been completely processed, the *EN* bit will be set to 0 by the hardware, while leaving the other configuration registers unchanged.

113.3.4 DMA and Data-in and Data-out registers

An ONFI 4.0 compliant NAND flash memory device has configuration and status register as well as a flash memory array, for more details see the ONFI 4.0 specification.

A NAND flash memory device's registers may be accessed using either the AHB bus via DMA master or the dedicated datain/dataout registers. Accesses can be done with either option with the restriction that a maximum of 16 bytes may be used with the APB option.

The choice of data source is determined by the DMA selection bit *DD* (DMA Disable) in the *Descriptor command register*.

If DMA mode is used, the memory location of the data is determined by a data pointer in the descriptor.

The APB interface mode is only supported in Register mode, i.e when the *LLM* bit in the *CTRL0 register* is set to 0.

The EDAC and randomization features are not supported when the APB interface is used, see section 113.4.2 and 113.4.3 for more information about EDAC and randomization.

Note that the interrupt behavior might be different depending on this setup, e.g for a read command an interrupt is generated when data is available in the Data-in register(s) or when the transfer over the AHB bus is completed.

113.4 Flash Memory Array

This section includes some preliminaries regarding a NAND flash memory device, related core features, and the different ways NANDFCTRL2 can be used to access the array efficiently. For details about a specific NAND flash memory, please refer to the corresponding data sheet.

Memory array data is typically transfered over the AHB bus using DMA.

113.4.1 Flash Memory Device

Recall that a NAND flash memory device has configuration and status registers as well as a flash memory array. The device holds one or more page registers that may be viewed as divided into data and spare area.

113.4.2 Error Detection And Correction

A NAND flash memory device typically comes with a requirement on some type of error correction, see the ONFI 4.0 specification section 3.4. In addition to correction, the core also has an implementation of a detection algorithm for uncorrectable errors.

The EDAC typically works on sizes which are smaller than the page data area dividing it into smaller partitions which here are referred to as chunks. For this reason a fixed layout scheme for the data area of the page is used when EDAC is enabled. Those chunks can be split into data chunks and the ECC produced check bytes. The number of ECC-generated check bytes in the chunk may allow for some extra bytes in the end of a page in the spare area. Those bytes are here referred to as flag area and may be used for any purpose.

The core implements a configurable BCH EDAC. Up to 2 blocks with different configuration may be instantiated simultaneously. The EDAC may be bypassed at runtime for each access. See the BCH encoder and decoder chapters for more information.

The number of check bytes per chunk produces depends on the generics and is equal to

- 2 * ceiling(eccX_cap * eccX_gfsize / 16)

i.e, the number bits necessary, rounded up to nearest even number of bytes.

The data chunks are placed in the page data area and the check bytes in the spare area.

The figure below shows 3 typical examples how the core splits the page register into data chunks, preserve bad block area, chunks of check bytes and flag area.

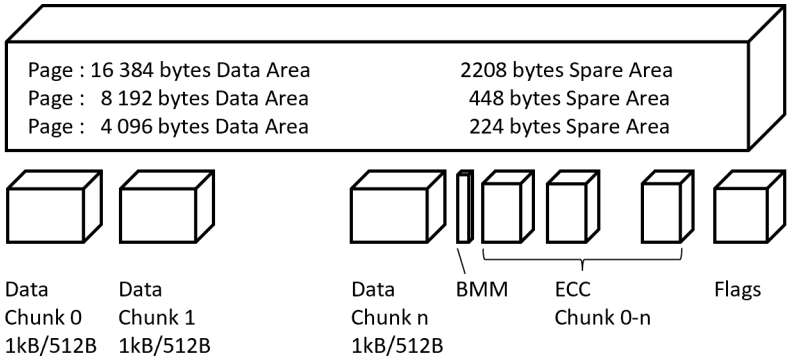


Figure 303. Page - example of page layout (set by generic parameters).

113.4.3 Randomization

Some NAND flash memory devices put a requirement on randomizing of the data.

The core has support for a generic-controlled pseudo-randomization algorithm, with periodicity and randomness behaviour in accordance with the requirements for e.g. NAND Flash Memory Device UT81NDQ512G8T. The algorithm uses the row and column addresses as seed input. This ensures the randomization is invertible regardless of start address for read or write commands.

Should randomization not be required, this functionality can be bypassed during runtime.

See section 113.10 for more details regarding the randomization.

113.4.4 Preserve Bad Block Mark

A NAND flash memory device comes with factory bad block marks on defective blocks.

The core has a feature for preserving those two bytes. This may be useful during e.g. initial setup or debugging when the bad block mark must not be overwritten by mistake. When the *BBM* field is set to '1' in *CTRL0 register* the controller avoids using the first 2 bytes in spare area for PAGE WRITE and CHUNK WRITE accesses.

Note that it is still possible to write to these 2 bytes by explicitly by setting the page column address to these bytes (See section 113.4.9).

Note that when using the Preserve Bad Block feature and randomization, those two bytes will not be randomized.

113.4.5 Memory Selection

The core supports up to 3 memory configurations and 2 ECC configurations simultaneously. Those are set through generics and are size of data area, size of spare area, size of chunk size, ecc capability and ecc gf size. Each memory configuration is connected to a ECC configuration through a selection generic. The user setup select memory configuration, which includes ecc configuration, to access with the *MSEL* field in the *CTRL0 register*.

The following 3 memory and ECC configurations are the ones NANDFCTRL2 will be configured to support using the default values of the generics.

- page size of 16 kbytes data size and 2208 bytes spare size, 16 chunks of 1024 bytes each and 60 error detection and correction capability.
- page size of 8 kbytes and 448 bytes spare size, 16 chunks of 512 bytes each and 16 error detection and correction capability.

- page size of 4 kbytes and 224 bytes spare size, 8 chunks of 512 bytes each and 16 error detection and correction capability.

113.4.6 Configuring the NAND flash array - Set and Get Feature

The core can be used for setting and reading NAND flash device's configuration. When the core is using the set or get feature command, the row address field is used for the feature address. Recall that the row address is three bytes. Should only two bytes be needed for the address, the lower bytes are used.

113.4.7 Accessing the flash array - Chunk Access

If the configured size and address equal a chunk size and an integer multiple of a chunk size respectively, this type of access will be performed. The core will write data to the data area of the page register.

If EDAC is enabled the core will automatically write the check bytes to the corresponding part of the spare area when the data area is accessed. If Preserve Bad Block is enabled the placement of the check bytes will have an offset of 2 bytes. Note that spare area is only written when EDAC or randomization is enabled.

If randomization is enabled and EDAC disabled, the data area will be randomized. The spare area corresponding to where the check bytes would have been placed will be filled with pseudo-randomized data. Writing the last chunk of a page will also randomize the flag area of that page.

In case the page write is completed and terminated without the full data area of the page register written, the controller will write additional bytes for certain configurations. If randomization is enabled, the controller will randomize the remaining part of the data area and corresponding spare area, including flag area, taking the preserve bad block mark configuration into consideration. If EDAC is enabled and randomization disabled, the controller will write the value 0 to the same addresses.

The number of accesses (a combination of Page Program and Change Write Column / Change Read column) the core perform will depend on configuration of EDAC and randomization. The core updates the descriptor status and then generates an interrupt.

If more than one chunk of data is to be programmed, several accesses will be required before starting the flash array programming, i.e. with a 10h command.

113.4.8 Accessing the flash array - Whole Page Access

If the configured size is equal to the data area for the selected memory, this type of access will be performed. The core will perform a loop of Chunk Accesses for all chunks and program the data into the NAND flash memory array. Randomization, EDAC and Preserve Bad block work in the same way as for Chunk Accesses, meaning that the whole page including flag area will be programmed if EDAC or Randomization is enabled and Preserve Bad block is disabled. The core generates an interrupt upon completion if enabled.

It is possible to access the flash array with a few additional bytes than a full page, e.g. file system or other data, using the feature called tag enable. This can be used if EDAC is enabled and the size is configured to the data area of the selected memory. When enabled the controller will handle these additional tag bytes during the last chunk access. Those extra data bytes will be protected by ECC and will be stored in the spare area together with the ecc bytes of the last chunk. In the case of a write to the NAND flash memory, the user needs to make sure that the tag data to be written is located next to the page data in the memory. In the case of a read, the tag data will be placed next to the page data in the memory. The size of the tag data is configurable through a generic.

113.4.9 Accessing the flash array - Free size

The NAND flash memory page register can be accessed freely up to 4096 bytes. If the address and size do not trigger a Chunk Access nor Whole Page Access, this type of access will be performed.

The core will write the data to the page register. If randomization is enabled, the data will be randomized. The core generates an interrupt upon completion if enabled.

EDAC is not supported for this type of access.

If more data than the NANDFCTRL2 maximum of 4096 bytes is to written to a page, more than one free size access of the page register, several accesses will have to be performed before starting the flash array programming, i.e. with a 10h command.

In the case randomization is enabled and the page write is terminated without the whole page register is written, the controller will randomize the remaining part of the page taking the preserve bad block mark configuration into consideration.

Care must be taken in order to not accidentally overwrite the factory bad block mark using this type of access and configured column address.

113.4.10Accessing the flash array - Gather

If the user has one or more fragments of a size which is not a multiple of a chunk, this feature may be used.

When writing to the NAND flash memory array, if EDAC is enabled and the size of the access does not equal one chunk nor one page, this type of access will be performed.

When NANDFCTRL2 enters gather mode, it will stay in gather mode until a write is finished, i.e. a *Change Write Column* command is terminated with 0x10.

When in gather mode, NANDFCTRL2 will accept one or more fragments of size no more than one chunk, glue them together, add checkbits and write to the NAND flash memory array. When using this feature, the user must ensure the following conditions are met.

- No fragment is more than one chunk.
- The total size before terminating is no more than one page.
- EDAC is on during the entire access. Randomization does not toggle during the entire access.
- The first data gather mode needs to be aligned to a chunk.

In the case randomization is enabled and the page write is terminated without the whole page register is written, the controller will randomize the remaining part of the data area and corresponding spare area, including flag area, taking the preserve bad block mark configuration into consideration.

In the current release the following limitation apply:

- **All fragments must have a size which is a multiple of the AHB bus width.**
- **The first fragment must start at column address = 0.**
- **The first fragment must not have exactly the size of a chunk.**
- **If a contains exactly two fragments, the second one must not cross a chunk border.**

All of the these limitations will be removed in future releases.

113.4.11Accessing the flash array - Scatter

If one or more fragments of an ECC protected memory which do not equal en exact chunk are to be read, this feature may be used.

When reading from the NAND flash memory array, if EDAC is enabled and the size of the access does not equal one chunk nor one page, this type of access will be performed.

When NANDFCTRL2 enters scatter mode, the controller will leave scatter mode on the first command which is not CHANGE READ COLUMN.

When in scatter mode, NANDFCTRL2 will transmit one or more fragments of size no more than one chunk, correct any bit flips, up to the configured capability and send them to the desired data area. When using this feature, the user must ensure the following conditions are met.

- No fragment is more than one chunk.
- The total size before terminating is no more than one page.
- EDAC is on during the entire access. Randomization does not toggle during the entire access.

In the current release the following limitation apply:

- **All fragments must have a size which is a multiple of the AHB bus width.**
- **The first fragment must start at column address = 0.**
- **The first fragment must not have exactly the size of a chunk.**
- **If a contains exactly two fragments, the second one must not cross a chunk border.**

All of the these limitations will be removed in future releases.

113.5 Channel and target selection

The core features the possibility to write to any number of channels and targets simultaneously, with the same command and data payload, using the *Descriptor target select* and the *Descriptor channel select registers*. Read is limited to one channel and target per access. In the case of more than one channel or target activated simultaneously the lowest active will be used.

Configuration of which R/B_n signal to listen to on a per access basis, using the *Descriptor ready/busy select register*. Incorrect setting lead to the core monitoring wrong target and thus not appropriately detecting when an access is finished.

Some NAND flash memory devices use 8 RE_N and WE_N signals while only 1 ALE, CLE, WP and 1 8-bi IO data bus. To support these devices the controller features a control bit named *Channel mode* located in the *Core control register 0.* When this bit is set to 1 only bit 0 and 8 will be valid for CLE, ALE and IO0-8, e.g.

ALE(0) = CH_SEL(0) or CH_SEL(1) or … or CH_SEL(7)

ALE(8) = CH_SEL(8) or CH_SEL(9) or … or CH_SEL(15)

Channel mode is only supported when using the SDR data interface.

Note that WP are always handled separately and are not effected.

113.6 Clock, Reset, Abort and Stop linked list

113.6.1 Clock

NANDFCTRL2 works as a bridge from the so called sys clk used by the host system and the AMBA interface, through the so called core clk used by the core of the controller, to the target so called phy clk of the target PHY. The PHY that only supports the SDR data interface only utilize the core clk and does not require a phy clk, refer to NANDFCTRL2_PHY chapter of this manual for more information of the specific PHYs.

The core performs clock-domain crossing synchronization between the system and the core clock, and the core clock and phy clock, domains by mean of handshake mechanisms, meta-stabilization and dual port buffers.

The current version of the core requires the clock frequency of clk_sys to be larger or equal to clk_core.

The phy clock of the PHY should be 4 times the core clock and the two clocks should be synchronous and the positive edges must align.

The core features several options to reset or abort the core itself and the NAND flash memory device.

113.6.2 Hard reset

Activating the reset signals to the core resets the core itself, while leaving the connected NAND flash memory device(s) in its current state. The core's reset behavior depends on the GRLIB configuration, section 113.16.2 for more information.

To prevent the risk of double-driving any of the bi-directional signals between the controller and the flash device(s) the controller will set and keep these signals as inputs for the generic *rst_cycles* core clock cycles. This state is indicated by the *RDY* status bit in the *Core status 0 register* is cleared to '0' while the reset is active, which is a few clock cycles more than *rst_cycles*. The controller will ignore any descriptor trigger (*DT* field in the *Descriptor command register*) while *RDY* is '0'.

Depending on which state the flash device is in when the hard reset is activated, it might be needed to reset the flash device before starting the next operation.

113.6.3 Soft reset

Setting the *RST* bit in the *Core control 2 register* activates a soft reset. This will reset the APB registers but nothing else of the core.

It is not possible to combine abort and soft reset in one access. Such a request needs to be split into a sequence of two accesses, including waiting for abort to complete prior to starting a soft reset.

113.6.4 Sending reset command

The core can be used to transmit an ONFI command to reset parts of, or the entire, NAND flash memory device, while leaving the core in its current state.

113.6.5 Abort

The core implements an abort function. This can be useful if the core is in an unknown state, with soft reset as an additional option. When the abort function is triggered, NANDFCTRL2 will safely abort the current operation. If the core, at the time of abort, is processing a descriptor, it will interrupt processing that descriptor and terminate and transfer over the ONFI bus but continue transmitting over the AHB bus. After the abort has been executed, the core will write the final descriptor registers and stop the linked list if in linked list mode. An AMBA interrupt is generated if enabled.

An abort can be triggered in the following ways

- Setting the *ABORT* bit in the *Core control 2 register* to '1'.
- The core detects an invalid command.
- Timeout from the NAND flash memory device.
- AHB error.
- Errors on internal RAM memories.

Similar as described in the Hard reset section 113.6.2, the controller will be set in an abort state while abort is active. The state is indicated with the *RDY* field in the *Core status 0 register* set to '0'. Abort is active a few cycles more than *rst_cycles* and *tRHW* core clock cycles.

Note that it is possible to configure the core incorrectly so it will get stuck. Causes include programming a NAND flash memory device that are in Write Protect mode or selecting an incorrect R/B_n signal that is not going busy.

In the event where the core or the NAND flash memory device has ended up in a stuck state, the user can check that the NAND flash memory device is not busy by reading the *RBSTS field* in the *Core Status 3 register* prior to issuing any command.

Depending on which state the controller and the flash device is in when the abort is triggered, it might be needed to reset the flash device before starting the next operation.

It is not possible to combine abort and soft reset in one access. Such a request needs to be split into a sequence of two accesses, including waiting for abort to complete prior to starting a soft reset.

113.6.6 Stop linked list

It is possible to stop an ongoing linked list by using the Stop Linked list feature. Setting the *STOP_LL field* in the *Core control 2 register* to '1' will complete the current active descriptor and then stop the linked list.

113.7 Programmable Timeout with SEFI handling

The core handles timeouts in the event that a R/B_n signal does not respond as expected.

All commands which use a R/B_n signal to detect when a command has been completed can be monitored using the *Programmable timeout register.* An R/B_n signal will be monitored if it has been selected in the *Descriptor ready/busy select register* and the *Programmable timeout register* is configured to a non-zero value.

When the core issues a command that expects an R/B_n response, the timeout block will load the value in the *Programmable timeout register* and start to count down shortly after the time *tWB* configured in the *Programmable timing 1 register*. The value in this register corresponds to the same number of core clock cycles to be used. If the counter hits 0 an interrupt is issued (the *TO field* in *Core status 1 register* indicates if any timeout occurred and the *Core status 2 register* indicates which R/B_n signal that triggered it).

The controller needs to be reset to a known state by using the abort function after a timeout has occurred. It is also possible that the nand flash device needs to be reset.

The selected R/B_n signal is expected to toggle, meaning that a faulty selected signal will stay static and cause the timeout to trigger indicating an incorrect setup.

The timeout feature is disabled for accesses configured to not wait for R/B (the *SRB field* in the *Descriptor command register* set to '1').

The user can use the digital output signal SEFI_FLAG (one per R/B_n signal) to handle the timeout externally. This is done by setting the corresponding bit in the *Core control 4 register*. Setting any of the SEFI_FLAG bits to '1' will also set *all* bidirectional signals as inputs and drive *all* outputs to '0'. It is needed to trigger an abort of the controller once all the SEFI_FLAG bits have been disabled(*ABORT* bit in the *Core control 2 register* to '1').

113.8 Interrupts

There are several interrupt sources which can generate interrupt. Each source has an enable bit in *Core control 1 register* and a corresponding status bit in *STS1 register.* There is also a main IRQ bit that selects if an interrupt should be generated on the AMBA bus. The interrupt status bits are sticky and are cleared when a '1' is written to corresponding bit. Several of the interrupt sources have a corresponding status bit on descriptor level in the *Descriptor status register.* The interrupt status bits of the *Descriptor status register* are only valid when the controller is configured to Linked List Mode. Several of the interrupt sources also have an additional enable bit in the *Descriptor command register* that can be used to disable interrupts per access. Both the enable bits of *Core control 1 register* and *Descriptor command register* needs to be enabled for an interrupt to be generated.

The interrupt sources are:

IRQ-CERRDL - Correctable error in downlink (*write to* flash device) syncrams.

IRQ-CERRUL - Correctable error in uplink (*read from* flash device) syncrams.

IRQ-UCERRDL - Uncorrectable error in downlink syncrams.

IRQ-UCERRUL - Uncorrectable error in uplink syncrams.

IRQ-STOPLL - Stop linked list.

IRQ-ABORT - Abort.

IRQ-TMOUT - Timeout from timeout block.

IRQ-CMD - Invalid command.

IRQ-DL - An error in DMA Downlink access (AMBA bus read error).

IRQ-UL - An error in DMA Uplink access (AMBA bus write error).

IRQ-ECC - An uncorrectable checksum error from the BCH decoder.

IRQ-DS - A descriptor has finished.

113.9 Error detection and correction

The core uses parametrized BCH encoder(s) and decoder(s) with configurable capability and chunk size of 512 byte or 1024 byte. It supports up to 60-bit correcting capability. Up to 2 blocks with different configurations may be instantiated simultaneously.

For details regarding their implementations, please see thee BCH encoder and BCH decoder chapters.

If error correction is activated, for each READ command the following statistics are collected

- the number of corrected errors in the worst correctable chunk during this READ in field *CEC* in register *Descriptor ECC status*. Note uncorrectable chunks are ignored in this field.
- the total number of corrected errors in this READ in field *CE* in register *Descriptor ECC status*. Note uncorrectable chunks are ignored in this field.
- if any error correction was unsuccessful in field *UE* in register *Descriptor status*.
- in which chunk(s) error correction was unsuccessful in field *ECFAIL* in register *Descriptor status*.

These statistics are cleared each time a new descriptor is triggered.

In addition, an ECC IRQ is issued every time a chunk isn't correctly decoded (if enabled). However, the core does not stop any ongoing access or active linked list independent of the result from the decoder.

113.10 Randomization

The randomization feature can be turned off by either the *rnd* generic, the *RE* field in the *Core control 0 register,* or the *RD* field in the *Descriptor command register*.

If activated, the randomization produces a pseudo-random bitmask based on the page and column address of the data. The periodicity of the randomizer is one nand flash block. The randomization and derandomization are identical.

Since randomization utilize the page data, it requires the row address (*RowAddr* field of *Descriptor row & control register)* to be configured, even if the specific ONFI access does not.

113.10.1 Algorithm

Each page within a block is getting a unique seed and thus creates a unique sequence. The sequence generation is based on a 64-bit starting point and an LSFR with polynomial $x^{64} + x^4 + x^3 + x + 1$.

Should the column be non-zero, then the sequence is fast-forwarded the appropriate number of steps. This ensure that parts of a write can be read, and the device is still able to derandomize it.

113.11 Write Protect

A flash memory device can be set in Write Protect mode using the WP_n pin.

The core controls those pins through the *Core control 3 register*, one bit per channel. Setting any of those bits to '1' will activate the corresponding WP_n pin. If the field is updated while the controller is executing another command, the WP_n pin will be updated after that command is finished. The status can be monitored in the *WP field* of the *Core status 0 register*.

113.12 Wait Array Ready

Some flash devices will drive the R/B_n pin high while an array operation is still ongoing causing the user to start the next array operation too early and illegally.

The core implements a Wait array ready feature to get around this. Configuring the core for a *Read Status* command and enabling the *WARDY* field in the *Descriptor command register* will cause the core to repeatedly send the Read status command until the ARDY bit of the nand flash device status register is read '1'.

113.13 Usage and examples

113.13.1Configuration and start

To use NANDFCTRL2, it has to be instantiated with generic parameters that configure its usage, see section 113.17 for more details. General setup and run-time configuration of the core are done through a number of APB registers and a descriptor which can be placed in the APB area or memory elsewhere. Status is placed both in the descriptor and in the other registers.

Data interface is configured using the *IFSEL* field in the *Core control 0 register*. Configuration of timing mode is done in the *Programmable timing register 0-11* and the *EDO* bit in the *Core control 0 register*. For all accesses, the core decodes and selects the timing to be used from the timing registers and the frequency of the core and phy clocks.

The interrupt mask is configured in the *Core control 1 register* and the Ready/Busy timeout is setup in the *Programmable Timeout 0 register.* Interrupt status and timeout status can be read in the *Core Status 1* and *Core status 2 registers*. There is also a number of descriptor interrupt enable bits in the *Descriptor command register* which can be used to disable interrupts per descriptor access. Note that both interrupt enable bits in *Core control 1 register* and in *Descriptor command register* needs to be set for the core to generate an interrupt. Several of the interrupt status bits are also available on descriptor level in the *Descriptor status register.*

The *Core control 0 register* hold the setting of EDAC enable (*EE* bit), Data randomization enable (*RE* bit), Preserve bad block (*BBM* bit) and Memory Selection (*MSEL* field). In addition Linked List Mode is enabled setting the *LLM* field to '1'.

If the AHB bus is used, the *DD* bit in the *Descriptor command register* must be '0' and the *Descriptor data pointer registers* must point to where data should be fetched or written. If *DD* is set to '1' and the current operation has a write data payload, assure the *Data-out registers* are properly configured. If data is read from the flash memory device it will be stored in the *Data-in registers* (when *DD* is set to '1'). Note that the *DD* field is placed in the descriptor and can be updated per access.

If linked list mode should be used, configure *LLM* to '1' and setup the *Linked list pointer registers* to the first descriptor in memory. Note that the last descriptor in a linked list shall have its *EN* bit set to 0 to stop reading the linked list. This descriptor will not be executed. If register mode should be used, configure the descriptor over APB (*Descriptor registers*).

Warmup cycles are configured in the *Core control 0 register*. The *WW* field configures the number of write cycles and the *WR* field the number of read cycles.

Setup the Descriptor registers *Descriptor target select 0, Descriptor channel select register* and *Descriptor ready/busy select register* for accessing the correct flash memory device(s).

Setup the Row address in the *Descriptor row & control register* for the current access. The *RowAddr* field is used for Row address but is also utilized for commands having an address phase that is not three bytes long. Those commands includes Read ID, Read Unique ID, Read Parameter Page, Get Features, Set features, LUN Set Features, LUN Get Features, Volume Select, ODT Configure, ZQ Calibration Long and ZQ Calibration Short. In the case of using one or two bytes, the lower bytes should be used. Note that if randomization is enabled the row address needs to be set correctly even if the specific access doesn't require it. This is due to the reason that the randomization algorithm utilize the row address.

TAGEN in *Descriptor row & control register* can be enabled if the generic *tag_size* is set to a non-zero value, EDAC is enabled and *SIZE* is set equal to the data area of the selected memory.

Configure the Column address and size in the *Descriptor column & size register*. The *COLADDR* field is used for column address only. The *SIZE* field is used for defining the number of bytes in the data phase. Software does not need to set this field for the commands Set Features, Get Features, Read Status, or Read Status Enhanced their data phases are always the same size. The core also ignores this field if the command issued doesn't have a data phase, as for example BLOCK ERASE.

Setup the *Descriptor command register* including the remaining descriptor-specific configuration fields. Those are described in more detail in the next section 113.13.2.

A Descriptor access is started by setting the *DT* bit in the *Core control 2 register*. If the *EN* bit in the *Descriptor command register* is set the core starts to fetch a descriptor. If the *EN* bit is cleared the descriptor will not be considered valid and will not be executed. The *DA* bit in the *Core Status 0 register* can be monitor if the current access is ongoing. In linked list mode the *DA* bit will stay high until the last descriptor is done.

The *Descriptor status register* holds the *UE* bit which indicates the BCH decoder have one or more uncorrectable errors. The *ECFAIL* field in the same register indicates which chunk(s) that was uncorrectable. The *INV* field in the same register indicates that the last access was invalid. The *Descriptor ECC status register* holds additional information from the BCH decoder including the number of corrected errors in the worst chunk (*CEC* field) and the total number of corrected errors over all chunks (*CE* field).

Non-descriptor accesses includes updating Write Protect, updating sefi_flag, skew control, send soft reset, abort and stop linked list. Updating the WP_N pin is done by changing the corresponding bit the *Core control 3 register.* The WP field in the *Core status 0 register* can be used to monitor if the WP_n value has been updated. Note that it might be a latency from updating WP control until the actual value is updated in case the controller is busy with another access. Updating the sefi_flag pin is done by setting the corresponding bit in the *Core control 4 register*. The *Skew Control Transfer register* and *Skew Control Transfer register* holds configuration that is output to the PHY. A soft reset is initialized by setting the *RST* bit in the *Core control 2 register*. It resets all APB registers of the core.

The *RDY* field in the *Core status 0 register* indicates if the controller is busy in a reset or abort procedure. A descriptor trigger (*DT* field in the *Descriptor command register*) will be ignored while the RDY field is set to '0'.

The *DA* field in the *Core status 0 register* indicates that the controller is executing one or a sequence of descriptors. It is prohibited to update any control signals while *DA* is set to '1'.

The *RBSTAT* field in the *Core Status register 3* can be used to monitor the current value of each R/B_n pin.

113.13.2Configuring a descriptor access

Some ONFI operations can be executed with a NANDFCTRL2 single descriptor access, while other might require a chain of accesses (multiple-access) to achieve the wanted sequence. Refer to section 113.1.2 for which commands that are supported in this revision of the IP. A single-access always starts with a command (*cmd1)* and can be followed by different combinations of address phase, datain or dataout phase, second command and wait for ready/busy depending on the configuration of the *Descriptor command register*. Below are the fields in the register described.

The *cmd1* field holds the first command byte to be executed. This corresponds to the 1st cycle in ONFI 4.0 specification table 90.

The *cmd2* field holds the second command byte to be executed. This corresponds to the 2nd cycle in ONFI 4.0 specification table 90. This field is ignored for commands only having a first command or accesses where the *sc2* field is set.

The *Pre-cache* field. The randomization algorithm uses row address as input. When performing cache operations the row address from a previous access should be used instead of the current. Enabling the *pre-cache* field will configure the controller to use the previous cached row address for its current operation. Pre-cache should be enabled when randomized data is read for a flash device for Read Cache Random, Read Cache Sequential, Read Cache END and Change Read Column commands.

SUB_CMD. Commands starting with *CMD1* = 0x80 or 0x85 will have different address phase and different timing depending on the setting of *SUB_CMD,* see the ONFI 4.0 Specification for more information. The *SUB_CMD* can be configured for a “normal” program operation, Small data move, Change Write Column or Change Row Address. The core ignores this field for all other commands.

Skip address (*SA*) can be used for accesses that should exclude the address phase. One example of usage is the Read Mode command in conjunction with an earlier command with *skip rb* enabled.

Skip data (*SD*) can be used for accesses that should exclude the data phase. Configuring *SIZE* to 0 in the *Descriptor column & size register*.will have the same effect. The following ONFI commands can be executed with or without a data phase: Read Cache End, Read Parameter Page, Copyback Read, Read (0x00, 0x30) Read Multiplane (0x00, 0x35), read Cache Sequential and Copyback Program.

The *sc2* field is used to skip the second command of the access. For some commands it is not possible to explicitly decode if both *CMD1* and *CMD2* should be used, and *SC2* needs to be set specifically. This includes accesses with *CMD1* = 0x80 or 0x85. One example is a page program combined with one Change Write Column, where the first access will be *CMD1* = 0x80 with *SC2* = 1, and the second (and final) access will be *CMD1* = 0x85 and *CMD2* = 0x10. The core ignores this field for commands for all other commands.

The *ED* field can be used to disable EDAC for the current access.

The *RD* field can be used to disable Randomization for the current access.

The *IRQ_DS, IRQ_ECC, IRQ_UERRDL and IRQ_UERRUL* fields can be used to disable descriptor interrupt(s) for the current access.

The *DD* field, DMA Disable, is described in the previous section.

Skip rb *(SRB).* When this field is disabled the core will wait for the selected R/B_n signals to go ‘0’ followed by a ‘1’. If this field is enabled the core will skip wait for any R/B_n signals. In the case of a read operation the core will go back to an idle state and a new access is needed to read the data once the memory device is in a ready state.

The *WARDY* field can be used with a Read status command to loop the command until the ARDY bit is set to ‘1’.

113.13.3First setup example

1. In the case the core is in an unknown state and have not been reset, the user can abort any ongoing operation by setting the *Abort* bit in the *Core control 2 register* and wait for the operation to complete by reading the *Abort* bit in the *Core status 1 register* and the *RDY* bit in the *Core status 0 register*.

2. Send a soft reset by setting the *RST* bit of the *Core control 2 register* to reset all APB registers.

3. Setup Core control registers and Timing registers and Timeout register.

4. Setup Target, Channel and Ready/Busy in Descriptor.
Desired targets and channels are selected using their dedicated registers: *Descriptor target select register*, *Descriptor channel select register* and *Descriptor ready/busy select register.*

5. Issue a RESET command to selected targets using CMD1 = 0xFF.

6. Perform a READ_ID for selected target to scan for NAND flash device.

7. Perform a READ PARAMETER PAGE for selected target.

8. Setup Core Control registers and Timing registers accordingly with information from parameter page.

9. Get/Set Feature (Target/LUN) if needed as part of setup.

10. Make sure the flash memory is not in write protect mode by reading the *WP* fields in the *STS0 register,* otherwise the WP bit can be cleared to deselect Write Protection signal in *CTRL3 register*.

113.13.4General setup example

1. Setup Target, Channel and Ready/Busy in Descriptor.

2. Setup command CMD1 or commands CMD1 & CMD2 to the *DCMD register*, and set the control bits accordingly to usage. Select if an interrupt should be generated when the command is finished and set EN bit to '1'.

3. Setup row address (or just an address) via *DROW register* if needed.

4. Setup column address via *DCOLSIZE register* if needed. Setup size via *DCOLSIZE register* if needed. Commands that reads or writes data needs to setup size.

5. Start execution by writing '1' to *DT* bit in the *CNTRL2 register.*

Once an access has been started software can wait for an interrupt if the core was configured to generate IRQ_DS or monitor STS0 to find out if the descriptor is active.

113.13.5Changing data interface

Configuration of the nand flash device(s) and the controller to the NV-DDR2 data interface requires a few steps.

1. Configure NANDFCTRL2 to SDR data interface using *IFSEL* and the *Programmable timing registers.*

2. Configure NANDFCTRL2 for a Set Feature command which configures the nand flash device(s) to NV-DDR2 data interface.

3. Configure NANDFCTRL2 for NV-DDR2 data interface using *IFSEL* and the *Programmable timing registers.*

113.13.6Write chunk example

It is possible to write to the data area with one chunk per access. It is then necessary to loop over all chunks and write them one by one using PAGE_PROGRAM command and multiple CHANGE_WRITE_COLUMN commands interleaved.

1st chunk written should have PAGE_PROGRAM for CMD1 and SC2 set to '1'. Size should be chunk size and column address set to aligned column address for the 1st chunk. Chunk 2 to n-1 should have CMD1 set to CHANGE_WRITE_COLUMN. Sub-command SUB_CMD should be set to Change_write_column. Size should be set to chunk size and column address set to aligned column address for the selected chunk. Last chunk should have CMD1 set to CHANGE_WRITE_COLUMN and CMD2 set to 0x10 (2nd cycle of PAGE PROGRAM), SC2 cleared to '0' and sub-command SUB_CMD should be set to Change_write_column. Size should be set to chunk size and column address set to aligned column address for the last chunk.

This setup creates a PAGE_PROGRAM with correct cmd1 cycle and cmd2 cycle with CHANGE_WRITE_COLUMNS inserted in between with cmd1 cycle.

If any write to spare area has to be performed after data area access, the last chunk must set SC2 to prevent a write access to start.

113.13.7Read chunk example

It is possible to read from the data area with one chunk per access. It is then necessary to loop over all chunks and read them one by one.

1st chunk read should have PAGE_READ command setup in CMD1 and CMD2. Size should be chunk size. Chunk 2 to n should have CMD1 and CMD2 set to CHANGE_READ_COLUMN. Size should be set to chunk size and column address set to aligned column address for the selected chunk.

This setup creates a PAGE_READ with correct CMD1 cycle and CMD2 cycle followed by CHANGE_READ_COLUMNS cycles.

113.13.8Write data example

It is possible to write to the whole data portion of a page with one access using PAGE_PROGRAM command and size equal to whole data area. NANDFCTRL2 will then loop over all chunks and write them one by one using PAGE_PROGRAM command with interleaved CHANGE WRITE COLUMN commands.

113.13.9Read data example

It is possible to read from the whole data portion of a page with one access using PAGE_READ command and size equal to whole data area. NANDFCTRL2 will then loop over all chunks and read them one by one using page read command and sequences of CHANGE_READ_COLUMN commands.

113.13.10Write spare example

A write access to the spare area is done via CHANGE_WRITE_COLUMN and size set to desired size.This presumes that the data area has been accessed before and that the 2nd cycle of PAGE_PROGRAM has not been issued. The 2nd cycle of PAGE_PROGRAM can be set as CMD2.

113.13.11Read spare example

A read access from the spare area is done via CHANGE_READ_COLUMN and size set to desired size. This presumes that a READ_PROGRAM has been issued before.

113.14 Registers

*Table 2169.*NANDFCTRL2 registers

APB address offset	Register
0x000	Core control 0 register
0x004	Core control 1 register
0x008	Core control 2 register
0x00c	Core control 3 register
0x010	Core control 4 register
0x020	Core status 0 register
0x024	Core status 1 register
0x028	Core status 2 register
0x02c	Core status 3 register
0x038	Capability 0 register
0x03c	Capability 1 register
0x040	Capability 2 register
0x044	Capability 3 register
0x048	Capability 4 register
0x04c	Capability 5 register
0x050	Programmable timing 0 register
0x054	Programmable timing 1 register
0x058	Programmable timing 2 register
0x05c	Programmable timing 3 register
0x060	Programmable timing 4 register
0x064	Programmable timing 5 register
0x068	Programmable timing 6 register
0x06c	Programmable timing 7 register
0x070	Programmable timing 8 register
0x074	Programmable timing 9 register
0x078	Programmable timing 10 register
0x07c	Programmable timing 11 register
0x0c0	Skew Control Transfer register
0x0c4	Skew Control Receive register
0x0d0	Programmable timeout 0 register
0x150	Linked list pointer low register
0x154	Reserved
0x160	Descriptor command register
0x164	Descriptor target select 0 register
0x168	Descriptor target select 1 register
0x16c	Descriptor channel select register
0x170	Descriptor ready/busy select register
0x174	Descriptor row & control register
0x178	Descriptor column & size register
0x17c	Descriptor status register
0x180	Descriptor ECC status register

*Table 2169.*NANDFCTRL2 registers

APB address offset	Register
0x190	Descriptor data pointer low register
0x194	Reserved
0x1e0	Data-in 0 register
0x1e4	Data-in 1 register
0x1e8	Data-in 2 register
0x1ec	Data-in 3 register
0x1f0	Data-out 0 register
0x1f4	Data-out 1 register
0x1f8	Data-out 2 register
0x1fc	Data-out 3 register

113.14.1Core control 0 register

*Table 2170.*0x000 - CTRL0 - Core control 0 register

31	22	21	19	18	16
RESERVED		WR		WW	
0		0		0	
r		rw		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED		EE	RE	RES	LLM	BBM	RES	EDO	IFSEL		MSEL		CM	RESERVED	
0		1	1	0	0	1	0	0	0		0		0	0	
r		rw	rw	r	rw	rw	r	rw	rw		rw		rw	r	

31:22 Reserved. This field is reserved for future use and should be set to 0.

21:19 Warmup read cycles (WR)
Controls the number of warmup cycles to be executed during a flash read access.

18:16 Warmup write cycles (WW)
Controls the number of warmup cycles to be executed during a flash write access.

15:14 Reserved. This field is reserved for future use and should be set to 0.

13 EDAC enable (EE)
0: Bypass EDAC (ECC).
1: Enable EDAC (ECC). The EDAC disable bit in the descriptor command register needs to be set to 0 to utilize this control bit.

12 Data randomization enable (RE)
0: Data randomization bypassed.
1: Data randomization enabled. The Data randomization disable bit in the descriptor command register needs to be set to 0 to utilize this control bit.

11 Reserved. This field is reserved for future use and should be set to 0.

10 Linked list mode enable (LLM)
0: Inactive. The core is configured to Register Mode and descriptors are read from APB descriptor registers.
1: Active. The core is configured to Linked List Mode and descriptors are read from memory over DMA (AHB) in linked lists.

9 Preserve Bad Block Marking (BBM)
0: Use first 2 bytes in spare area.
1: Do not use first 2 bytes in spare area.

8 Reserved. This field is reserved for future use and should be set to 0.

7 Extended Data Output (EDO)
0: EDO off.
1: EDO on.

6:5 Data Interface select (IFSEL).
0: Use SDR data interface.
1: Not valid
2: Use NV-DDR2 / NV-DDR3 data interface.
3: Not valid

4:3 Memory select (MSEL)
Select which memory the core should be setup to comply with according to *Capability 2, 3 and 4 registers*.
Note that this value will also indirectly choose which ECC block to be used in accordance with the values in the *Capability 1 register*.
Valid values: 0,1,2.

2 Channel mode (CM)
0: Normal operation.
1: Only bit 0 and 8 will be valid for ALE, CLE and IO0-7.

1:0 Reserved. This field is reserved for future use and should be set to 0.

113.14.2Core control 1 register

*Table 2171.*0x004 - CTRL1 - Core control 1 register

31	16
RESERVED	
0	
r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED		IRQ CERRDL	IRQ CERRUL	IRQ UCERRDL	IRQ UCERRUL	IRQ STOPLL	IRQ ABORT	RES	IRQ TMOUT	IRQ CMD	IRQ DL	IRQ UL	IRQ ECC	IRQ DS	IRQ
0		0	0	0	0	0	0	0	0	0	0	0	0	0	0
r		rw	rw	rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw	rw

31:14 Reserved. This field is reserved for future use and should be set to 0.

13 Correctable error in downlink (*write to* flash device) syncrams (IRQ-CERRDL)
0: Interrupt is disabled.
1: Interrupt is enabled.

12 Correctable error in uplink (*read from* flash device) syncrams (IRQ-CERRUL)
0: Interrupt is disabled.
1: Interrupt is enabled.

11 Uncorrectable error in downlink syncrams (IRQ-UCERRDL)
0: Interrupt is disabled.
1: Interrupt is enabled. An interrupt is generated if there is an uncorrectable error in any of the downlink srams, if both this field and the *irq_uerrdl* field in *Descriptor command register* are enabled.

10 Uncorrectable error in uplink syncrams (IRQ-UCERRUL)
0: Interrupt is disabled.
1: Interrupt is enabled. An interrupt is generated if there is an uncorrectable error in any of the downlink srams, if both this field and the *irq_uerrul* field in *Descriptor command register* are enabled.

9 Stop linked list interrupt (IRQ-STOPLL)
0: Interrupt is disabled.
1: Interrupt is enabled.

8 Abort interrupt (IRQ-ABORT)
0: Interrupt is disabled.
1: Interrupt is enabled.

7 Reserved. This field is reserved for future use and should be set to 0.

6 Ready/Busy timeout interrupt (IRQ-TMOUT)
0: Interrupt is disabled for timeout on all Ready/Busy signals.
1: Interrupt is enabled for timeout on all Ready/Busy signals.
This interrupt is issued when there is a Ready/Busy timeout for the current transaction on any of the selected RB_n signals.

5 Invalid command interrupt enable (IRQ-CMD)
0: Interrupt is disabled.
1: Interrupt is enabled.
This interrupt is issued when there is an invalid descriptor is decoded.

4 DMA read error interrupt (IRQ-DL)
0: Interrupt is disabled.
1: Interrupt is enabled.
This interrupt is issued when there is an error in DMA controller downlink (*read from* AMBA bus).

3 DMA write error interrupt (IRQ-UL)
0: Interrupt is disabled.
1: Interrupt is enabled.
This interrupt is issued when there is an error in DMA uplink (*write to* AMBA bus).

2 ECC interrupt (IRQ-ECC)
0: Interrupt is disabled.
1: Interrupt is enabled.
This interrupt is issued when the BCH decoder detects an uncorrectable chunk, if both this field and the *irq_ecc* field in *Descriptor command register* are enabled.

*Table 2171.*0x004 - CTRL1 - Core control 1 register

1	Descriptor interrupt (IRQ-DS). 0: Interrupt is disabled. 1: Interrupt is enabled. An interrupt is generated when a descriptor has finished executing, if both this field and the *irq_ds* field in *Descriptor command register* are enabled.
0	Main Interrupt enable (IRQ) If set to '1' an AMBA interrupt will be generated if one or more of the interrupt sources have been triggered.

113.14.3Core control 2 register

*Table 2172.*0x008 - CTRL2 - Core control 2 register

31	16
RESERVED	
0	
r	

15	4	3	2	1	0
RESERVED		STOP_LL	ABORT	DT	RST
0		-	-	-	-
r		w	w	w	w

31:4	Reserved. This field is reserved for future use and should be set to 0.
3	Stop Linked List (STOP_LL) Stop an ongoing linked list. The current active descriptor will complete and then the linked list will be stopped.
2	Abort (ABORT) Abort any ongoing execution in the core and sets it in an idle state. The status of an ongoing abort operation can be monitored in the *Core status 1 register*. It is not possible to combine abort and soft reset in one access. This bit is cleared automatically when command has been executed.
1	Descriptor trigger (DT) 1: Trigger descriptor command. The core starts fetching a descriptor and starts executing it if the EN bit in the descriptor command register is set. The descriptor is fetched from the AHB memory or the descriptor APB registers depending on the setting of the *LLM field* in the *core control 0 register.* This bit is cleared automatically when command has been executed.
0	Software reset (RST) 1: All APB registers are reset to their default state. Reset of the registers is completed in the next clock cycle. It is not possible to combine abort and soft reset in one access. This bit is cleared automatically when a reset command has been executed.

113.14.4Core control 3 register

*Table 2173.*0x00c - CTRL3 - Core control 3 register

31	16
WP	
0xFFFF	
rw	

15	0
RESERVED	
0	
r	

31:16 Write protect (WP)
One bit per channel. Channel 0 corresponds to bit 0.
When one or more of these bits is set to '1' the core puts the flash memory devices in write protect mode by asserting the corresponding WP_n output signal(s) to '0'. In write protect mode, the memories won't respond to PROGRAM or ERASE commands. If the core is active when this field is updated there is a delay before the actual write protect signal goes low. Software can use the *WP* field in the *Core status 0 register* to see if the output has changed value.

Note that the polarity of this register field is the opposite compared to the WP_N pin.

15:0 Reserved. This field is reserved for future use and should be set to 0.

113.14.5Core control 4 register

*Table 2174.*0x010 - CTRL4 - Core control 4 register

31	16
SEFI_FLAG	
0	
rw	

15	0
SEFI_FLAG	
0	
rw	

31:0 SEFI flag (SEFI_FLAG)
One bit per SEFI_FLAG output signal. SEFI_FLAG signal 0 corresponds to bit 0.
0: Sets SEFI_FLAG output signal to low.
1: Sets SEFI_FLAG output signal to high. Enabling any sefi_flag will set all flash output signals to logic '0' and bi-directional to inputs.

113.14.6Core status 0 register

*Table 2175.*0x020 - STS0 - Core status 0 register

31	16
WP	
0xFFFF	
r	

15	4	3	2	1	0
RESERVED		DA	RESERVED		RDY
0		0	0		0*
r		r	r		r

31:16 Write protect status (WP)
Shows if the flash memory devices are in write protect mode or not. One bit per channel. Also see the description of the WP field in the Core control 3 register.
0: Not in write protect mode.
1: In write protect mode.

15:4 Reserved. This field is reserved for future use and should be set to 0.

3 Descriptor active (DA)
1: Descriptor active. Indicates that the descriptor is active and a command sequence is ongoing.
0: No access is ongoing via descriptors and the core is in an idle state.
For a descriptor chain, i.e. when using linked list mode, the DA signal stays active until the last descriptor is executed and no more RAM accesses are performed by the DMA on that chain.

2:1 Reserved. This field is reserved for future use and should be set to 0.

0 Ready (RDY)
Shows if the controller is ready to start a descriptor after reset or abort of the controller. The RDY signal stays ‘0’ during active reset and active abort.
In reset RDY goes ‘1’ a few cycles after after the time set by the generic *(rst_cycles*) * core clock cycles.
In abort RDY goes ‘1’ a few cycles after after the time set by the generic *(rst_cycles*) * core clock cycles or the timing parameter *tRHW*, whichever is largest.

113.14.7Core status 1 register

*Table 2176.*0x024 - STS1 - Core status 1 register

31	16
RESERVED	
0	
r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED		CERRDL	CERRUL	UCERRDL	UCERRUL	STOPLL	ABORT	RES	TMOUT	CMD	DL	UL	ECC	DS	IRQ
0		0	0	0	0	0	0	0	0	0	0	0	0	0	0
r		wc	wc	wc	wc	wc	wc	r	wc	wc	wc	wc	wc	wc	wc

31:14 Reserved.

13 Correctable error in downlink syncrams (CERRDL)
This is set when there is an correctable error in any of the downlink syncrams.

12 Correctable error in uplink syncrams (CERRUL)
This is set when there is an correctable error in any of the uplink syncrams.

11 Uncorrectable error in downlink syncrams (UCERRDL)
This is set when there is an ucorrectable error in any of the downlink syncrams.

10 Uncorrectable error in uplink syncrams (UCERRUL)
This is set when there is an ucorrectable error in any of the upink syncrams.

9 Stop linked list interrupt (STOPLL)
This is set when an Stop Linked list operation is completed.

8 Abort interrupt (ABORT)
This is set when an abort operation have completed. An abort will always trigger a stop linked list, i.e. the STOPLL interrupt will also be set.

7 Reserved.

6 Ready/Busy timeout (TMOUT)
This is set when there is an Ready/Busy timeout on any of the signals. Read *Core status 2 register* to see which RB_n input signal that triggered this signal.

5 Invalid command (CMD)
This is set when there is an invalid descriptor is being decoded.

4 DMA read error (DL).
This is set when there is an error in DMA Downlink.

3 DMA write error (UL).
This is set when there is an error in DMA Uplink.

2 ECC error (ECC).
This is set when the BCH decoder detects an uncorrectable chunk.

1 Descriptor finished (DS).
This is set when the descriptor is finished.

0 Main Interrupt (IRQ)
This is set when an AMBA interrupt have been issued.

113.14.8Core status 2 register

*Table 2177.*0x028 - STS2 - Core status 2 register

31	16
TMOUT	
0	
wc	

15	0
TMOUT	
0	
wc	

31:0 Ready/Busy timeout (TMOUT).
This is set when there is an Ready/Busy timeout. There is one bit per RB_n signal, i.e. Ready/Busy Tmout signal 0 corresponds to bit 0.

113.14.9Core status 3 register

*Table 2178.*0x02c - STS3 - Core status 3 register

31	16
RBSTS (31:16)	
0	
r	

15	0
RBSTS (15:0)	
0	
r	

31:0 Ready/Busy Status (RBSTS) for RB_n signals 0-31
The register shows the current status for each R/B_n input signal.
The least significant bit in this field corresponds to the lowest ready busy signal (RB_n(0)) etc.
The actual number of bits implemented equals the *nrofrb* field +1 in the *Capability 0 register*.

113.14.10Capability 0 register

*Table 2179.*0x038 - CAP0 - Capability 0 register

31 25	24 23	22 20	19	18	17 16
RESERVED	MEMSUP	FT	RES	RND	NECC
0	*	*	0	*	*
r	r	r	r	r	r

15 11	10 6	5 0
NRB	NCHAN	NTARGS
*	*	*
r	r	r

31:25 Reserved.

24:23 Number of memories supported (MEMSUP)
Determined by the number of generics *mem0_data*, *mem1_data* and *mem2_data* set to a non-zero value.
Value: 0-3.

22:20 Fault-tolerance support (FT)
Determined by the generic *ft*.

19 Reserved.

18 Randomization supported (RND)
Determined by the generic *rnd*.

17:16 Number of ECC blocks supported (NECC)
Determined by the number of generics *ecc0_cap* and *ecc1_cap* set to a non-zero value.
Value: 0-2.

15:11 Number of RB_n signals (NRB)
Determined by the generic *nrofrb*-1, i.e. the implemented number of RB_n signals are NRB +1.
Value: 0-31.

10:6 Number of channels (NCHAN)
Determined by the generic *nrofch* -1, i.e. the implemented number of channels are NCHAN+1.
Value: 0-15.

5:0 Number of targets (NTARGS) - This field indicates how many individual targets the core can access, i.e. number of chip enable signals.
Determined by the generic *nrofce* -1, i.e. the implemented number of CE_n signals are NTARGS+1.
Value: 0-63.

113.14.11Capability 1 register

*Table 2180.*0x03c - CAP1 - Capability 1 register

31 26	25 21	20 16
E1CAP	E1CHUNK	E1GF
*	*	*
r	r	r

15 10	9 5	4 0
E0CAP	E0CHUNK	E0GF
*	*	*
r	r	r

31:26 Capability for ECC1 (E1CAP)
Determined by the generic *ecc1_cap*.

25:21 Chunk size for ECC1 (E1CHUNK)
The value is the base 2 logarithm of the generic *ecc1_chunk*.

23:16 GF size for ECC1 (E1GF)
Determined by the generic *ecc1_gfsize*.

15:10 Capability for ECC0 (E0CAP)
Determined by the generic *ecc0_cap*.

9:5 Chunk size for ECC0 (E0Chunk)
The value is the base 2 logarithm of the generic *ecc0_chunk*.

4:0 GF size for ECC0 (E0GF)
Determined by the generic *ecc0_gfsize*.

113.14.12Capability 2 register

*Table 2181.*0x040 - CAP2 - Capability 2 register

31	30 29	28 16
M0SEL	RESERVED	M0SPARE
*	0	*
r	r	r

15 0
M0DATA
*
r

31 Memory 0 ECC Selection (M0SEL)
Determined by the generic *mem0_ecc_sel*.

30:29 Reserved.

28:16 Memory 0 spare size in bytes (M0SPARE)
Determined by the generic *mem0_spare*.

15:0 Memory 0 data size in bytes (M0DATA)
Determined by the generic *mem0_data*.

Frontgrade Gaisler AB
Kungsgatan | SE-411 19 | Goteborg | Sweden
+46 31 7758650 | frontgrade.com/gaisler

113.14.13Capability 3 register

*Table 2182.*0x044 - CAP3 - Capability 3 register

31	30 29	28 16
M1SEL	RESERVED	M1SPARE
*	0	*
r	r	r

15 0
M1DATA
*
r

31 Memory 1 ECC selection (M1SEL).
Determined by the generic *mem1_ecc_sel*.

30:29 Reserved.

28:16 Memory 1 spare size in bytes (M1SPARE)
Determined by the generic *mem1_spare*.

15:0 Memory 1 data size in bytes (M1DATA)
Determined by the generic *mem1_data*.

113.14.14Capability 4 register

*Table 2183.*0x048 - CAP4 - Capability 4 register

31	30 29	28 27 16
M2SEL	RESERVED	M2SPARE
*	0	*
r	r	r

15 0
M2DATA
*
r

31 Memory 2 ECC Selection (M2SEL)
Determined by the generic *mem2_ecc_sel*.

30:29 Reserved.

28:16 Memory 2 spare size in bytes (M2SPARE)
Determined by the generic *mem2_spare*.

15:0 Memory 2 data size in bytes (M2DATA)
Determined by the generic *mem2_data*.

113.14.15Capability 5 register

*Table 2184.*0x04c - CAP5 - Capability 5 register

31	16
RESERVED	
0	
r	

15	14	13	8	7	0
RESERVED		TAG_SIZE		RST_CYCLES	
0		*		*	
r		r		r	

31:14 Reserved

13:8 Tag size (TAG_SIZE)
Determined by the generic *tag_size*.

7:0 Reset cycles (RST_CYCLES)
Determined by the generic *rst_cycles*.

113.14.16Programmable timing 0 register

*Table 2185.*0x050 - TME0 - Programmable timing 0 register

31	16
tCS	
0	
rw	

15	0
tWW	
0	
rw	

31:16 CE_n setup time (tCS)
SDR: Length of tCS in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).
NV-DDR2/3: Length of tCS in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

15:0 WP_n transition to WE low (tWW)
SDR: Length of tWW in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).
NV-DDR2/3: Length of tWW in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

113.14.17Programmable timing 1 register

*Table 2186.*0x054 - TME1 - Programmable timing 1 register

31	16
tRR	
0	
rw	

15	0
tWB	
0	
rw	

31:16 Ready to RE_n low (tRR)
SDR: Length of tRR in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).
NV-DDR2/3: Length of tRR in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

15:0 WE_n high to SR[6] low (tWB)
SDR: Length of tWB in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).
NV-DDR2/3: Length of tWB in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

113.14.18Programmable timing 2 register

*Table 2187.*0x058 - TME2 - Programmable timing 2 register

31	16
tRHW	
0	
rw	

15	0
tWHR	
0	
rw	

31:16 RE_n high to WE_n low (tRHW)
SDR: Length of tRHW in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).
NV-DDR2/3: Length of tRHW in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

15:0 WE_n high to RE_n low (tWHR)
SDR: Length of tWHR in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).
NV-DDR2/3: Length of tWHR in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

113.14.19Programmable timing 3 register

*Table 2188.*0x05c - TME3 - Programmable timing 3 register

31	16
tADL	
0	
rw	

15	0
tCCS	
0	
rw	

31:16 ALE to data loading time (tADL)
SDR: Length of tADL in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).
NV-DDR2/3: Length of tADL in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

15:0 Change Column setup time (tCCS)
SDR: Length of tCCS in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).
NV-DDR2/3: Length of tCCS in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

113.14.20Programmable timing 4 register

*Table 2189.*0x060 - TME4 - Programmable timing 4 register

31	16
tREH	
0	
rw	

15	0
tRP	
0	
rw	

15:0 RE_n high hold time (tREH)
SDR: Length of tREH in **core clock cycles** minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).
NV-DDR2/3: When configured to NV-DDR2/3 this parameter is used inside the PHY and not the core itself. It should be configured to the length of tREH in **phy clock cycles** minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

31:16 RE_n pulse width (tRP)
SDR: Length of tRP in **core clock cycles** minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).
NV-DDR2/3: When configured to NV-DDR2/3 this parameter is used inside the PHY and not the core itself. It should be configured to the length of tRP in **phy clock cycles** minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

113.14.21 Programmable timing 5 register

Table 2190. 0x064 - TME5 - Programmable timing 5 register

31	16
tWH	
0	
rw	

15	0
tWP	
0	
rw	

31:16 WE_n high hold time (tWH)
SDR: Length of tWH in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).
NV-DDR2/3: Length of tWH in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

15:0 WE_n pulse width (tWP)
SDR: Length of tWP in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).
NV-DDR2/3: Length of tWP in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

113.14.22 Programmable timing 6 register

Table 2191. 0x068 - TME6 - Programmable timing 6 register

31	16
tCALS	
0	
rw	

15	0
tCSD	
0	
rw	

31:16 CLE and ALE setup time (tCALS)
SDR: Not used.
NV-DDR2/3: Length of tCALS in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

15:0 ALE, CLE, WE_n hold time from CE_n high (tCSD)
SDR: Not used.
NV-DDR2/3: Length of tCSD in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

113.14.23Programmable timing 7 register

*Table 2192.*0x06c - TME7 - Programmable timing 7 register

31	16
tCHZ	
0	
rw	

15	0
tWPRE	
0	
rw	

31:16 CE_n high to output Hi-Z (tCHZ)
SDR: Not used.
NV-DDR2/3: Length of tCHZ in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

15:0 DQS write preamble (tWPRE)
SDR: Not used.
NV-DDR2/3: Length of tWPRE in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

113.14.24Programmable timing 8 register

*Table 2193.*0x070 - TME8 - Programmable timing 8 register

31	16
tRPST	
0	
rw	

15	0
tDQSRH	
0	
rw	

31:16 Read postamble (tRPST)
SDR: Not used.
NV-DDR2/3: Length of tRPST in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

15:0 DQS hold time after (RE_n low or RE_t/RE_c crosspoint) (tDQSRH)
SDR: Not used.
NV-DDR2/3: Length of tDQSRH in core clock cycles. **Note that this parameter is equal to the number of clock cycles and not minus as other timing parameters.**

113.14.25Programmable timing 9 register

*Table 2194.*0x074 - TME9 - Programmable timing 9 register

31	16
tWPST	
0	
rw	

15	0
tWPSTH	
0	
rw	

31:16 DQS write postamble (tWPST)
SDR: Not used.
NV-DDR2/3: Length of tWPST in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

15:0 DQS write postamble hold time (tWPSTH)
SDR: Not used.
NV-DDR2/3: Length of tWPSTH in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

113.14.26Programmable timing 10 register

*Table 2195.*0x078 - TME10 - Programmable timing 10 register

31	16
tDQSL	
0	
rw	

15	0
tDQSH	
0	
rw	

31:16 DQS low level width (tDQSL)
SDR: Not used.
NV-DDR2/3: When configured to NV-DDR2/3 this parameter is used inside the PHY and not the core itself. It should be configured to the length of tDQSL in **phy clock cycles** minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

15:0 DQS low high width (tDQSH)
SDR: Not used.
NV-DDR2/3: When configured to NV-DDR2/3 this parameter is used inside the PHY and not the core itself. It should be configured to the length of tDQSH in **phy clock cycles** minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

113.14.27Programmable timing 11 register

*Table 2196.*0x07c - TME11 - Programmable timing 11 register

31	16
tCS_VOL	
0	
rw	

15	0
tVDLY	
0	
rw	

31:16 CE_n setup time for Volume select command (tCS_VOL)
SDR: Length of tCS in core clock cycles minus one for the Volume select command (Setting this field to 9 clock cycles will give a time of 10 clock cycles).
NV-DDR2/3: Length of tCS in core clock cycles minus one for the Volume select command (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

15:0 tVDLY (tVDLY)
Delay prior to issuing the next command after a new Volume is selected using the Volume Select command.
SDR: Length of tVDLY in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).
NV-DDR2/3: Length of tVDLY in core clock cycles minus one (Setting this field to 9 clock cycles will give a time of 10 clock cycles).

113.14.28Skew Control Transfer register

*Table 2197.*0x0c0 - TXSKEW0 - Skew Control Transfer register

31 30	29 27	26 24	23 21	20 18	17 16
Reserved	TXRE	TXDQS	TXD7	TXD6	TXD5
0	0	0	0	0	0
r	rw	rw	rw	rw	rw

15	14 12	11 9	8 6	5 3	2 0
TXD5	TXD4	TXD3	TXD2	TXD1	TXD0
0	0	0	0	0	0
rw	rw	rw	rw	rw	rw

31:30 Reserved. This field is reserved for future use and should be set to 0.

29:27 Skew transfer RE (TXRE)
Control signal to be connected to PHY.

26:24 Skew transfer DQS (TXDQS)
Control signal to be connected to PHY

23:21 Skew transfer Datao[7] (TXD7)
Control signal to be connected to PHY

20:18 Skew transfer Datao[6] (TXD6)
Control signal to be connected to PHY

17:15 Skew transfer Datao[5] (TXD5)
Control signal to be connected to PHY

14:12 Skew transfer Datao[4] (TXD4)
Control signal to be connected to PHY

11:9 Skew transfer Datao[3] (TXD3)
Control signal to be connected to PHY

8:6 Skew transfer Datao[2] (TXD2)
Control signal to be connected to PHY

5:3 Skew transfer Datao[1] (TXD1)
Control signal to be connected to PHY

2:0 Skew transfer Datao[0] (TXD0)
Control signal to be connected to PHY

113.14.29Skew Control Receive register

*Table 2198.*0x0c4 - RXSKEW0 - Skew control receive register

31 30 29 27	26 24	23 21	20 18	17 16
Reserved	RXDQS	RXD7	RXD6	RXD5
0	0	0	0	0
r	rw	rw	rw	rw

15	14 12	11 9	8 6	5 3	2 0
RXD5	RXD4	RXD3	RXD2	RXD1	RXD0
0	0	0	0	0	0
rw	rw	rw	rw	rw	rw

31:27 Reserved. This field is reserved for future use and should be set to 0.

26:24 Skew receive RE (RXDQS)
Control signal to be connected to PHY.

23:21 Skew receive Datai[7] (RXD7)
Control signal to be connected to PHY.

20:18 Skew receive Datai[6] (RXD6)
Control signal to be connected to PHY.

17:15 Skew receive Datai[5] (RXD5)
Control signal to be connected to PHY.

14:12 Skew receive Datai[4] (RXD4)
Control signal to be connected to PHY.

11:9 Skew receive Datai[3] (RXD3)
Control signal to be connected to PHY.

8:6 Skew receive Datai[2] (RXD2)
Control signal to be connected to PHY.

5:3 Skew receive Datai[1] (RXD1)
Control signal to be connected to PHY.

2:0 Skew receive Datai[0] (RXD0)
Control signal to be connected to PHY.

113.14.30Timeout 0 registers

*Table 2199.*0x0d0 - Timeout 0 registers

31 16
TIMEOUT (31:16)
0
rw

15 0
TIMEOUT (15:0)
0
rw

31:0 Timeout (TIMEOUT) registers for the R/B_n signals.
Timeout starts to count after the time twb. If no R/B rising edge is detected within the set number of core clk cycles an Ready/Busy timeout interrupt (see *TMOUT* field in the *Core status 1 register*) will be generated. Setting this field to 0 disables the timout feature.

113.14.31 Linked list pointer low register

Table 2200. 0x150 - LLPL - Linked list pointer low register

31	16
LINKED LIST POINTER (31:16)	
0	
rw	

15	0
LINKED LIST POINTER (15:0)	
0	
rw	

31:0 Linked list pointer (LINKED LIST POINTER) (31:0)

113.14.32 Descriptor command register

Table 2201. 0x160 - DCMD - Descriptor command register

31	24	23	16
CMD2		CMD1	
0		0	
rw		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ_UERRDL	IRQ_UERRUL	IRQ_ECC	PRECH	DD	ED	RD	SUB_CMD		SRB	SA	SD	SC2	WARDY	IRQ_DS	EN
0	0	0	0	0	0	0	0		0	0	0	0	0	0	0
rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

31:24 Second command phase (CMD2)
If the command to execute is a two byte (two phase) command then software should write the second byte of the command to this field. This field corresponds to the 2nd cycle in the ONFI 4.0 specification table 90.
The core ignores this field for commands that only have one command phase, or if SC2 is set.

23:16 First command phase (CMD1)
Software should write this field with the first byte of the command to execute. This field corresponds to the 1st cycle in the ONFI 4.0 specification table 90.

15 Descriptor uncorrectable error in downlink (IRQ_UERRDL)
This field should be set together with the corresponding field in the *Core control 1 register*.

14 Descriptor (IRQ_UERRUL)
This field should be set together with the corresponding field in the *Core control 1 register*.

13 Descriptor (IRQ_ECC)
This field should be set together with the corresponding field in the *Core control 1 register*.

12 Pre-cache (PRECH)
When enabled the controller use the setting of the previous command as input to the randomization block. This should be used for cache read operations when randomization is enabled.

11 Data DMA disable (DD)
0: enable data read /write over DMA.
1: disable data read/write over DMA (i.e. use data-in and data-out registers).
Selects if data is read via the Data-in registers or data is written via the Data-out registers via APB or via AHB DMA master.
EDAC and Randomization is not supported when DD = ‘1’.

10 EDAC disable (ED)
0: the *EE* bit in the *Core control register* 0 is used as specified in its description.
1: the *EE* bit in the *Core control register* 0 is ignored and EDAC is bypassed for the current descriptor.

Table 2201. 0x160 - DCMD - Descriptor command register

9	Data Randomization disable (RD) 0: the *RE* bit in the *Core control register* 0 is used as specified in its description. 1: the *RE* bit in the *Core control register* 0 is ignored and data randomization is bypassed for the current descriptor.
8:7	Sub command (SUB_CMD) This field is used to specify how an access with CMD1 = 0x80 or 0x85 should be executed. The different settings will use different length of the address phase (2 or 5 bytes) and use different timing parameters (tCCS or tADL). 0: Ordinary usage of CMD1 (all different PAGE PROGRAM commands). 1: Small Data Move. 2: Change Write Column. 3: Change Row Address.
6	Skip Ready/Busy (SRB) 0: The controller waits for the selected R/B_n signals to go low and high. 1: The controller do not wait for the R/B_n signal. In the case of a read operation that triggers the memory device to go busy the controller will go to an idle state and not read any data. A new access is required to read the data once the memory device is in a ready state.
5	Skip Address (SA) 0: The address phase will be included in the descriptor access. 1: The address phase will be excluded in the descriptor access.
4	Skip Data (SD) 0: The data phase will be included in the descriptor access. 1: The data phase will be excluded in the descriptor access.
3	Skip second command phase (SC2) For some command (cmd1 = 0x80 or 0x85) it is not possible to decode if both cmd1 and cmd2 shall be used and it is necessary to explicitly exclude cmd2 using this bit.
2	Wait ARDY (WARDY) When enabled together with a ONFI Read Status command, the controller will loop Read Status to the flash device until the ARDY bit is set high. This field should must be set to ‘0’ for all other ONFI commands.
1	Descriptor interrupt (IRQ_DS) This field should be set together with the corresponding field in the *Core control 1 register*.
0	Descriptor Enable (EN) NANDFCTRL2 will execute the descriptor when the descriptor is triggered by setting the *DT* bit of the *Core Control 2 register* if this bit is set, otherwise the descriptor is not valid and will not be executed. 0: The descriptor is invalid and is disregarded by the core. 1: The descriptor is valid and enabled

113.14.33Descriptor target select 0 register

*Table 2202.*0x164 - DTARSEL0 - Descriptor target 0 select register

31	16
TARGSEL(31:16)	
0	
rw	

15	0
TARGSEL(15:0)	
0	
rw	

31:0 Target select (TARGSEL) for target 0-31
The core uses this field to select from which targets to send the command to (which chip enable (CE_n) signals to assert).
The least significant bit in this field corresponds to the first target (CE_n(0)) etc.
The actual number of bits implemented equals the *ntargs* field + 1 in the *Capability 0 register*.

Note that it is possible to send a write command to several targets simultaneously while it is only possible to read from one.

113.14.34Descriptor target select 1 register

*Table 2203.*0x168 - DTARSEL1 - Descriptor target 1 select register

31	16
TARGSEL(63:48)	
0	
rw	

15	0
TARGSEL(47:32)	
0	
rw	

31:0 Target select (TARGSEL) for target 32-63
The core uses this field to select which targets from to send the command to (which chip enable (CE_n) signals to assert).
The least significant bit in this field corresponds to the 32nd target (CE_n(32)) etc.
The actual number of bits implemented equals the *ntargs* field +1 in the *Capability 0 register*.

Note that it is possible to send a write command to several targets simultaneously while it is only possible to read from one.

113.14.35Descriptor channel select register

*Table 2204.*0x16c - DCHSEL - Descriptor channel select register

31	16
RESERVED	
0	
r	

15	0
CHSEL	
0	
rw	

31:16 Reserved. This field is reserved for future use and should be set to 0.

15:0 Channel select (CHSEL)
The core uses this field to select which channels are to be active. Signals linked to a channel are IO0-I015, DQS, ALE, CLE, RE_N, WE_N, WP, DATAOE and DQSOE. In addition the PHY signals that are linked to a channel includes DDR_MDE, DDR_VALID and DATAI_VALID.
The least significant bit in this field corresponds to the first channel. The actual number of channels implemented equals the *nrofch* field +1 in the *Capability 0 register*.

Note that it is possible to send a write command to several channels simultaneously while it is only possible to read from one.

113.14.36Descriptor ready/busy select register

*Table 2205.*0x170 - DRBSEL - Descriptor ready/busy select register

31	16
RBSEL (31:16)	
0	
rw	

15	0
RBSEL (15:0)	
0	
rw	

31:0 Ready/Busy select (RBSEL)
The core uses this field to select which RB_n signals are to be active. The least significant bit in this field corresponds to the first RB_n signal. The actual number of RB_n signals implemented equals the *nrofrb* field +1 in the *Capability 0 register*.

113.14.37Descriptor row & control register

*Table 2206.*0x174- DROW - Descriptor row & control register

31	24	23	16
RESERVED	TAGEN	ROWADDR(23:16)	
0	0	0	
r	rw	rw	

15	0
ROWADDR(15:0)	
0	
rw	

31:25 Reserved. This field is reserved for future use and should be set to 0.

24 Tag enable (TAGEN)
Enabling this field will read/write additional data bytes equal to the generic tag_size in the end of the memory area. Tag enable is supported if the generic tag_size is configured to a non-zero value, EDAC is enabled and size is configured equal to the data area of the selected memory.

23:0 Row Address (ROWADDR)
This field sets the three byte row address, which is used to address LUNs, blocks and pages.
As described in the ONFI 4.0 specification the least significant part of the row address is the page address, the middle part block address, and the most significant part is the LUN address.
(7:0) first address byte to flash device,
(15:8) second address byte to flash device,
(23:16) third address byte to flash device.
Exactly how many bits that are used for each part of the address depends on the architecture of the flash memory.
Software needs to write this field prior to issuing any command that has an address phase that includes the row address, as well as for those special commands that have a one or two byte address phase (SET FEATURES; GET FEATURES; READ ID, READ UNIQUE ID, READ PARAMETER PAGE, LUN Set Features, LUN Get Features, Volume Select, ODT Configure, ZQ Calibration Long and ZQ Calibration Short).
The core ignores this field if the command doesn't use the row address.
If randomization is enabled, the row address needs to be set correctly even if the specific access doesn't require it.

113.14.38Descriptor column & size register

*Table 2207.*0x178 - DCOLSIZE - Descriptor column & size register

31	16
SIZE	
0	
rw	

15	0
COLADDR	
0	
rw	

31:16 Command data size (SIZE)
If a command has a data out or data in phase then software needs to set this field to the number of bytes the data that should be read / written.
Software does not need to set this field for the commands SET FEATURES, GET FEATURES, READ STATUS, or READ STATUS ENHANCED since their data phases are always the same size.
The core also ignores this field if the command issued doesn't have a data phase, as for example BLOCK ERASE.

15:0 Column address (COLADDR)
This field sets the two byte column address, which is used to address into a flash memory page.
See the ONFI 4.0 specification for more information about column address.
(7:0) first column byte to flash device.
(15:8) second column byte to flash device.

113.14.39Descriptor status register

*Table 2208.*0x17c - DSTS - Descriptor status register

31	16
ECFAIL	
0	
r*	

15	9	8	7	6	5	4	3	2	1	0
RESERVED		CERR DL	CERR UL	UERR DL	UERR UL	TMOUT	STOP LL	ABORT	UE	INV
0		0	0	0	0	0	0	0	0	0
r		r	r	r	r	r	r	r	r	r

31:16 ECC Chunk fail field (ECFAIL)
Each bit set to 1 in this field corresponds to a specific chunk that failed ecc. The lsb bit represents the first chunk read.
This field is cleared when a new descriptor is triggered.

15:9 Reserved.

8 Descriptor correctable error in downlink syncrams (CERR_DL)
This is the status signal for a specific descriptor access corresponding the status signal in the *Core status 1 register*.
This field is not valid in Register Mode.

7 Descriptor correctable error in uplink syncrams (CERR_UL)
This is the status signal for a specific descriptor access corresponding the status signal in the *Core status 1 register*.
This field is not valid in Register Mode.

6 Descriptor uncorrectable error in downlink syncrams(UCERR_DL)
This is the status signal for a specific descriptor access corresponding the status signal in the *Core status 1 register*.
This field is not valid in Register Mode.

5 Descriptor uncorrectable error in uplink syncrams (UCERR_UL)
This is the status signal for a specific descriptor access corresponding the status signal in the *Core status 1 register*.
This field is not valid in Register Mode.

4 Descriptor timout (TMOUT)
This is the status signal for a specific descriptor access corresponding the status signal in the *Core status 1 register*.
This field is not valid in Register Mode.

3 Descriptor stop linked list (STOP_LL)
This is the status signal for a specific descriptor access corresponding the status signal in the *Core status 1 register*.
This field is not valid in Register Mode.

2 Descriptor abort (ABORT)
This is the status signal for a specific descriptor access corresponding the status signal in the *Core status 1 register*.
This field is not valid in Register Mode.

1 Descriptor uncorrectable error (UE)
This is the status signal for a specific descriptor access corresponding the status signal in the *Core status 1 register*.
This field is not valid in Register Mode.

0 Descriptor invalid command (INV)
This is the status signal for a specific descriptor access corresponding the status signal in the *Core status 1 register*.
This field is not valid in Register Mode.

113.14.40 Descriptor ECC status register

Table 2209. 0x180 - DECCSTS0 - Descriptor ECC status register

31	24	23	16
RESERVED		CEC	
0			
r		r*	

15	12	11	0
RESERVED		CE	
0		0	
r		r*	

31:24 Reserved.

23:16 Corrected errors in the worst chunk (CEC).
This field is cleared when a new descriptor is triggered.

15:12 Reserved.

11:0 Corrected errors (CE)
Total number of corrected errors over all chunks.
This field is cleared when a new descriptor is triggered.

113.14.41 Descriptor data pointer low register

Table 2210. 0x190 - DDPL - Descriptor data pointer low register

31	16
DATA POINTER (31:16)	
0	
rw	

15	0
DATA POINTER (15:0)	
0	
rw	

31:0 Data pointer (DATA POINTER)(31:0)
Address in memory from where DMA reads or writes data.

113.14.42Data-in 0-3 register

Data-in register 0 holds the data bytes (Data 0-3) read from the flash memory device for each command reading data.

Data-in register 1 holds the data bytes (Data 4-7) read from the flash memory device for each command reading data.

Data-in register 2 holds the data bytes (Data 8-11) read from the flash memory device for each command reading data.

Data-in register 3 holds the data bytes (Data 12-15) read from the flash memory device for each command reading data.

Data-in register 0 bit 7:0 is the first byte read from flash.

*Table 2211.*0x1e0-0x1ec - Data-in 0 - Data-in 3 registers little endian

31 24	23 16
DATA 3, 7, 11, 15	DATA 2, 6, 10, 14
0	0
r	r

15 8	7 0
DATA 1, 5, 9, 13	DATA 0, 4, 8, 12
0	0
r	r

31:24	DATA 3, 7, 11, 15
23:16	DATA 2, 6, 10, 14
15:8	DATA 1, 5, 9, 13
7:0	DATA 0, 4, 8, 12

Those registers are not reset by the soft reset.

*Table 2212.*0x1e0-0x1ec - Data-in 0 - Data-in 3 registers big endian

31 24	23 16
DATA 0, 4, 8, 12	DATA 1, 5, 9, 13
0	0
r	r

15 8	7 0
DATA 2, 6, 10, 14	DATA 3, 7, 11, 15
0	0
r	r

31:24	DATA 0, 4, 8, 12
23:16	DATA 1, 5, 9, 13
15:8	DATA 2, 6, 10, 14
7:0	DATA 3, 7, 11, 15

Those registers are not reset by the soft reset.

113.14.43Data-out 0-3 register

Data-out register 0 holds the data bytes (Data 0-3) to be written to the flash memory device for each command writing data.

Data-out register 1 holds the data bytes (Data 4-7) to be written to the flash memory device for each command writing data.

Data-out register 2 holds the data bytes (Data 8-11) to be written to the flash memory device for each command writing data.

Data-out register 3 holds the data bytes (Data 12-15) to be written to the flash memory device for each command writing data

Data-out register 0 bit 7:0 is the first byte written to flash.

*Table 2213.*0x1f0-0x1fc - Data-out 0 - Data-out 3 registers little endian

31 — 24	23 — 16
DATA 3, 7, 11, 15	DATA 2, 6, 10, 14
0	0
rw	rw

15 — 8	7 — 0
DATA 1, 5, 9, 13	DATA 0, 4, 8, 12
0	0
rw	rw

31:24	DATA 3,7,11,15
23:16	DATA 2,6,10,14
15:8	DATA 1,5,9,13
7:0	DATA 0,4,8,12

*Table 2214.*0x1f0-0x1fc - Data-out 0 - Data-out 3 registers big endian

31 — 24	23 — 16
DATA 0, 4, 8, 12	DATA 1, 5, 9, 13
0	0
rw	rw

15 — 8	7 — 0
DATA 2, 6, 10, 14	DATA 3, 7, 11, 15
0	0
rw	rw

31:24	DATA 0, 4, 8, 12
23:16	DATA 1, 5, 9, 13
15:8	DATA 2, 6, 10, 14
7:0	DATA 3, 7, 11, 15

113.15 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x0C5. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

113.16 Implementation

113.16.1Technology mapping

The core has four technology mapping VHDL generics: *memtech_uldl*, *memtech_ecc0*, *memtech_ecc1* and *tech*. The generic *memtech_uldl, memtech_ecc0* and *memtech_ecc1* controls the technology used for memory cell implementation of uplink (*from* flash device *to* AMBA) and downlink (*from* AMBA *to* flash device) FIFOs, ecc block 0 and ecc block 1. The generic *tech* controls the technology used for the R/B_n synchronization flip-flops.

113.16.2Reset

The core changes reset behavior depending on settings in the GRLIB configuration package (see GRLIB User's Manual). Additionally, some of the outputs from the core are always asynchronously reset including *ce_n*, *datao*, *dataoe* and *dqsoe*.

113.16.3Endianness

The core is designed for big-endian and little-endian systems. NANDFCTRL2 automatically detects the endianness of the system by reading a dedicated sideband signal included in the AMBA records.

113.17 Configuration options

Table 2215 shows the configuration options of the core (VHDL generics).

*Table 2215.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index.	0 - NAPBSLV-1	0
pirq	APB irq number.	0 - NAHBIRQ-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFE#
ahbbits	Width of AHB read/write data buses and maximum access size.	32, 64	AHBDW
memtech_uldl	Memory technology used for in uplink and downlink	0 - NTECH	inferred(0)
memtech_ecc0	Memory technology used for in ECC block 0	0 - NTECH	inferred(0)
memtech_ecc1	Memory technology used for in ECC block 1	0 - NTECH	inferred(0)
tech	Technology used for the R/B_n synch flip-flops	0 - NTECH	inferred(0)
nrofce	Number of chip select signals connected to the core.	1 - 64	8
nrofch	Number of channels (controls IO0-15, DQS, RE_N, WE_N, CLE, ALE, WP_N and OE) connected to the core.	1 - 16	8
nrofrb	Number of R/B_n and SEFI_FLAG signals connected to the core.	1 - 32	8
rnd	Enable randomization support.	0 - 1	1
mem0_data	Data size of page in bytes.	4096, 8192, 16384	16384
mem0_spare	Spare size of page in bytes. Note that the spare size needs to be larger than the size of the ECC produces check bytes.	2 - 4096	2208
mem0_ecc_sel	Mapping memory setting to a specific ECC block.	0 - 1	0

*Table 2215.*Configuration options

Generic name	Function	Allowed range	Default
mem1_data	Data size of page in bytes.	4096, 8192, 16384	8192
mem1_spare	Spare size of page in bytes. Note that the spare size needs to be larger than the size of the ECC produces check bytes.	2 - 4096	448
mem1_ecc_sel	Mapping memory setting to a specific ECC block.	0 - 1	1
mem2_data	Data size of page in bytes.	4096, 8192, 16384	4096
mem2_spare	Spare size of page in bytes. Note that the spare size needs to be larger than the size of the ECC produces check bytes.	2 - 4096	224
mem2_ecc_sel	Mapping memory setting to a specific ECC block.	0 - 1	1
ecc0_gfsize	Size of internal symbols in ECC 0. This generic should be set to the lowest value fulfilling the following constraint: ecc0_gfsize > log2 (8*ecc0_chunk + ecc0_gfsize*ecc0_cap)	13, 14	14
ecc0_chunk	Number of data bytes for ECC 0 (note that this is the size of the data chunk and not including the ECC bytes).	512, 1024	1024
ecc0_cap	Error correction capability of ECC 0. If 0, then no BCH block is instantiated.	0 - 60	60
ecc1_gfsize	Size of internal symbols in ECC 1. This generic should be set to the lowest value fulfilling the following constraint: ecc1_gfsize > log2 (8*ecc1_chunk + ecc1_gfsize*ecc1_cap)	13, 14	13
ecc1_chunk	Number of data bytes for ECC 1 (note that this is the size of the data chunk and not including the ECC bytes).	512, 1024	512
ecc1_cap	Error correction capability of ECC 1. If 0, then no BCH block is instantiated. ECC 0 needs to be implemented (ecc_cap > 0) for ECC 1 to be instantiated.	0 - 60	16
rst_cycles	rst_cycles controls the time the controller keeps the bi-directional signals to the nand flash device(s) as inputs after reset. It should be set to the number of core clock cycles that gives a total time of 100 ns, which assure the bi-directional signals are not double driven.	0 - 200	10
tag_size	tag_size controls the size of tag data in number of bytes when tagen is enabled. Setting this generic to 0 will disable ecc protection of tag data.	0 - 63	0
ft	Fault-tolerance on internal syncrams. Technology specific protection is further documented in the GRLIB-FT User's Manual (grlib-ft.pdf)	0 - 5	0
scantest	Enable scantest support.	0 - 1	0
oepol	Polarity of pad output enable signal.	0 - 1	0

113.18 Signal descriptions

Table 2216 shows the interface signals of the core (VHDL ports).

*Table 2216.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	System reset	Logical 0
CLK_SYS	N/A	Input	System clock	6
CORE_RSTN	N/A	Input	Core reset	Logical 0
CLK_CORE	N/A	Input	Core clock	6, 7
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
PHYI	rb_n(31:0)[4]	Input	Ready/Busy signal	Logical 0
	datai(15:0)(15:0)[3]	Input	Data input	-
	datai_valid(15:0)[3]	Input	Data input valid from PHY	Logical 1
	phy_ready	Input	PHY ready	Logical 1
PHYO	ce_n(63:0)[2]	Output	Chip enable	Logical 0
	wp_n(15:0)[3]	Output	Write protect	Logical 0
	datao(15:0)(15:0)[3]	Output	Data output	-
	dqso(15:0)[3]	Output	Data strobe output	-
	cle(15:0)[3]	Output	Command latch enable	Logical 1
	ale(15:0)[3]	Output	Address latch enable	Logical 1
	we_n(15:0)[3]	Output	Write enable	Logical 0
	re_n(15:0)[3]	Output	Read enable	Logical 0
	dataoe(15:0)[3]	Output	Data output enable	1
	dqsoe(15:0)[3]	Output	DQS output enable	1
	sefi_flag(31:0)[4]	Output	SEFI output control signal	Logical 1
	ddr_mde(15:0)[3]	Output	DDR Mode to PHY	Logical 1
	ddr_valid(15:0)[3]	Output	DDR Data valid to PHY	Logical 1
	ifsel(1:0)	Output	Interface Select to PHY	-
	t_reh(15:0)	Output	Timing parameter tREH to PHY	-
	t_rp(15:0)	Output	Timing parameter tRP to PHY	-
	t_dqsh(15:0)	Output	Timing parameter tDQSH to PHY	-
	t_dqsl(15:0)	Output	Timing parameter tDQSL to PHY	-
	abort	Output	Abort PHY	Logical 1
	skew_ctrl _rx(9:0)(2:0)	Output	PHY rx skew control	
	skew_ctrl _tx(9:0)(2:0)	Output	PHY tx skew control	
	testen	Output	Scantest	Logical 1
	testrst	Output	Scantest	Logical 0

Table 2216. Signal descriptions

Signal name	Field	Type	Function	Active
MTESTI[5]	ul_memtest_vector_array	Input	Memory BIST input signal for the uplink FIFO.	-
	dl_memtest_vector_array	Input	Memory BIST input signal for the downlink FIFO.	-
	bch1_memtest_vector_array	Input	Memory BIST input signal for the FIFO in ecc block 0	-
	bch2_memtest_vector_array	Input	Memory BIST input signal for the FIFO in ecc block 1	-
MTESTO[5]	ul_memtest_vector_array	Output	Memory BIST output signal for uplink FIFO.	-
	dl_memtest_vector_array	Output	Memory BIST output signal for the downlink FIFO	-
	bch1_memtest_vector_array	Output	Memory BIST output signal for the FIFO in ecc block 0	-
	bch2_memtest_vector_array	Output	Memory BIST output signal for the FIFO in ecc block 1	-
MTEST_CLK[5]	N/A	Input	Memory BIST clock	-

* See GRLIB IP Library User's Manual

[1] The polarity of the output enable signal is implementation dependent.

[2] The number of CE_n is set as generic *nrofce.*

[3] The size of the signals depends on the generic *nrofch.* The size of datai and datao is equal to 16**nrofch.*

[4] The size of the signals depends on the generic *nrofrb.*

[5] Not available in FPGA releases

[6] In the case a PHY with a phy_clock ipnut is connected to the core. The phy clock of the PHY should be 4 times the core clock and the two clocks should be synchronous and the positive edges must align

[7] **The current version of the core requires the frequency of clk_sys to be equal or larger than clk_core.**

113.19 Signal definitions and reset values

The signals and their reset values are described in table 2217.

*Table 2217.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
rb_n (31:0)[4]	Input	Ready/Busy signal	-	-
data (15:0)(15:0)[3]	Input/Output	Data (datai and datao)	-	-
datai_valid(15:0)[3]	Input	Data valid from PHY	-	-
phy_ready	Input	PHY ready	-	-
ce_n (31:0)[2]	Output	Chip enable	Logical 0	Logical 1
wp_n(15:0)[3]	Output	Write protect	Logical 0	Logical 0
cle(15:0)[3]	Output	Command latch enable	Logical 1	Logical 0
ale(15:0)[3]	Output	Address latch enable	Logical 1	Logical 0
we_n(15:0)[3]	Output	Write enable	Logical 0	Logical 1
re_n(15:0)[3]	Output	Read enable	Logical 0	Logical 1
dqso(15:0)[3]	Output	Data strobe	-	-
dataoe(15:0)[3]	Output	Data output enable	1	1
dqsoe(15:0)[3]	Output	DQS output enable	1	1
sefi_flag(31:0)[4]	Output	SEFI output cntrl	Logical 1	Logical 0
ddr_mde(15:0)[3]	Output	DDR Mode to PHY	Logical 1	Logical 0
ddr_valid(15:0)[3]	Output	DDR Data valid to PHY	Logical 1	Logical 0
ifsel(1:0)	Output	Interface Select to PHY	-	Logical 0
t_reh(15:0)	Output	Timing parameter tREH to PHY	-	Logical 0
t_rp(15:0)	Output	Timing parameter tRP to PHY	-	Logical 0
t_dqsh(15:0)	Output	Timing parameter tDQSH to PHY	-	Logical 0
t_dqsl(15:0)	Output	Timing parameter tDQSL to PHY	-	Logical 0
abort	Output	Abort PHY	Logical 1	Logical 0
skew_ctrl _rx(9:0)(2:0)	Output	PHY rx skew control	-	Logical 0
skew_ctrl _tx(9:0)(2:0)	Output	PHY tx skew control	-	Logical 0
testen	Output	Scantest	Logical 1	-
testrst	Output	Scantest	Logical 0	-

[1] The polarity of the output enable signal is implementation dependent.

[2] The number of CE_n is set as generic *nrofce*

[3] The size of the signals depends on the generic *nrofch*. The size of datai and datao is equal to 8**nrofch*.

[4] The size of the signals depends on the generic *nrofrb*.

113.20 Library dependencies

Table 2218 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2218.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	NANDFC-TRL2_PKG	Signals, component	Component declaration
TECHMAP	GENCOMP	components	Technology mapping

113.21 Instantiation

This example shows how the core can be instantiated using one ecc block together with the nandfc-trl2_nvddr2_phy_generic. The impact of the generics can be seen in table 2215.

```
library ieee, grlib, techmap, gaisler;
use ieee.std_logic_1164.all;
use grlib.amba.all;
use techmap.gencomp.all;
use gaisler.nandfctrl2_pkg.all;

entity nandfctrl2_ex is
  port (
    rstn       : in  std_logic;
    clk_sys    : in  std_logic;
    core_rstn  : in  std_logic;
    clk_core   : in  std_logic;
    clk_phy    : in  std_logic; -- clk_phy is equal to 4 times clk_core
    phy_rstn   : in  std_logic;
    .. );
end;

architecture rtl of nandfctrl2_ex is

  -- NANDFCTRL2 constants
  constant NROFCE : integer :=  4 ;
  constant NROFCH : integer :=  2 ;
  constant NROFRB : integer :=  4 ;

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- NANDFCTRL2 signals
  signal nf2_to_phy_in  : nf2_to_phy_out_type;
  signal nf2_to_phy_out : nf2_to_phy_in_type;
  signal phyi           : from_nandf_pads_type;
  signal phyo           : to_nandf_pads_type;


begin

...

  -- NANDFCTRL2 core
  nand0 : nandfctrl2
    generic map(
      hindex       => 10,
      pindex       => 10,
      paddr        => 10,
      pirq         => 10,
```

```
        nrofce       => NROFCE,
        nrofch       => NROFCH,
        nrofrb       => NROFRB,
        mem0_data    => 16384,
        mem0_spare   => 2208,
        mem0_ecc_sel => 0,
        ecc0_gfsize  => 14,
        ecc0_chunk   => 1024,
        ecc0_cap     => 60,
        ecc1_cap     => 0)
      port map(
        rstn         => rstn,
        clk_sys      => clk_sys,
        core_rstn    => core_rstn,
        clk_core     => clk_core,
        apbi         => apbi,
        apbo         => apbo(10),
        ahbmi        => ahbmi,
        ahbmo        => ahbmo(10),
        phyi         => nf2_to_phy_in,
        phyo         => nf2_to_phy_out
        );

  -- NANDFCTRL2_PHY
    nand_phy : nandfctrl2_nvddr2_phy_generic
      generic map(
        nrofce       => NROFCE,
        nrofch       => NROFCH,
        nrofrb       => NROFRB)
      port map(
        rstn_core    => rstn,
        clk_core     => clk_core,
        rstn_phy     => phy_rstn,
        clk_phy      => clk_phy,
        nf2i         => nf2_to_phy_out,
        nf2o         => nf2_to_phy_in,
        nandfi       => phyi,
        nandfo       => phyo
        );

    -- Pads for NANDFCTRL2 core

    nandf_ce0_0 : outpad generic map (tech => padtech)
      port map (Ce0_0_n, phyo.ce_n (0) );
    nandf_ce0_1 : outpad generic map (tech => padtech)
      port map (Ce0_1_n, phyo.ce_n (1));
    nandf_ce1_0 : outpad generic map (tech => padtech)
      port map (Ce1_0_n, phyo.ce_n (2));
    nandf_ce1_1 : outpad generic map (tech => padtech)
      port map (Ce1_1_n, phyo.ce_n (3));

    -- rb_n input pads with pull-up resistor

    nandf_rb0_0 : inpad generic map (tech => padtech)
      port map (Rb0_0_n, phyi.rb_n (0) );
    nandf_rb1_0 : inpad generic map (tech => padtech)
      port map (Rb0_1_n, phyi.rb_n (1) );
    nandf_rb0_1 : inpad generic map (tech => padtech)
      port map (Rb1_0_n, phyi.rb_n (2) );
    nandf_rb1_1 : inpad generic map (tech => padtech)
      port map (Rb1_1_n, phyi.rb_n (3) );

    nandf_d_0 : iopadv generic map (tech => padtech, width => 8)
      port map (Dq_Io_0, phyo.dq (0), phyo.dq_oe (0), phyi.dq (0));
    nandf_dqs_0 : iopad generic map (tech => padtech)
      port map (Dqs_0, phyo.dqs (0), phyo.dqs_oe (0), phyi.dqs (0));
    nandf_we_0 : outpad generic map (tech => padtech)
      port map (Clk_We_0_n, phyo.we_n (0) );
    nandf_re_0 : outpad generic map (tech => padtech)
      port map (Wr_Re_0_n, phyo.re_n (0));
    nandf_cle_0 : outpad generic map (tech => padtech)
```

```
      port map (Cle_0, phyo.cle (0));
    nandf_ale_0 : outpad generic map (tech => padtech)
      port map (Ale_0, phyo.ale (0));
    nandf_wp_0 : outpad generic map (tech => padtech)
      port map (Wp_0_n, phyo.wp_n (0));

    nandf_d_1 : iopadv generic map (tech => padtech, width => 8)
      port map (Dq_Io_1, phyo.dq (1), phyo.dq_oe (1), phyi.dq (1));
    nandf_dqs_1 : iopad generic map (tech => padtech)
      port map (Dqs_1, phyo.dqs (1), phyo.dqs_oe (1), phyi.dqs (1));
    nandf_we_1 : outpad generic map (tech => padtech)
      port map (Clk_We_1_n, phyo.we_n (1) );
    nandf_re_1 : outpad generic map (tech => padtech)
      port map (Wr_Re_1_n, phyo.re_n (1) );
    nandf_cle_1 : outpad generic map (tech => padtech)
      port map (Cle_1, phyo.cle (1) );
    nandf_ale_1 : outpad generic map (tech => padtech)
      port map (Ale_1, phyo.ale (1) );
    nandf_wp_1 : outpad generic map (tech => padtech)
      port map (Wp_1_n, phyo.wp_n (1));

  phyi.rb_n(MAX_RB-1 downto NROFRB) <= (others => '1');
  gen_nandf_ch_iopads_unused : FOR ii IN MAX_CHANNELS-1 DOWNTO NROFCH GENERATE
      phyi.dq(ii)  <= (others => '0');
      phyi.dqs(ii) <= '0';
    END GENERATE gen_nandf_ch_iopads_unused;

  end;
```

114 NANDFCTRL2 PHY - NAND Flash Memory Controller PHY

114.1 Overview

This chapter specifies different PHYs that should be used between NANDFCTRL2 - NAND flash memory controller and one or more nand flash memory device(s).

The PHY handles the physical interface to the nand flash device(s) assuring the configured timing mode and performance requirements are achieved including DDR. It uses control signals from NANDFCTRL2 including selection of SDR and NV-DDR2/3 data interface and timing parameter.

114.2 NANDFCTRL2 Generic NV-DDR2 PHY

The nandfctrl2_generic_nvddr2_phy supports the NV-DDR2, NV-DDR3 and SDR data interface.

114.2.1 Clocks and constraints

The PHY uses a core clock (clk_core) and a phy clock (clk_phy). There are some constraints on the clocks that needs to be fulfilled:

- clk_core should be the same clock as the core clock of NANDFCTRL2.
- clk_phy must be four times the clk_core.
- clk_phy must be synchronous to clk_core.
- clk_core positive edge must match a clk_phy positive edge.

114.2.2 Reset

The PHY changes reset behavior depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

114.2.3 Configuration options

Table 2219 shows the configuration options of the core (VHDL generics).

*Table 2219.*Configuration options

Generic name	Function	Allowed range	Default
NROFCE	Number of chip select signals connected to the core.	1 - 64	8
NROFCH	Number of channels (controls DQ0-7, DQS, RE_N, WE_N, CLE, ALE, WP_N, DQ_OE, DQS_OE, DDR_MDE, DDR_VALID and DATAI_VALID).	1 - 16	8
NROFRB	Number of RB_n signals.	1 - 32	8
NROFSEFI	Number of SEFI_FLAG signals.	0 - 32	1
OEPOL	Polarity of pad output enable signal.	0 - 1	0
SCANTEST	Enable scantest support.	0 - 1	0
SKEW_RX_MAX	Size of skew rx control registers	1 - 7	7
SKEW_TX_MAX	Size of skew tx control registers	1 - 7	7
SYNC_STAGES	The number of sync stages for read data	0 - 2	2
TECH	Technology used for the dqsi synch flip-flops	0 - NTECH	inferred(0)

114.2.4 Signal descriptions

Table 2220 shows the interface signals of the core (VHDL ports).

*Table 2220.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN_CORE	N/A	Input	Core reset	Logical 0
RSTN_PHY	N/A	Input	PHY reset	Logical 0
CLK_CORE	N/A	Input	Core clock	-
CLK_PHY	N/A	Input	PHY clock	-
NF2O	rb_n(31:0)[4]	Output	Ready/Busy signal	Logical 0
	datai(15:0)(15:0)[3]	Output	Data input	-
	datai_valid(15:0)[3]	Output	Data input valid	Logical 1
	phy_ready	Output	PHY ready	Logical 1
NF2I	ce_n(63:0)[2]	Input	Chip enable	Logical 0
	wp_n(15:0)[3]	Input	Write protect	Logical 0
	datao(15:0)(15:0)[3]	Input	Data output	-
	dqso(15:0)[3]	Input	Data strobe output	-
	cle(15:0)[3]	Input	Command latch enable	Logical 1
	ale(15:0)[3]	Input	Address latch enable	Logical 1
	we_n(15:0)[3]	Input	Write enable	Logical 0
	re_n(15:0)[3]	Input	Read enable	Logical 0
	dataoe(15:0)[3]	Input	Data output enable	1
	dqsoe(15:0)[3]	Input	DQS output enable	1
	sefi_flag(31:0)[5]	Input	SEFI output control signal	Logical 1
	ddr_mde(15:0)[3]	Input	DDR Mode to PHY	Logical 1
	ddr_valid(15:0)[3]	Input	DDR Data valid to PHY	Logical 1
	ifsel(1:0)	Input	Interface Select to PHY	-
	t_reh(15:0)	Input	Timing parameter tREH to PHY	6
	t_rp(15:0)	Input	Timing parameter tRP to PHY	6
	t_dqsh(15:0)	Input	Timing parameter tDQSH to PHY	6
	t_dqsl(15:0)	Input	Timing parameter tDQSL to PHY	6
	abort	Input	Abort PHY	Logical 1
	skew_ctrl_rx(9:0)(2:0)	Input	PHY rx skew control	-
	skew_ctrl_tx(9:0)(2:0)	Input	PHY tx skew control	-
	testen	Input	Scantest	Logical 1
	testrst	Input	Scantest	Logical 0
NANDFI	rb_n(31:0)[4]	Input	Ready/Busy signal	Logical 0
	dq(7:0)(15:0)[3]	Input	Data input	-
	dqs(15:0)[3]	Input	Data strobe input	-

*Table 2220.*Signal descriptions

Signal name	Field	Type	Function	Active
NANDFO	ce_n(63:0)[2]	Output	Chip enable	Logical 0
	ale(15:0)[3]	Output	Address latch enable	Logical 1
	cle(15:0)[3]	Output	Command latch enable	Logical 1
	dq(7:0)(15:0)[3]	Output	Data output	-
	dq_oe(15:0)[3]	Output	Data output enable	1
	dqs(15:0)[3]	Output	Data strobe output	-
	dqs_oe(15:0)[3]	Output	DQS output enable	1
	we_n(15:0)[3]	Output	Write enable	Logical 0
	re_n(15:0)[3]	Output	Read enable	Logical 0
	wp_n(15:0)[3]	Output	Write protect	Logical 0
	sefi_flag[5]	Output	SEFI output control signal	Logical 1

[1] The polarity of the output enable signal is implementation dependent.

[2] The number of CE_n is set as generic *nrofce.*

[3] The size of the signals depends on the generic *nrofch.* The size of datai and datao is equal to 16**nrofch.* The size of dq is 8**nrofch.*

[4] The size of the signals depends on the generic *nrofrb.*

[5] The size of the signals depends on the generic *nrofsefi.*

[6] **The current version of the core requires the cycle time of DQS and RE_n (tDSC and tRC) to be at least 2 core clock periods (8 phy clock periods) if using the NV-DDR2/3 data interface. This limitation does not apply to the SDR (asynchronous) data interface**

114.2.5 Library dependencies

Table 2221 shows the libraries used when instantiating the PHY (VHDL libraries).

*Table 2221.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	NANDFTRL2_PKG	Signals, component	Component and signal declaration

114.2.6 Instantiation

Refer to the NANDFCTRL2 section for how to instantiate NANDFCTRL2_GENERIC_NVDDR2_PHY together with NANDFCTRL2.

114.3 NANDFCTRL2 Generic SDR PHY

The nandfctrl2_generic_sdr_phy supports the SDR data interface.

114.3.1 Clocks and constraints

The PHY only use the core clock (clk_core).

114.3.2 Reset

The PHY changes reset behavior depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

114.3.3 Configuration options

Table 2219 shows the configuration options of the core (VHDL generics).

*Table 2222.*Configuration options

Generic name	Function	Allowed range	Default
NROFCE	Number of chip select signals connected to the core.	1 - 64	8
NROFCH	Number of channels (controls DQ0-7, DQS, RE_N, WE_N, CLE, ALE, WP_N, DQ_OE, DQS_OE, DR_MDE, DDR_VALID and DATAI_VALID).	1 - 16	8
NROFRB	Number of RB_n signals.	1 - 32	8
NROFSEFI	Number of SEFI_FLAG signals.	0 - 32	1
OEPOL	Polarity of pad output enable signal.	0 - 1	0
SCANTEST	Enable scantest support.	0 - 1	0
SYNC_STAGES	Unused and obsolete.	0 - 3	0

114.3.4 Signal descriptions

Table 2220 shows the interface signals of the core (VHDL ports).

*Table 2223.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN_CORE	N/A	Input	Core reset	-
CLK_CORE	N/A	Input	Core clock	-
NF2O	rb_n(31:0)[4]	Output	Ready/Busy signal	Logical 0
	datai(15:0)(15:0)[3]	Output	Data input	-
	datai_valid(15:0)[3]	Output	Data input valid from PHY	Logical 1
	phy_ready	Output	PHY ready	Logical 1

Table 2223. Signal descriptions

Signal name	Field	Type	Function	Active
NF2I	ce_n(63:0)[2]	Input	Chip enable	Logical 0
	wp_n(15:0)[3]	Input	Write protect	Logical 0
	datao(15:0)(15:0)[3]	Input	Data output	-
	dqso(15:0)[3]	Input	Data strobe output	-
	cle(15:0)[3]	Input	Command latch enable	Logical 1
	ale(15:0)[3]	Input	Address latch enable	Logical 1
	we_n(15:0)[3]	Input	Write enable	Logical 0
	re_n(15:0)[3]	Input	Read enable	Logical 0
	dataoe(15:0)[3]	Input	Data output enable	1
	dqsoe(15:0)[3]	Input	DQS output enable	1
	sefi_flag(31:0)[4]	Input	SEFI output control signal	Logical 1
	ddr_mde(15:0)[3]	Input	DDR Mode to PHY	Logical 1
	ddr_valid(15:0)[3]	Input	DDR Data valid to PHY	Logical 1
	ifsel(1:0)	Input	Interface Select to PHY	-
	t_reh(15:0)	Input	Timing parameter tREH to PHY	-
	t_rp(15:0)	Input	Timing parameter tRP to PHY	-
	t_dqsh(15:0)	Input	Timing parameter tDQSH to PHY	-
	t_dqsl(15:0)	Input	Timing parameter tDQSL to PHY	-
	abort	Input	Abort PHY	Logical 1
	skew_ctrl_rx(9:0)(2:0)	Input	PHY rx skew control	-
	skew_ctrl_tx(9:0)(2:0)	Input	PHY tx skew control	-
	testen	Input	Scantest	Logical 1
	testrst	Input	Scantest	Logical 0
NANDFI	rb_n(31:0)[4]	Input	Ready/Busy signal	Logical 0
	dq(7:0)(15:0)[3]	Input	Data input	-
	dqs(15:0)[3]	Input	Data strobe input	-

*Table 2223.*Signal descriptions

Signal name	Field	Type	Function	Active
NANDFO	ce_n(63:0)[2]	Output	Chip enable	Logical 0
	ale(15:0)[3]	Output	Address latch enable	Logical 1
	cle(15:0)[3]	Output	Command latch enable	Logical 1
	dq(7:0)(15:0)[3]	Output	Data output	-
	dq_oe(15:0)[3]	Output	Data output enable	[1]
	dqs(15:0)[3]	Output	Data strobe output	-
	dqs_oe(15:0)[3]	Output	DQS output enable	[1]
	we_n(15:0)[3]	Output	Write enable	Logical 0
	re_n(15:0)[3]	Output	Read enable	Logical 0
	wp_n(15:0)[3]	Output	Write protect	Logical 0
	sefi_flag[5]	Output	SEFI output control signal	Logical 1

[1] The polarity of the output enable signal is implementation dependent.

[2] The number of CE_n is set as generic *nrofce.*

[3] The size of the signals depends on the generic *nrofch.* The size of datai and datao is equal to 16**nrofch.* The size of dq is 8**nrofch.*

[4] The size of the signals depends on the generic *nrofrb.*

[5] The size of the signals depends on the generic *nrofsefi.*

114.3.5 Library dependencies

Table 2221 shows the libraries used when instantiating the PHY (VHDL libraries).

*Table 2224.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	NANDFTRL2_PKG	Signals, component	Component and signal declaration

114.3.6 Instantiation

Refer to the NANDFCTRL2 section for how to instantiate NANDFCTRL2_GENERIC__SDR_PHY together with NANDFCTRL2.

115 NOELVSYS - High-performance RISC-V RV64GC Processor Subsystem

115.1 Overview

NOEL-V is processor core that can be configured to conform to the 32-bit (RV32) or 64-bit (RV64) RISC-V architecture. It is designed for embedded applications, combining high performance with low complexity and low power consumption.

The NOELVSYS block combines the NOEL-V processor with essential peripherals needed to create a NOEL-V based system-on-chip

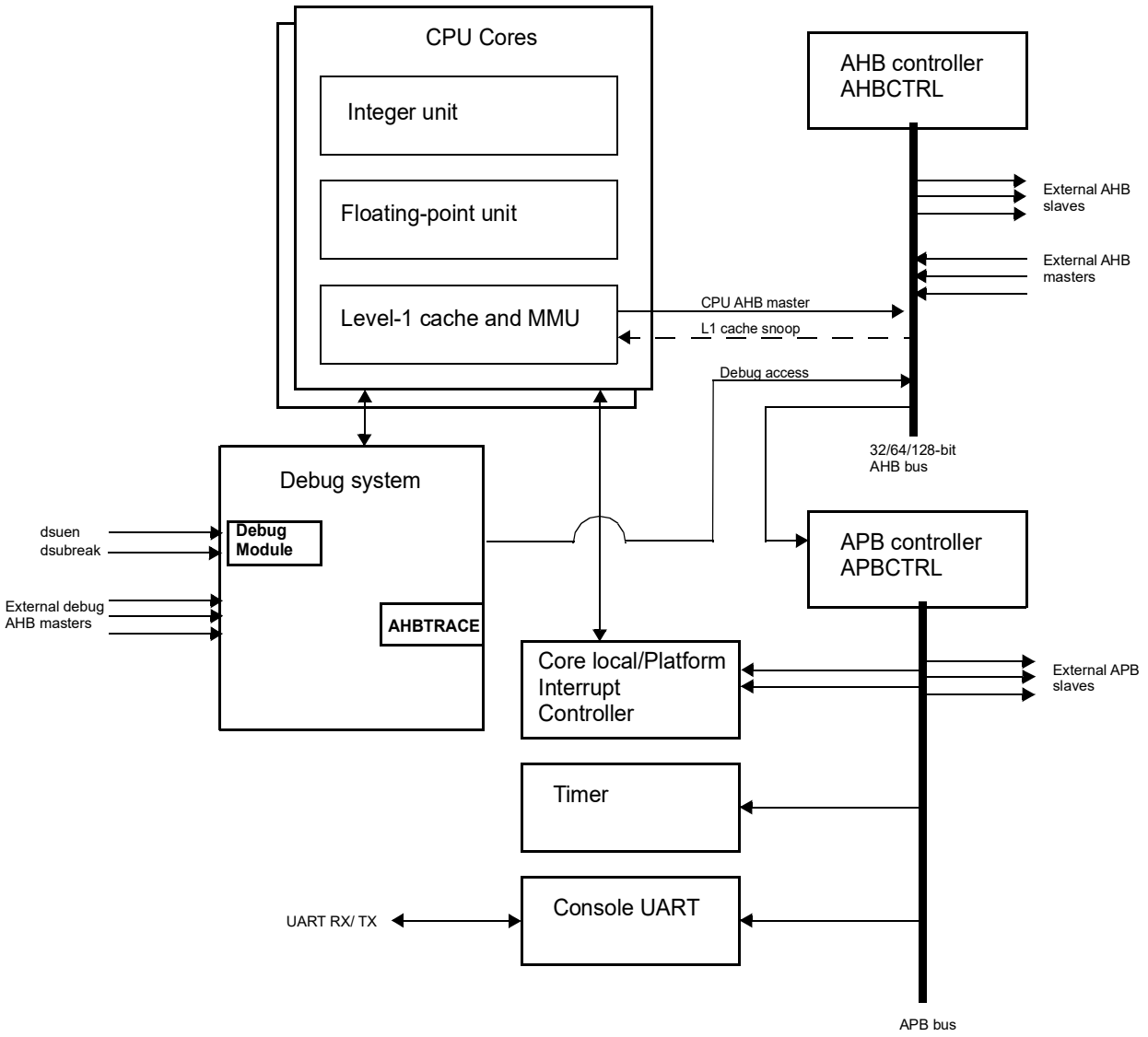


Figure 304. NOELVSYS subsystem block diagram

115.1.1 Standard configurations

Various standard configurations are provided that target both FPGA and ASIC implementations.

Table 2225. Standard configurations

Standard-Configuration	HP	GP		MC		Note
Sub-config	**-**	**GP**	**GP-lite**	**MC**	**MC-lite**	**lite = limited version**
CFG	0x400	0x30x	0x38x	0x20x	0x28x	NOEL-V configuration setting For some configurations 2 can be added to the CFG value to disable FPU or 1 is added to select single-issue pipeline.
XLEN	64/32	64/32		64/32		64- or 32-bit architecture
Pipeline	D	D/S		S		Single- or Dual-issue
Privilege levels	MSU +HSHU	MSU +HSHU	MSU	MU	MU	
Extension						
A	Y	Y	Y	Y	Y	Atomic operations
C	Y	Y	Y	Y	N*	Compressed Instructions
M	Y	Y	Y	Y	Y	Mul / Div
FD	Y	Y	Y	Y/N	N	Single-/Double-precision floating-point
H	Y	Y	N	N	N	Hypervisor support
Zba	Y	Y	Y	Y	N	Bit-manipulation: Address generation
Zbb	Y	Y	Y	N	N	Bit-manipulation: Base instructions
Zbs	Y	Y	Y	Y	N	Bit-manipulation: Single-bit instructions
Zbc/Zbkc	Y	Y	N	N	N	Bit-manipulation: Carry-less multiplication
Zbkb	Y	Y	N	N	N	Simple crypto bitmanipulation
Zbkx	Y	Y	N	N	N	Simple crypto permutation
Zcb	Y	Y	Y	Y	Y*	Additional compressed instructions
Zkn	Y	Y	N	N	N	Crypto AES and SHA instructions
Zkr	Y	Y	N	N	N	Crypto entropy (CSR)
Scofpmf	Y	Y	Y	Y	N	Counter filtering and overflow interrupts
Sstc	Y	Y	N	N	N	Supervisor mode timer interrupt
Smsstateen	Y	Y	N	N	N	
Smrnmi	Y	Y	N*	N*	N*	
Smepmp	Y	Y	N	Y	N	
Smaia	Y	Y	N	N	N	AIA interrupts
Ssaia	Y	Y	N	N	N	AIA interrupts
Zicsr	Y	Y	Y	Y	Y	Support for control and status registers (CSR)
Zifencei	Y	Y	Y	Y	Y	Support for instruction-fetch fence
Zicbom	Y	Y	Y	TBD*	TBD*	Cache management operations
Zicond	Y	Y	Y	N*	N*	
Zimop	Y	Y	Y	Y	Y	
Zfh	Y	Y	Y	N	N	Half-Precision Floating-Point
Zfhmin	Y	Y	Y	Y	N	Minimal Half-Precision Floating-Point

Table 2225. Standard configurations

Standard-Configuration	HP	GP		MC		Note
Sub-config	**-**	**GP**	**GP-lite**	**MC**	**MC-lite**	**lite = limited version**
Zfa	Y	Y	Y	Y	N	Additional Floating-Point
Features						
Late branch/ ALU	Y/Y	Y/Y	Y/Y	Y/Y	N/N	
PMP (entries/ granularity)	8/10	8/10	-	8/10*	-	
TLB (D/I/H)	16/16/16*	16/16/16	8/8/8	-	-	
BTB (entries/ ways)	16/2*	16/2*	16/2*	16/2*	16/2*	
BHT (entries/ length)	128/5*	128/5*	64/5*	64/5*	64/5*	
RAS (entries)	8**	8**	8**	8**	8**	Retrun-Address-Stack (currently disabled in all configurations)
I-Cache	4x4k	4x4k	4x4k	2x4k	2x4k	
D-Cache	4x4k	4x4k	4x4k	2x4k	2x4k	
TCM (I/D)	NA**	NA**	NA**	NA**	NA**	Tightly-coupled memory (not yet implemented)

* May be updated/changed in future releases.
** To be enabled in future releases.

[Note: Configurations can be updated when new features are available and standardization of more extension are complete.]

The High-performance (**HP**) configuration is defined to primarily target ASIC implementations. The General-purpose (**GP**) is more suited for large/medium FPGA implementations. The Minimal configuration (**MC**) is defined for smaller FPGAs. The “lite” version is a limited version (to reduce resource utilization) of the standard configuration without reducing the feature-set in a significant way.

115.1.2 RISC-V specification

The NOEL-V implements the following versions of the RISC-V specification.

Table 2226. RISC-V Unprivilege specification

The RISC-V Instruction Set Manual Volume I: Unprivileged ISA Document Version 20191213		
Base / Extension	**Version**	**Note**
RVWMO	**2.0**	
RV32I	**2.1**	
RV64I	**2.1**	
M	**2.0**	
A	**2.1**	
F	**2.2**	
D	**2.2**	
C	**2.0**	
Counters	**2.0**	
Zicsr	**2.0**	
Zifencei	**2.0**	

Table 2227. RISC-V Privilege specification

The RISC-V Instruction Set Manual Volume II: Privileged Architecture Document Version 20211203		
Extension	**Version**	**Note**
Machine ISA	**1.12**	
Supervisor ISA	**1.12**	
Hypervisor ISA	**1.0**	

Table 2228. RISC-V Other specification (external or not yet included)

Extension	Version	Note
RISC-V Debug Support	**1.0**	Update from version 0.13.2 is currently ongoing
Efficient Trace for RISC-V	**2.0**	Update from version 1.1.3 is currently ongoing
RISC-V Count Overflow and Mode-Based Filtering Extension (Sscofpmf)	**0.5.2**	Version 0.5.2-01d1df0, 2021-10-13: frozen **Ratified November 2021**
RISC-V Bit-Manipulation ISA-extensions: Zba, Zbb, Zbc, Zbs	**1.0.0**	Version 1.0.0-38-g865e7a7, 2021-06-28: Release candidate. **Ratified November 2021**
RISC-V Cryptography Extensions: Scalar & Entropy Source Instructions	**1.0.1**	**Ratified November 2021**
RISC-V Base Cache Management Operation ISA Extensions: Zicbom, Zicbop, Zicboz	**1.0.1**	Version 1.0.1-b34ea8a, 2022-05-13: Ratified **Ratified November 2021**
RISC-V "stimecmp/vstimecmp" Extension: Sstc		Version 0.5.4-3f9ed34, 2021-10-13: frozen **Ratified November 2021**
"Zfh" and "Zfhmin" Standard Extensions for Half-Precision FloatingPoint	**1.0**	
Additional Floating-Point: Zfa	**0.1**	Frozen
RISC-V Advanced Interrupt Architecture: Ssaia, Smaia	**1.0**	
"Smrnmi" Standard Extension for Resumable Non-Maskable Interrupts	**0.4**	Draft
RISC-V State Enable Extension: Ssstateen	**1.0**	
RISC-V Code Size Reduction: Zcb	**1.0**	
PMP Enhancements for memory access andexecution prevention on Machine mode: Smepmp	**1.0**	
RISC-V Integer Conditional operations: Zicond	**1.0-RC2**	Frozen

115.1.3 Integer unit

The NOEL-V integer unit implements the RISC-V RV64GCH and RV32GCH specification, including hardware multiply and divide instructions, floating-point instruction, compressed instruction, cryptography instruction, the hypervisor extension and also sub-sets of the bit-manipulations extension. The dual-issue pipeline consists of 7 stages with separate instruction and data cache interfaces (Harvard architecture).

115.1.4 Level 1 cache

NOEL-V has separate instruction and data caches, each with a 64 bit interface towards the CPU pipeline. The instruction cache maintains one valid bit per cache line and buffer forwarding during line-refill to minimize refill latency. The data cache has one valid bit per cache line, uses write-through policy and implements a four double-word write-buffer. Bus-snooping on the AHB bus is used to maintain cache coherency for the data cache. Both data and instruction caches supports up to 16 KiByte size and 4-ways set associativity. The cache line size is 32 bytes and replacement policy is LRU.

[Note: the size of data and instruction cache depends on the standard configuration selected.]

115.1.5 Floating-point unit

The NOEL-V floating point unit implements the RISC-V F and D extensions, i.e. single and double precision floating point operations. A fully pipelined high-performance GRFPUnv or an iterative nanoFPUnv version can be configured. Note that the GRFPUnv is provided separately.

115.1.6 Memory management unit

A Memory Management Unit is integrated into the NOEL-V cache controller. It implements the full RISC-V specification, and provides mapping between multiple 64/32-bit virtual address spaces and physical memory. A hardware table-walk is implemented, and the MMU has separate fully associative TLBs for instructions and data (and G-stage when hypervisor is available).

[Note: For RV64 only Sv39 (i.e. 39 bit virtual addressing using a three level page table) is available. RV32 only supports Sv32.]

115.1.7 Physical Memory Protection unit

The NOEL-V supports the standard RISC-V PMP. This allows for region based memory protection, whether an MMU is available or not.

115.1.8 Virtualization support

The NOEL-V implements the Hypervisor extension. This extension is enabled for the HPP and GPP standard configurations. All mandatory and some optional features of the extension are implemented, including the extension of the MMU to support two-stage address translation (with hardware page table walk).

[Note: The virtualization currently only supports RV64 Sv39x4. Currently the PLIC interrupt controller do not support directly assignments of interrupts to guests virtual machines. This will be added in future releases.]

115.1.9 On-chip debug support

The NOEL-V subsystem includes functionality to allow non-intrusive debugging on target hardware. Through a debug module interface, full access to all processor registers is provided. The debug interface also allows single stepping, setting hardware breakpoints, and instruction tracing. An internal trace buffer can monitor and store executed instructions, which can later be read out via the debug interface.

Interfaces to access the debug functionality are connected using a dedicated AHB port that does not interfere with normal operation.

115.1.10Interrupt interface

A core local interrupt controller is included in the subsystem to support local interrupts (which is described in the RISC-V privileged architecture standard v1.10). This controller supports software interrupts, the timer interrupts, and the external interrupt (generated via the platform interrupt controller PLIC).

The new RISC-V Advanced Interrupt Architecture (AIA) and the RISC-V Advanced Core Local Interruptor (ACLINT) are also supported. This includes both the Incoming MSI Controller (IMSIC) and Advanced Platform-Level Interrupt Controller (APLIC).

Core Local Interrupt controller register:

*Table 2229.*AHB Status registers

Address offset	Registers
0x0000	MISP Software interrupt pending (hart 0)
0x0004	MISP Software interrupt pending (hart 1)
...	...
0x4000	MTCMPL M-Timer compare low (hart 0)
0x4004	MTCMPH M-Timer compare high (hart 0)
0x4008	MTCMPL M-Timer compare low (hart 1)
0x400C	MTCMPH M-Timer compare high (hart 1)
...	...
0xBFF8	MTIMEL M-Timer (low)
0xBFFC	MTIMEH M-Timer (high)

*Table 2230.*CLINT MSIP Software interrupt pending register (0xE0000000)

31	1	0
RESERVED		MSIP
0		0
r		rw

31:1 Reserved, always 0

0 (MSIP) Machine mode software interrupt

*Table 2231.*CLINT MTCMPL M-Timer compare (low) register (0xE0004000)

31	0
MTIMECMP-LO	
0	
rw	

31:0 M-timer low compare value

*Table 2232.*CLINT MTCMPH M-Timer compare (high) register (0xE0004004)

31	0
MTIMECMP-HI	
0	
rw	

31:0 M-timer high compare value

*Table 2233.*CLINT MTIMEL M-Timer (low) register (0xE000BFF8)

31	0
MTIME-LO	
0	
rw	

*Table 2233.*CLINT MTIMEL M-Timer (low) register (0xE000BFF8)

31:0	M-timer low value

*Table 2234.*CLINT MTIMEH M-Timer (high) register (0xE000BFFC)

31	0
MTIME-HI	
0	
rw	

31:0	M-timer high value

The subsystem also includes a platform interrupt controller (PLIC, following the RISC-V Platform-Level Interrupt Controller Specification). The controller supports multiple harts, 32 interrupt sources, and 8 priority levels. Fore each hart, 4 context are defined.

*Table 2235.*PLIC context maping

Context	Hart interrupt mapping
0	Hart 0 machine-level external interrupts
1	Hart 0 supervisor-level external interrupts
2	Hart 0 Reserved
3	Hart 0 Reserved
4	Hart 1 machine-level external interrupts
5	Hart 1 supervisor-level external interrupts
6	Hart 1 Reserved
7	Hart 1 Reserved
8	...

115.1.11Standard peripherals

The NOELVSYS block includes standard APBUART and GPTIMER peripherals as are typically expected by software.

115.1.12AMBA interface

An AMBA 2.0 AHB bus is included in the subsystem. Each processor's cache system implements an AMBA AHB master, to load and store data to/from the caches, that is connected to the AHB bus. During line refill, incremental bursts are generated to optimise the data transfer. The AMBA interface can be configured to either 64 or 128-bit data width.

External slaves (memory controllers) and masters (peripheral DMA) are connected to the subsystem via ports. The external AHB slaves may either be narrow (support maximum 32-bit width) or wide (support full width of AMBA bus).

An APB controller is also included in the subsystem for the internal peripherals, and additional external peripherals may be connected via a subsystem port.

115.1.13 Power-down mode

The WFI (Wait For Interrupt) instruction is implemented either as a NOP or to stall the pipeline until a interrupt is asserted. During the pipeline stall, the clock to the processor can be gated to put the processor in a power-down mode.

115.1.14 Multi-processor support

To support Multiprocessor systems the A (Atomic) extension is implemented. The write-through L1-caches combined with the snooping mechanism guarantees memory coherency in shared-memory systems.

115.2 NOEL-V integer unit

115.2.1 Overview

The NOEL-V integer unit contains the main processor pipeline and implements the integer part of the RISC-V instruction set. The implementation is focused on high performance and has the following main features:

- 7-stage dual issue instruction pipeline
- Separate instruction and data cache interfaces
- Hardware multiply/divide unit and cryptography unit
- Dynamic branch prediction, branch target buffer and return address stack
- Four ALUs supporting all standard operations, two of them late in the pipeline to reduce stalls

Figure 305 shows a block diagram of the integer unit (and FPU integration).

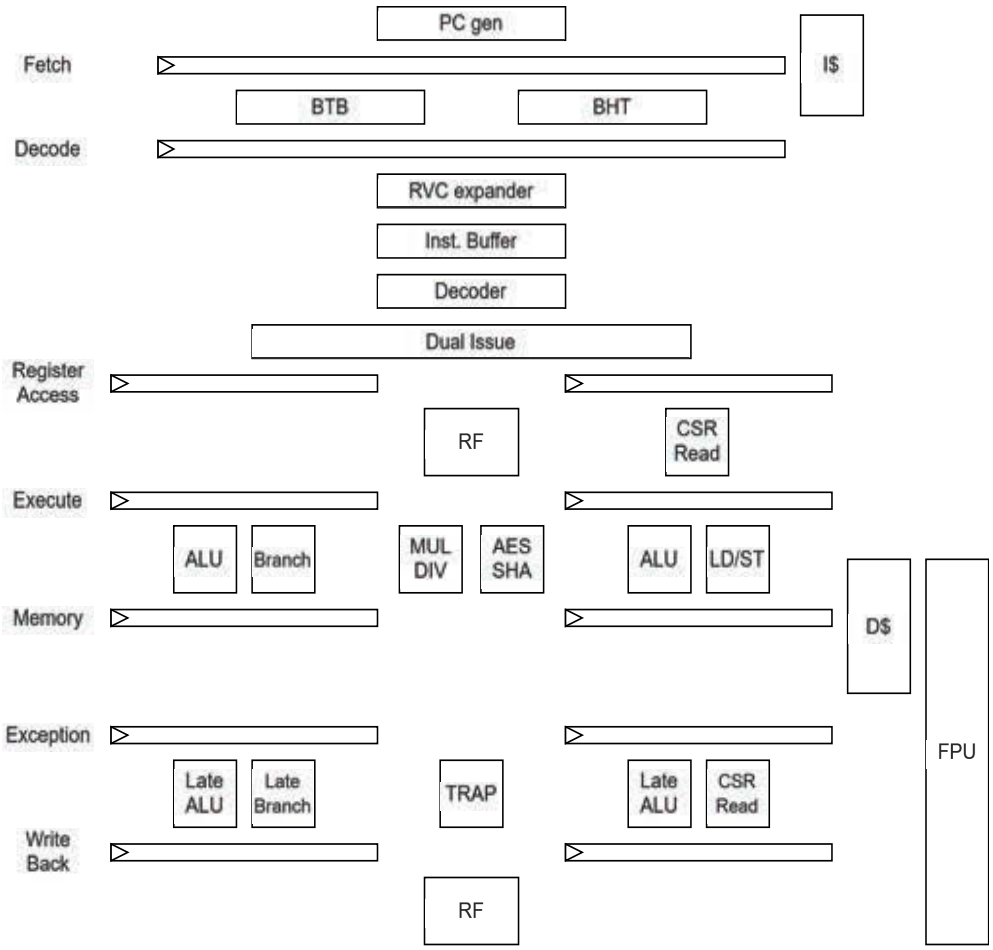


Figure 305. NOEL-V integer unit datapath diagram

115.2.2 Instruction pipeline

The NOEL-V integer unit uses a dual-issue instruction pipeline with seven stages:

1. FE (Instruction Fetch): A 64-bit word (holding two to four instructions) is fetched from the instruction cache. The instructions are valid at the end of this stage and are latched inside the IU.
2. DE (Decode): Two instructions at a time are decoded and the branch target addresses are generated. It is determined if dual issue is possible and which instruction goes into which lane of the following stages.
3. RA (Register access): Operands are read from the register file or from internal data bypasses. This is also where pipeline bubbles are inserted to enable dependencies to be satisfied.
4. EX (Execute): ALU, logical, and shift operations are performed in both lanes. Some operations (CLMUL) from the bitmanipulation extension is only supported in one lane. For memory operations (e.g., LD) and for JAL/JALR, the address is generated.
5. ME (Memory): Read data is received from the data cache if accessed. Store data is generated and fed into the data cache inputs at this time.
6. XC (Exception) Exceptions and interrupts are resolved. For cache writes, the data update takes place inside the cache memory on this cycle. Late ALUs and branch unit with full standard functionality are included in this stage to allow delayed dependency resolution.
7. WR (Write-back): The results of ALU and cache operations are written back to the register file.

115.2.3 Interlocks

Additional conditions that can extend an instruction's duration in the pipeline are listed below.

Branch interlock: A mispredicted control flow change will lead to annulation of the predicted instructions, and fetching of the non-predicted target, losing the cycles gained by prediction.

Memory access hold: During cache miss processing or when blocking on the store buffer, the pipeline will be held still until the data is ready, effectively extending the execution time of the instruction causing the miss by the corresponding number of cycles. Note that since the whole pipeline is held still, hold cycles will not mask other delays.

Divide/remainder: These operations will hold the pipeline until finished.

FPU instructions generating integer results: These operations will hold the pipeline until finished.

CSR fflags access: Accessing the FPU flags must wait for all FPU instructions to complete first.

Unresolvable dependencies: When trying to use data results before they are available, the instruction pair will be delayed as needed. The late ALUs in the pipeline will often help avoid this.

Back-to-back store: Smaller than 32 bit writes are not allowed back-to-back.

CSR read: A read from a recently written CSR must wait until the former completed.

Memory access after CSR write: Some CSR writes may affect following a load/store, so it must wait.

Fence instructions, CSR writes that can affect following instruction, and instructions that change privilege mode, will cause a pipeline flush.

115.2.4 Branch predictor and branch target buffer

The pipeline implements a branch predictor in the decode stage. The predictor is configurable at build-time. The pipeline can also be switched over to a static always-taken prediction mode to improve timing predictability.The branch predictor is a two-level per-address adaptive, using per-address pattern history tables (PAp).

A branch target buffer is implemented to predict branch targets in the fetch stage to avoid a one-cycle gap between branches and the target instruction. There is also a return-address-stack to improve handling of subroutine calls.

115.2.5 Late ALU

Late ALUs and a late branch unit are implemented to allow ALU and branch operations normally done in the EX stage to be deferred to the XC state. This allows masking the load and multiply to ALU/branch operation latency, as well as allowing some dependent instructions to be paired.

115.2.6 Hardware breakpoints

A trigger module (following the RISC-V Debug Specification) is implemented to support Match Control triggers and a instruction count trigger. The core has support for two hardware breakpoint registers. The Match control trigger is able to match on instruction, PC, read/write address or read/write data. Currently only action 1 (Enter Debug Mode) is supported. Read/write data triggers will fire after the data access. All other triggers will be fired before the trigger event is executed.

115.2.7 Instruction trace buffer

The instruction trace buffer consists of a circular buffer that stores executed instructions. This is accessed only through the processor's debug port via the Debug Module. When the CPU is running, the following information is stored in real time, without affecting performance:

- Instruction address and opcode
- Instruction result
- Load/store data
- Trap/Interrupt information
- Privilege mode
- 32-bit time tag

115.2.8 Processor reset operation

The processor is reset by asserting the RESET input for at least 4 clock cycles. The following table indicates the reset values of a subset of the registers which are affected by the reset.

*Table 2236.*Processor reset values

Register	Reset value
PC (program counter)	0xC0000000

By default, the execution will start from address 0xC0000000. This can be changed from the debug port before the integer unit is started.

115.3 Dual issue pairing and Optimization Guide

This chapter assumes that a full standard configuration of NOEL-V is used. That is, with both pipelines active and containing late ALUs and a late branch unit, as well as the pipelined FPU.

115.3.1 The NOEL-V pipeline

The NOEL-V instruction pipeline (see 305) is capable of issuing and retiring two instructions per cycle, compressed or not. Compressed instructions obviously use less instruction cache space, but that is their only impact on performance. The discussion below does not concern itself with this.

There is currently no instruction fusion. If that is eventually added, it is possible that compressed instructions might be advantageous.

Instructions are always fetched 8 bytes at a time, from aligned addresses. This does not mean, however, that two instructions that were fetched together will necessarily be paired in the pipeline. There may be buffered instructions available, for example due to the use of compressed instructions.

The integer side of the pipeline is currently strictly in-order - no instruction will ever "pass by" another during execution, invisibly or not. Only a few instructions take more than one cycle to actually execute, so the forwarding mechanism in the pipeline will usually allow instructions in an imme-

diately following pair to depend on results from the previous pair without penalty. Even dependencies within a pair can often be handled by having the second instruction execute in a late ALU.

The pipelined FPU has only a single “lane” (though there are actually three separate internal pipelines) and since most operations take more than one cycle to execute, there is only limited forwarding. FPU instructions actually pass through the integer pipeline, notifying the FPU (to issue) when they pass the execute stage, and when they reach the write-back stage (to retire/cancel). Inside the FPU, instructions will issue in-order once their dependencies can be met. But unlike on the integer side, they may complete out-of-order (both with respect to each other and the integer side). The FPU is capable of queuing a couple of instructions, so as not to stall the integer side immediately when internal dependencies block immediate issue. Data transfer from the FPU to the integer side will, however, force an immediate stall since the data is returned directly in the execute stage. Transfers the other way will never stall the integer side, and happen from the exception stage.

115.3.2 Simple optimization

While the details will be described below, they can get rather complicated. Often it is enough to follow a few relatively simple rules for reasonable performance.

- It never hurts to put more separation between dependent instructions.
- A single instruction as separation between dependent ones is usually enough for non-FPU.
- Even no separation is often OK, but avoid doing it for more than two instructions in a row.
- Only one branch or jump in a pair.
- Only one load or store in a pair.
- Avoid load in a pair directly after a pair that does a store.
- Avoid stores of smaller than 32 bit data in successive pairs.
- Avoid FPU instructions without separation (currently does not help, but will later).
- Four pairs of separation is often OK for FPU dependencies.
- Dependencies on FPU divide and square root ought to be some ~25 pairs away.
- Avoid integer division - it does not (currently) help if dependencies are far away.

115.3.3 Optimization details

The following text refers to RISC-V opcodes and NOEL-V internal instruction types.

See tables 2242 and 2243 for descriptions of these.

115.3.4 Pairing conflicts

Pairing of instructions is handled in the decode stage. After two instructions are fetched and/or removed from the fetch buffer, they are initially assumed to be paired - the first instruction in lane 0 and the second in lane1. They will stay together through the pipeline unless there is a specific reason to disallow this pairing, which would split them apart (the second one then becoming available for the next pairing).

Paired instructions will be swapped automatically between the two lanes if at least one of them can only be handled in a specific pipeline.

*Table 2237.*Forced lanes

Instruction type	Lane	Comment
Anything that touches memory	0	
Branches	1	
FPU operations	0	Currently. May change in the future, except for FPU load / store.
CSR accesses	0	Currently. May change in the future.
Carry-less multiplication (clmul / cmulr / cmulh)	0	Can only be done in the early ALU of lane 0 (could be a build option). Other special ALU operations like this may be added later.

Most of the checks for conflict refer only to the two instructions in the proposed pair. But there is also a check for a dependency on a late ALU operation in the preceding pair.

*Table 2238.*Register name explanation

Register	Meaning
Rd^0 / Rd^1	Destination register in lane 0/1.
$Rs1^0$ / $Rs1^1$	Source register 1 in lane 0/1.
$Rs2^0$ / $Rs2^1$	Source register 2 in lane 0/1.

Table 2239 shows pairings that are not allowed for a full-featured NOEL-V. This is before swapping lanes, so it needs to check for any necessary swapping and disallow pairings that would make such impossible.

*Table 2239.*Pairing disallowed

Lane 0 (1st instruction)	Lane 1 (2nd instruction)
LD / *ST* / *AMO*	Anything forced to lane 0
JAL / *JALR*	Anything
BRANCH	Anything except OP / OP-32 / OP-IMM-32 / OP-IMM / LUI
ecall / ebreak / mret / sret	Anything
sfence.vma / hfence.vvma / hfence.gvma	Anything
wfi	Anything
CSR write that may affect instruction execution or flush pipeline	Anything
CSR write to FPU flags	FPU (if FPU lane /= CSR lane, any variant except load / store)
CSR access	Anything forced to lane 0
CSR write to MEPC	mret
CSR write to SEPC	sret
fence / fence.i	Anything
MUL	*MUL*
CRYPTO	*CRYPTO*
FPU (not load / store)	CSR access to FFLAGS / FCSR
FPU (lane 0)	Anything forced to lane 0
div / rem (any variant)	Anything
FPU (any variant)	Anything
ALU_SPECIAL	Anything
Anything	CSR write that may affect instruction execution or flush pipeline
Anything with destination (Rd^0)	$Rs1^1 = Rd^0$ or $Rs2^1 = Rd^0$, for SYSTEM / MISC-MEM / JALR / *custom-0* / OP-FP (any variant using integer Rs1 or Rs2) [2nd dependent on 1st and not simple ALU operation.]
Anything with destination (Rd^0)	$Rs1^1 = Rd^0$, for *LD* / *ST* / *AMO* [Address]
MUL (Rd^0)	$Rs2^1 = Rd^0$, for *ST* / *AMO* [Data]
OP / OP-32 / OP-IMM / OP-IMM-32 / LUI / AUIPC (Rd^0)	$Rs1^1 = Rd^0$ or $Rs2^1 = Rd^0$, for *MUL*
CSR Rd^0	$Rs1^1 = Rd^0$ or $Rs2^1 = Rd^0$), for any
Anything with destination (Rd^0)	$Rd^1 = Rd^0$, when either instruction is *LD* or CSR access
OP / OP-32 / OP-IMM / OP-IMM-32 / LUI / AUIPC (Rd^0) and instruction using late ALU in previous pair (RA stage), where lateRA-Rd^0 = $Rs1^0$ / $Rs2^0$ or lateRA-Rd^1 = $Rs1^0$ / $Rs2^0$	$Rs1^1 = Rd^0$ or $Rs2^1 = Rd^0$, for any [2nd dependent on 1st (which would have been handled by doing 2nd late), and late ALU is impossible due to 1st being dependent on late in RA (and thus also late).]

115.3.5 Pipeline hold and late ALU / branch

n the register access stage, it is decided whether the pipeline needs to hold (insert bubbles) or not. The "redirection" to the late ALUs or the late branch unit is also handled here.

Table 2240 shows the cases where the pairing itself requires one of the instructions to be executed late, and when earlier things in the pipeline forces a hold.

*Table 2240.*Pipeline hold and late ALU / branch, PART I

Dependency	Resolution
Dependent *ALU* in lane 1 (no swap) paired with *ALU* / *LD* / *MUL* / *FPU* / *CRYPTO*	late ALU (dp)
Dependent *BRANCH* paired with *ALU* / *MUL* / *LD* / *FPU* / *CRYPTO*	late branch (dp)
Dependent *ALU* / *BRANCH* in pair where ALU must be late	hold, not late ALU, not late branch
LD / *ST* close after write to mstatus / mstatush / sstatus / hstatus / pmpcfg / pmpaddr	hold
LD / *ST* close after write to mie / sie / uie / hie / hgeie / mideleg / sideleg / hideleg	hold
ALU_SPECIAL can never be late	hold, not late ALU
CSR access	late CSR
Reading from CSR that is being written anywhere in the pipeline	hold
Reading from CSR that is dependent on other CSR written anywhere in the pipeline	hold
Instruction close after CSR write to pipeline flush capable things	hold
Instruction close after TLB or instruction fence	hold
CSR accessing FPU flags while FPU is doing something	hold
FPU close after CSR write to FPU related things	hold
FPU instruction when FPU is not ready	hold
FPU instruction that uses data from IU dependent on something in pipeline	hold
Instruction in RA dependent on late *ALU* in MEM (except *ST* data)	hold

Table 2241 shows the troublesome dependency situations between the register access stage and the execute stage.

*Table 2241.*Pipeline hold and late ALU / branch, part II

Register access stage	Execute stage	Resolution
Dependent *BRANCH*	*LD*	late branch
Dependent *BRANCH*	late *ALU*	late branch
Dependent *ALU*	*LD*	late ALU
Dependent *ALU*	*MUL*	late ALU
Dependent *ALU*	late *ALU*	late ALU
ST	late *BRANCH* (swapped)	hold, not late branch
Dependent *ST* (data)	late *ALU* (swapped)	hold, not late ALU
Dependent *ST* (data)	late *CSR* (swapped)	hold, not late ALU
Dependent operation that cannot be late	late *ALU*	hold
Dependent, not *ALU* / *BRANCH* (except *ST* data)	*LD*	hold
Dependent, not *ALU* (except *ST* data)	*LD*	hold
Any instruction	*MUL*	hold
ST (when size of either store is less than 32 bits)	*ST*	hold
LD	*ST*	hold
ST / *AMO* / TLB fences	late *BRANCH*	hold
LD (not *AMO*)	late *BRANCH*	speculative
LD / ST	CBO instruction	hold

*Table 2242.*RISC-V main opcodes

Opcode	Name	Description
00000 11	LOAD	Integer load
00001 11	LOAD-FP	Floating point load
00010 11	*custom-0*	Used for NOEL-V diagnostic operations.
00011 11	MISC-MEM	Mainly fences (fence, fence.tso, fence.i, pause)
00100 11	OP-IMM	“Normal” operations with immediate value as second source.
00101 11	AUIPC	PC-relative address set
00110 11	OP-IMM-32	Like OP-IMM, but for 32 bit operations in RV64.
00111 11	*48b*	
01000 11	STORE	Integer store
01001 11	STORE-FP	Floating point store
01010 11	*custom-1*	
01011 11	AMO	Atomic operations
01100 11	OP	“Normal” operations
01101 11	LUI	Upper integer value set
01110 11	OP-32	Like OP-REG, but for 32 bit operations in RV64.
01111 11	*64b*	
10000 11	FMADD	Floating point fused multiply-add
10001 11	FMSUB	Variant of fused multiply-add
10010 11	FMNSUB	Variant of fused multiply-add
10011 11	FMNADD	Variant of fused multiply-add
10100 11	OP-FP	Other floating point operations
10101 11	OP-V	Vector operations (not available in NOEL-V)
10110 11	*custom-2/rv128*	
10111 11	*48b*	
11000 11	BRANCH	Conditional branches
11001 11	JALR	Indexed jump with return address
11010 11	*reserved*	
11011 11	JAL	Fixed jump with return address
11100 11	SYSTEM	Most “system” operations, including CSR accesses.
11101 11	*reserved*	
11110 11	*custom-3/rv128*	
11111 11	*>=80b*	

*Table 2243.*Internal instruction types

Type	Description	Details
ALU	Simple register to register operations (see *ALU_SPECIAL*)	OP / OP-32 / OP-IMM / OP-IMM-32 / LUI / AUIPC (except *MUL*) and CSR accesses (from SYSTEM)
ALU_ SPECIAL	Register to register operations that are only possible in the lane 0 early ALU.	Some of OP (see *ALU*) (currently clmul / clmulh / clmulr).
AMO	Atomic operations and lr / sc	AMO
BRANCH	Branch instruction variants	BRANCH
DIAG	Diagnostic custom instructions	*custom-0*
FPU	FPU instructions returning integer results	Some of OP-FP
JAL	The jal instruction	JAL
JALR	The jalr instruction	JALR
LD	Any load, including atomic instructions that load. Also load type custom instructions.	LOAD / LOAD-FP / AMO and some of *custom-0*
MUL	Any variant of mul / div / rem	Some of OP / OP-32
CRYPTO	Any AES / SHA instruction	OP / OP-32 / OP-IMM / OP-IMM-32
ST	Any store, including atomic instructions that store. Also store type custom instructions.	STORE / STORE-FP / AMO and some of *custom-0*

115.4 Cache system

115.4.1 Overview

The NOEL-V processor pipeline implements a Harvard architecture with separate instruction and data buses, connected to two separate caches. As long as the execution does not cause a cache miss, the cache controllers can serve one beat of an instruction fetch and one data load/store per cycle, keeping the pipeline running at full speed.

On cache miss, the cache controller will assert a hold signal, freezing the IU pipeline. After delivering the data, the hold signal is again lifted so that execution continues. The miss processing is handled by a state machine common to both the instruction and data caches.

Another important component included in the data cache is the write buffer, allowing stores to proceed in parallel to executing instructions.

Cachability (memory areas that are cachable) for both caches is controlled through the AHB plug&play address information or using a VHDL generic, see section 115.4.14.

115.4.2 Cache operation

Each cache controller has two main memory blocks, the tag memory and the data memory. At each address in the tag memory, a number of cache entries, ways, are stored for a certain set of possible memory addresses. The data memory stores the data for the corresponding ways.

For each way, the tag memory contains the following information:

- Valid bits saying if the entry contains valid data or is free. Both caches have a single valid bit for each cache line.
- The tag, all bits of the cached memory address that are not implied by the set

When a read from cache is performed, the tags and data for all cache ways of the corresponding set are read out in parallel, the tags and valid bits are compared to the desired address and the matching

way is selected. In the hit case, this is all done in the same cycle to support the full execution rate of the processor.

In the miss case, the cache will at first deliver incorrect data. However on the following cycle, a hold signal will be asserted to stall the pipeline and prevent it from proceeding with that data. After the miss has been processed, the correct data is injected into the pipeline using a memory data strobe (mds) signal, and afterwards the hold signal can be released. If the missed address is cachable, then the data read in from the cache miss will be stored into the cache, possibly replacing one of the existing ways.

The instruction cache implements a buffer forwarding scheme to reduce instruction cache miss latency. While the cache line is being read in from the AHB bus and stored into the cache memories, the data is stored temporarily in a flip flop buffer inside the cache controller. The cache can serve instructions to the pipeline directly out of this partially filled buffer. This works even in case of branches as long as the instruction flow stays on the same cache line.

115.4.3 Cache configuration

Least-recently-used (LRU) replacement is used for the caches. This maintains the order of usage for each set in the cache and replaces the one which was used least recently. The LRU information needs to be updated on every cache hit and is therefore not stored with the tags but in separate flip flops. The cache configuration (size and associativity) for the standard CPU configuration is described in Table 2225.

115.4.4 Address mapping

The addresses seen by the CPU are divided into tag, index and offset bits. Because the index is used to select the set in the cache, only a limited number of cache lines with the same index part can be stored at same time in the cache. The tag is stored in the cache and compared upon read.

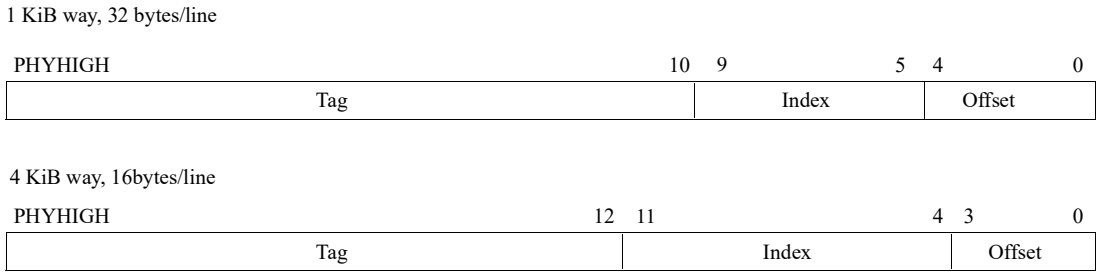


Figure 306. Cache address mapping examples

PHYHIGH: Highest bit in the physical address (equal 31 for system with 32-bit physical address).

115.4.5 Data cache policy

The data cache employs a write-through policy, meaning that every store made on the CPU will propagate, via the write buffer, to the bus and there are no “dirty” lines in the cache that have not yet been written out, apart from what is in the buffer. The store will also update the cache if the address is present, however a new line will not be allocated in the case when it is not.

*Table 2244.*NOEL-V Data caching behavior

Operation	In cache	Cacheable	Bus action	Cache action	Load data
Data load	No	No	Read	No change	Bus
	No	Yes	Read	Line allocated/replaced	Bus
	Yes	-	None	No change	Cache
Data store	No	No	Write (via buffer)	No change	(N/A)
	No	Yes	Write (via buffer)	No change	(N/A)
	Yes	-	Write (via buffer)	Data updated	(N/A)

115.4.6 Write buffer

The data cache contains a write buffer able to hold up to four 8, 16, 32, or 64-bit writes. For half-word or byte stores, the stored data is replicated into proper byte alignment for writing to a word-addressed device.

The write is processed in the background, so the system can keep executing while the write is being processed. The write buffer acts as a FIFO where new stores can be added continuously to allow one store per cycle throughput. Only “regular” stores that hit in the TLB are handled through the write buffer, other types of writes will be handled separately.

While a store is ongoing, any following instruction or data access that requires bus access will block until the write buffer has been emptied. Loads served from cache will however not block, due to the cache policy used there can not be a mismatch between cache data and store buffer.

Since the processor executes in parallel with the write buffer, a write error will not cause an exception to the store instruction. Depending on memory and cache activity, the write cycle may not occur until several clock cycles after the store instruction has completed. Since there is nothing that can be done about a late write buffer error, an NMI interrupt will be taken.

[Note: The NMI interrupt will be added in future releases.]

115.4.7 Operating with MMU

The MMU functionality is integrated into the cache controller. When MMU is enabled, the virtual addresses seen by the running code no longer correspond directly to the physical addresses on the AHB bus. The cache will translate each access’s virtual address to its physical address in parallel with looking up the tag for that access.

The cache uses a virtually-indexed-physically-tagged (VIPT) approach, where the tags are based on the physical addresses but the address into the cache memory is based on the virtual address.

115.4.8 Snooping

The data cache includes support for AHB bus snooping. The AHB bus that the processor is connected to is monitored for writes from other masters to an address which is in the cache. If a write is done to a cached address, that cache line is marked invalid and the processor will be forced to fetch the (new) data from memory the next time it is read.

An extra tag memory, storing separate physical tags, is added to allow comparing with the physical address on the AHB bus.

The processor can snoop on itself in order to detect MMU updating page table entries, and to flush out possible aliases in the cache.

Snooping requires the way size of the cache to be equal or smaller than the MMU page size, otherwise the index into the physical and virtual tag RAM:s may not match, resulting in aliasing problems.

115.4.9 Enabling and disabling cache

Both instruction and the data cache are disabled after reset. They can be enabled by writing to the custom CSR (CCTRL cache control register). See section 115.12.10. The cache tags are initialized automatically after reset to ensure that all tags are invalid.

115.4.10Cache freeze

Each cache can be in one of three modes: disabled, enabled and frozen. If disabled, no cache operation is performed and load and store requests are passed directly to the memory controller. If enabled, the cache operates as described above. In the frozen state, the cache is accessed and kept in sync with the main memory as if it was enabled, but no new lines are allocated on read misses. The cache mode is set via the CCTRL custom CSR.

115.4.11Flushing

The instruction cache is flushed when a FENCE.I is executed. The data cache (and also the instruction cache) can be flushed via the custom CSR (CCTRL). See section 115.12.10. Cache flushing takes one clock cycle per cache set, during which the IU will be halted. When the flush operation is completed, the cache will resume the state (disabled, enabled or frozen) indicated in the cache control register (CCTRL custom CSR). Data cache lines can also be invalidated/flushed using the instructions defined in the cache management operations extension (cbo.inval/clean/flush). The invalidate and flush instruction invalidate the cache line (no data write back due to write-through L1 cache). The clean instruction is handled as a NOP.

115.4.12Diagnostic access

The cache tag and data contents can be directly accessed for diagnostics via various custom instructions, see section 115.14.

115.4.13AMBA interface

The NOEL-V cache controller has one AHB master interface to the surrounding system. The types of AMBA accesses supported and performed by the processor depend on the accessed memory area's cachability, the maximum bus width, if the corresponding cache is enabled, and if the accessed memory area has been marked as being on the wide bus.

Cachable instructions are fetched with a burst of 32-bit accesses, or 64- or 128-bit accesses depending on the cache line size and the AHB bus width.

115.4.14Cachability

Cachability for both caches can be controlled through the AHB plug&play address information, or set manually via the *cached* VHDL generic.

For plug'n'play based cachability, the memory mapping for each AHB slave indicates whether the area is cachable, and this information is used to (statically) determine which access will be treated as cacheable. This approach means that the cachability mapping is always coherent with the current AHB configuration.

When the *cached* VHDL generic is not zero, it is treated as a 16-bit field, defining the cachability of each 256 MiB address block on the AMBA bus. For example, a value of 16#00F3# will define cachable areas in 0 - 0x20000000 and 0x40000000 - 0x80000000.

In order to access the plug'n'play information, the processor takes the ahbso vector as input. Only the static hconfig signals are used, so the use of this input will be eliminated through constant propagation during synthesis.

115.4.15AMBA access size

Cacheable data is fetched in a burst of 64- or 128-bit accesses, depending on the cache line size and AHB bus width.

The type of AMBA accesses used, and supported by the processor, for a memory area depends on the area's cachability and the values of the *wbmask* and *busw* VHDL generics.

The area which supports 64- or 128-bit access is indicated in the *wbmask* VHDL generic. This VHDL generic is treated as a 16-bit field, defining the 64/128-bit capability of each 256 MiB address block on the AMBA bus. A value of 16#00F3# will thus define areas in 0 - 0x20000000 and 0x40000000 - 0x80000000 to be 64/128-bit capable. The maximum access size to be used in the area(s) marked with WBMASK is determined by the *busw* VHDL generic.

Store instructions result in an AMBA access with size corresponding to the executed instruction. The table below indicates the access types used for instruction and data accesses depending on cachability, wide bus mask (wbmask), and cache configuration.

Processor operation	**Accessed memory area is 32-bit only, wbmask(address) = 0**			**Accessed memory area is on wide bus wbmask(address) = 1**		
	Area not cacheable[1]	**Area is cacheable[1]**		**Area not cacheable[1]**	**Area is cacheable[1]**	
		Cache enabled[2]	**Cache disabled**		**Cache enabled[2]**	**Cache disabled**
Instruction fetch	Burst of 32-bit read accesses	Burst of 32-bit read accesses		Burst of 64- or 128-bit accesses[3]		
Data load <= 32-bit	Read access with size specified by load instruction	Burst of 32-bit accesses	Read access with size specified by load instruction	Read access with size specified by load instruction	Burst of 64- or 128-bit accesses[3]	Read access with size specified by load instruction
Data load 64-bit	**Length-2 burst of 32-bit read accesses**	**Burst of 32-bit read accesses**	**Length-2 burst of 32-bit read accesses**		Burst of 64- or 128-bit accesses[3]	Single 64-bit read access
Data store <= 32-bit	Store access with size specified by store instruction.					
Data store 64-bit	**Length-2 burst of 32-bit stores**			64-bit store access		

[1] Cachability is determined by the *cached* VHDL generic, if *cached* is zero then cachability is determined via AMBA PnP.

[2] Bus accesses for reads will only be made on L1 cache miss.

[3] 64- or 128-bit accesses depending on *busw* VHDL generic.

115.4.16Error handling

An AHB ERROR response received while fetching instructions will normally cause an instruction access fault exception. However, if this occurs during streaming on an address that is not needed, the cache controller will just not set the corresponding valid bit in the cache tag. If the IU later fetches an instruction from the failed address, a cache miss will occur, triggering a new access to the failed address.

An AHB ERROR response while fetching data into the data cache will normally trigger a Load access fault exception. If the error was for a part of the cache line other than what was currently being requested by the pipeline, a trap is not generated and the valid bit for that line is not set.

An ERROR response during an MMU table walk will lead to a page fault exception, depending on which type of access that caused the table walk.

115.4.17Snoop port

For the snooping logic, the cache controller has an ahbsi input which listens to the processor AHB bus. For correct function, this must be tied to the same AHB bus as the master interface. It is not possible to snoop on another bus or to add extra pipeline registers to the snoop port, because the snoop logic must be in sync with the master interface.

115.5 Memory management unit

115.5.1 Overview

The memory management unit is integrated into the cache controller. The MMU provides address translation of both instructions and data via page tables stored in memory. When needed, the MMU will automatically access the page tables to calculate the correct physical address. The latest translations are stored in a special cache called the translation lookaside buffer (TLB), also sometimes referred to as a Page Descriptor Cache (PDC). The MMU also provides access control, making it possible to “sandbox” unpriviledged code from accessing the rest of the system.

[Note: For RV64 only Sv39 (i.e. 39 bit virtual addressing using a three level page table) is available. RV32 only supports Sv32.]

115.5.2 MMU/Cache operation

The caches operate with physical address mapping, although the indexing can be done with the relevant part of a virtual address since it is the same. The cache tags store the physical address and each access from the pipeline is translated from virtual to physical in parallel with the tag memory readout.

No additional clock cycles are needed by the MMU for instruction fetch, data fetch, or stores, as long as the virtual pages accessed are already cached inside the TLB. If there is a TLB miss, the page table must be traversed, resulting in a few AMBA read accesses and possibly one write-back operation.

See the RISC-V specification for the exact format of the page table.

An MMU page fault will generate the appropriate page fault exceptions. In case of multiple errors, the fault type values are prioritized as the specification requires. The cache and memory will not be modified on an MMU page fault.

115.5.3 Translation look-aside buffer (TLB)

Separate TLBs are implemented for instruction and data. The number of entries each TLB has depends on selected standard configuration. When the Hypervisor extension is enabled, a separate TLB is implemented for the G-stage address translation, the second stage of two-stage translation.

The TLBs are replaced using a pseudo-LRU algorithm. Each TLB entry has a single ‘visited’ bit that is set when that TLB entry is accessed, when all visited bits are set to 1 they are all cleared and the process starts over. When replacing an entry in the TLB, the lowest entry with the visited bit not set will be chosen.

115.5.4 Physical Memory Protection

PMP is configured from machine mode and makes it possible to, for example, enforce read/write/execution protection between separate tasks in a simple realtime system running in supervisor or user mode. It is also possible to enforce protection on machine mode code, by locking specific PMP entries, which cannot be undone without a system reset.

For further details, see the RISC-V documentation.

The number of PMP entries and granularity depends on selected standard configuration, see Table 2225. When MMU is included, PMP granularity cannot be smaller than the page size, 4 KiByte. Even without MMU, the granularity cannot be smaller than 8 bytes.

115.6 Floating-point unit

The NOEL-V floating point unit supports 32 and 64 bit floating point operations, according to the RISC-V standard which in turn is based on IEEE754-2008. It has its own register file and runs in parallel with the integer pipeline, except as necessary to pass results between the two units. Depending on configuration, it can be fully pipelined single-issue (GRFPUnv) or use a state machine to iteratively execute one operation at a time (nanoFPUnv).

As per the RISC-V standard, the normal IEEE754 state is kept in a CSR register - the FPU can not cause an exception (other than faulting memory accesses). Also per the standard, rounding mode can be specified statically in the instructions or dynamically via a CSR register.

115.6.1 Pipelined floating-point unit (GRFPUnv)

GRFPUnv is a high performance FPU, specially designed for the NOEL-V. It supports the RISC-V F and D extensions (single and double precision floating point), and is fully pipelined for normal FPU operations (a separate iterative unit does division and square root using Goldschmidt's algorithm). Operations always issue in-order, one at a time, but may complete out-of-order (besides for effects on execution time, this is invisible from the outside), again one at a time (except that operations returning integer results may complete at the same time as operations returning floating point results). There is no difference in performance between single and double precision operations (except for a couple of cycles in the Goldschmidt engine), but subnormal numbers have a 1-2 cycle penalty.

There is an implementation option to chose if a multiply can be done in one cycle in the target technology, or if two cycles are needed.

Implementation

In the NOEL-V, floating point instructions always pass all the way through the integer pipeline, but they are also sent to the FPU in the execution stage. For operations that give floating point results, the FPU will start working without affecting the integer pipeline. But if an integer result must be returned from the FPU, bubbles will be inserted after that stage until the result is available.

The FPU may temporarily stop accepting new operations if dependencies require an earlier operation to finish before a newer one can be started, or if the iterative Goldschmidt machinery is not yet available for a new operation. There is, however, a tiny "queue" in the FPU that enables it to accept a second operation while waiting, and only another one after that will necessitate the insertion of bubbles in the integer pipeline.

Data passes from the integer pipeline to the FPU without the former ever pausing, and eventually the integer pipeline informs the FPU about whether the instruction was committed or if something, for example misprediction, caused it to be annulled. Only operations that are reported as committed will have their results written to the FPU register file or otherwise used by the FPU. Annulled operations are removed from their place in the FPU.

For normal operations, when there are no dependencies, one floating point result can be produced every cycle. Latency varies depending on the type of operation and in some cases on the instruction mix, as well as on the values involved (subnormal numbers may need to be "normalized" early in the pipeline and subnormal results "denormalized" at the end, also some operations have special handling for NaN, infinity and zero). A diagram of the main FPU pipeline can be seen below. The FPU controller uses another two stages before an operation gets there, and operations that return integer values are done in a separate pipeline. The full latency will only be visible when an FPU operation is followed by a transfer of its result to the integer pipeline. For the latency of some directly dependent operations, see the table below.

An instruction that expects straight binary floating point data from the FPU (such as a store) will have to wait 2 cycles, in the absence of dependencies, while other results returned to the integer pipeline will take 4 cycles.

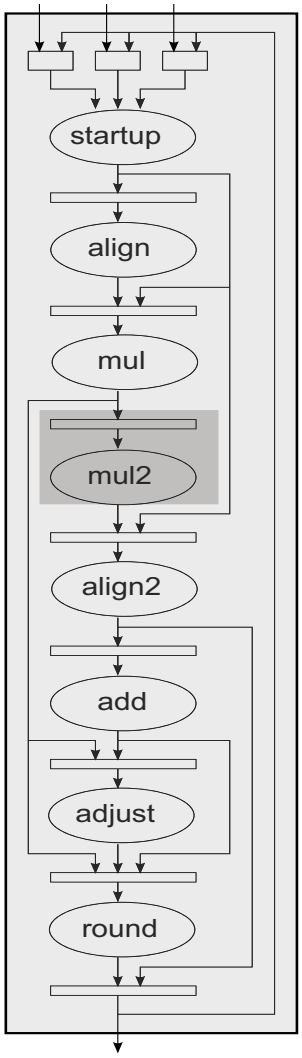


Figure 307. GRFPUnv pipeline block diagram

The "align" and "adjust" stages can often be skipped when no subnormals numbers are involved. Non-multiplies can often skip "mul", while multiplies can often skip "align2" and "add". Some operations can also skip "round", and operations that involve NaN, infinity or zero may skip some steps.

Notes on performance

Since NOEL-V is an in-order processor and GRFPUnv also always does in-order issueing of operations, it is beneficial to keep dependent operations apart. How far apart instructions must be to not cause unnecessary delays depend on the details. A general rule of thumb for GRFPUnv is that four cycles apart is often "good enough" (see below for more details), but with a dual-issue NOEL-V that is often more than four instructions (assuming some of them are non-FPU instructions).

Dependencies are mainly related to previous destinations being used as sources for later operations (Read after Write). However, there can also be a penalty for reusing a destination register (Write after Write).

Floating point operations that have been committed (they reached the end of the integer pipeline without being annulled) will always run to completion. This means that, for example, storing a floating point register value to the stack (such as on a context switch) can take a lot longer than expected. A square root might just have started executing, and a second one might be queued after it. The store of the destination register of the second square root could be waiting for dozens of cycles in the execution stage.

Table 2245. Throughput for consecutive independent instructions (tested with all the same)

Instruction	Description	Throughput
fsgn, fcvt.d.w, fmul, fadd, fmadd, fld, etc	Instructions with result written to the FPU register file	1
fmv.x.d, fsd, etc	Instructions that pass binary FPU register file data to the integer unit	2
fcvt.w.d, feq, etc	Other instructions that pass data to the integer unit	4
fdiv		21
fsqrt		26

The timings below assume that a single cycle multiplier is used, otherwise add one to the timing of operations that include a multiplication. Also, no subnormal numbers are involved, which could otherwise have added a cycle or two (depending on whether sources and/or results are subnormal).

Table 2246. Latency for consecutive dependent instructions

Instruction	Latency
fsgn fsgn fsgn...	2
fmul fmul fmul...	3
fadd fadd fadd...	4
fmadd fmadd fmadd...	5
fld fsd	7
fmadd eq	10

As the last two examples above show, latencies can be long when an instruction that returns an integer result depends on a directly preceding floating point instruction.

Copying data is best done using integer registers, where copying n elements can take n * 2 + 1 cycles by doing:

```
ld<times n> sd<times n>
```

That is, copying three elements:

```
ld sd ld sd ld sd (3 * (1 * 2 + 1) = 3 * 3 + 1 = 9 cycles)
ld ld ld sd sd sd (3 * 2 + 1 = 7 cycles)
```

If the floating point registers must be used, the same thing can be done and will take n * 3 + 1 cycles, but only for n >= 3 (due to latency inside the FPU):

```
fld<times n> fsd<times n>
```

That is, copying three elements:

```
fld fsd fld fsd fld fsd (3 * 7 = 21 cycles, see table above)
fld fld fld fsd fsd fsd (3 * 3 + 1 = 10 cycles)
```

Note, that this assumes that everything is in cache. Also, the write-buffer may limit achievable performance.

115.6.2 NanoFPUnv

The iterative FPU will take short-cuts where possible, but even the simplest float->float operation will take over half a dozen cycles.

115.7 Fault tolerance features

115.7.1 Overview

Note that the features described in this section are only available in FT and FT-FPGA releases of GRLIB.

The NOEL-V processor can be built with functionality to handle bit errors occurring in the SRAM memories included in the processor. The intent with this functionality is to handle random single-event upsets in the memories. Errors encountered are corrected automatically without any visible side effects to software except for a small delay. A scrubber function is included to prevent error build-up.

Two different modes of memory protection are supported, one intended for FPGA usage and one for ASIC and generic usage. The FPGA native ECC uses error detection and correction functionality already built into the FPGA memory blocks, while the generic version intended for ASIC implements error detection and correction codes entirely in RTL. The generic version is also usable for FPGAs lacking built-in ECC and for FPGA prototyping of ASIC designs.

115.7.2 Cache memory protection

The instruction cache tags, instruction cache data, data cache tags, data cache data, and tightly coupled memories (if enabled) can be protected with error correcting codes using either an RTL EDAC or using FPGA native error correction.

When the RTL/ASIC ECC is enabled, an error correcting code is used to protect the memories that has been designed for low impact on maximum frequency and capability to detect many types of multi-bit upset (the code is described further in section 115.7.7). The error detection part of the ECC decoding is performed in parallel with the cache tag/data lookup in order to allow execution to proceed without any performance overhead in the error-free case.

Detected errors during cache operation (tag/data lookups) are handled in the controller by entering a “correction cycle” where the location with the error is read out, corrected and written back while the processor pipeline is held. After the correction cycle, the requested cache lookup is performed again, and depending on cache hit/miss result, either execution continues or the cache miss is processed as usual. From the point of view of the processor pipeline, the cache controller behavior during the correction cycle is exactly the same as for a normal cache miss, but as the correction cycle is done internally in the tag memories there is no access to the processor bus.

The default behavior on multi-bit errors that are not correctable by the ECC, is to invalidate the cache line with the tag and re-fetch the data from memory. In the event of a multi-bit error in a cache tag, all tags belonging to the same set in the cache are invalidated. Note that because the cache is write-through, the latest contents of any memory location is always available in external memory. With this approach, errors in the cache that are uncorrectable from a local ECC point of view are correctable on system level. This approach is similar to how earlier generation of LEON processors (that only had error detection capability) handle errors in the cache.

The processor can be configured to other behaviors in the event of detecting multi-bit errors. It can be set to flush the entire cache where the error was detected.

For the AHB bus snooping functionality, when ECC is enabled, then the snoop tags are checked for errors and corrected on the fly before they are compared with the address of the AHB store. An extra pipeline stage is added to the snoop pipeline to allow for this error correction without impacting maximum clock frequency. After a snoop tag correction, the corrected snoop tags are stored in a register to be written back into the snoop tag memory at the earliest point where the snoop pipeline is free. Uncorrectable errors in the snoop tag are handled by invalidating all ways of the corresponding cache set. The behavior of the snoop tag error correction is not affected by the configuration registers and errors in the snoop tags do not trigger traps.

The FPGA native ECC mode by default operates the same way as described above, with correction cycles when an error is detected. Because the FPGA built-in ECC produces the corrected data immediately on the same cycle as the memory is read, an additional error handling option is available where correction cycles are not performed but the data is simply taken on the fly and the error is left to be corrected later by the scrubber.

115.7.3 Integer and FPU register file protection

When the integer register file used for the general purpose or FPU registers is implemented with SRAM memories, the processor provides options for ECC protection. As in the cache case, there is an RTL variant and a native FPGA variant.

Both for the RTL/ASIC ECC and FPGA native ECC the corrected values is taken on the fly and the error is left to be corrected later by the scrubber. Uncorrectable errors will trap with exception 31.

When error is detected during the scrubbing operation the error is corrected. Because the register file is made up of four two-port memories holding the same data (in order to provide the four read ports), correction can be performed by getting the register contents from another port that is backed by a different memory. In order to get the correct data, the processor pipeline performs a correction where the register that had the detected error is read from all memories and the first copy found without error is taken as the correct value and written into all memories.

When using the flip flop register file, protection options are not available on the IP core level (the flip flops are assumed to be hardened or protection applied through other means).

115.7.4 Hardware scrubber

When the processor has been configured with some form of error correction (either cache memory, TCM or integer register file protection), a hardware scrubber function is also included. The scrubber can be enabled or disabled and the scrub rate is configurable through a configuration register. The intention of the scrubber is to prevent build-up of errors in parts of the cache or register file that are not accessed or replaced regularly.

Each iteration of the scrubber, one location of every tag and data memory in parallel, as well as one register of the integer register file, are read to check for errors and written back in the event of a detected error. The processor pipeline is stalled during the scrubber iteration.

Under normal conditions, a scrub cycle stalls the processor for 7 cycles. The time required may be extended slightly if there is snoop activity on the AHB bus since the access to the snoop tag RAM has to be arbitrated. This is handled in a way similar to diagnostic accesses to the snoop tags as described in section 98.3.15.

For a cache with 4 ways, 4 KiB/way and 32 bytes cache line, it will take 128 scrubber iterations to loop through the tag memories and 512 iterations to loop through the data memories. The time it takes to loop through the TCM depends on selected size, as 64 bits of data can be scrubbed per iteration. It takes 32 scrubber iterations to loop through all integer/FPU registers.

The interval to set for the scrubber should be determined based on the necessary scrub rate needed to avoid error buildup, which would come from a system level analysis. If, as an example, a 100 Hz scrub rate is required for all the memories inside the processor and the processor is running at 500 MHz, the scrub period would have to be set to (10000 / 512) * 500 = 9765 cycles.

Setting very low scrubber periods (below the order 50 cycles) is not recommended as it will slow down execution significantly, and very low periods can make the execution to almost appear stalled.

Error in cache memories and tightly coupled memories that are not correctable by the ECC are handled in the same way as when they are discovered functionally as described in sections 115.7.2 and . Uncorrectable errors in the register file (parity or uncorrectable error in all copies) are handled by leaving the error in the register file to be discovered later by a functional access.

115.7.5 Error counting and reporting

The processor contains two bit wide error counters for correctable errors and one bit wide counters for uncorrectable errors. The counters are saturating in order to avoid accidentally wrapping around to zero. The counters are not reset in order to allow for post-mortem analysis following a crash and a reset. They will therefore contain random values on power up and need to be read out and cleared manually by software before they are used.

In order to not account twice for correctable errors that are found during execution and corrected with a correction cycle, the correctable error counter is only incremented when the the error is first encountered and the correction cycle is entered and is then not incremented again in the correction cycle when the value is re-read and corrected.

In addition to the error counters, there is also a counter for the number of times a correction cycle has been triggered. This can be checked to determine if a correctable error was found by the scrubber (not triggering a correction cycle) or found functionally (when loading/fetching instructions).

115.7.6 Error injection

The cache controller supports error injection into the cache tag and data memories, and the TCM data memories. The error injection interface has been designed to allow for easily injecting errors from software running on the processor.

The error injection is performed in two steps. First the error injection is “armed” by configuring the error injection register. Secondly, the error injection is performed by performing a diagnostic read from the memory and location where the error should be injected. The cache controller will during the read perform the normal diagnostic read, but in addition perform a write-back of the value with the selected error injected. By performing the injection with one read access, the error injection is “atomic” from point of view of the cache state, and there is no risk of for example a cache line getting replaced between reading and writing back the value.

The diagnostic read used to inject the error will first correct any existing correctable errors. Therefore errors will not accumulate when injecting errors multiple times into the same location.

For the native FPGA ECC implementation, not all FPGA technologies have support for error injection. There is a read-only field in the error injection register indicating whether error injection is supported for the current configuration.

115.7.7 Error correcting code description

The cache controller uses an error correcting code that has been designed for low error detection overhead in the cache lookup path, correction of single bit errors, and detection of all wide (up to 4 adjacent bits) errors caused by multiple bit upsets. The code can be viewed as a concatenation of a parity check and a BCH-like code.

The code adds 8 check bits to a data word that is typically 32 bits but can be up to 44 data bits.

Conceptually the code is built up as follows (this example assuming 32 bit wide data):

Step 1: The data is split into four 8-bit parts (“slices”) with each slice having one bit of each nibble (i.e data bits 31,27,...,7,3 in one slice, data bits 30,26,...,6,2 in one slice, data bits 29,25,...,5,1 in one slice, and data bits 28,24,...,4,0 in one slice).

Step 2: For each slice, four checkbits are formed by taking different XOR combinations of the 8 data bits of the slice, similar to a BCH encoder. Each data bit gets included in a unique combination of two or more checkbits.

Step 3: The 4x4 checkbits for the separate parts are combined by bitwise XOR into one set of 4 checkbits. This forms the lower four output checkbits of the encoding process.

Step 4: For the remaining four checkbits, the data bits and the lower four checkbits are viewed as one data word and again split into four parts with each part having one bit of each nibble. All the 9 bits of each part are now XOR:ed to create a parity bit of each part. The four generated parity bits are taken as the top four output checkbits of the encoding process

Since all computations made are based on exclusive-or gates, the steps above to create the different checkbits can be combined into a single XOR function of the data bits.

Decoding is done in two stages. First a parity check is made comparing the four top check bits read out to the XOR of the data bits together with the four lower check bits computed in the same way as in step 4 of the encoding process described above. The parity check generates an “error detected” signal for the cache controller to stop the current access and enter into a correction cycle.

For the error correction, a syndrome is created by generating the lower four checkbits using identical steps to the encoder steps 1-3 above and bitwise XOR:ing the resulting check bits with the check bits read out from the memory. With the result of the parity check indicating which part of the data word has the error, and the syndrome, a look up table can determine which bit caused the error should be corrected. Note that a syndrome with one bit set would be an indication of a bit error in the check bit itself. A parity check showing errors in more than one slice or a syndrome not matching any single-bit error would be an indication of an uncorrectable error at the decoder level. Note that as explained in section 115.7.2, uncorrectable errors in the cache memories can be recovered at higher level by the cache controller by re-reading from memory.

As is the case with any error correcting code, multi-bit error cases that are beyond what the ECC can handle will defeat the check and may result in incorrect data delivered either without any indication of error at all or incorrect data indicated as a correctable error.

A special case exists with this code, where some multi-bit error cases can defeat the initial parity check but are still detected by the syndrome check. These are called “late detected errors”. As the data may have already have been delivered by the cache controller by the time the error is detected, it may be too late to recover from the error. In this case the cache controller can be configured generate an internal error trap to break execution. If the pipeline was held at the time this happens for any reason (for example due to an unrelated cache miss), this gives extra time to act on the error and the pipeline will handle the error in the same way as it would for a regular uncorrectable error. The scrubber will always have the extra time to catch a late detected error so such errors found by the scrubber will always be acted on as uncorrectable errors. Two separate one-bit counters exist for non-recovered and recovered late errors.

115.8 Debug system

115.8.1 Overview

The subsystem includes a debug system (including a debug module) that connects to the debug ports of all CPUs. The debug module handles reset and debug control of the CPU. An AHB access trace is also included in the debug system.

115.8.2 Start-up handling

After reset, the debug module will start up all CPUs to boot from default boot address (see Table 2236). This can be suppressed by asserting the dsubreak signal during reset. If this is set, the processor will be left in stopped state and no instructions will be executed until the user connects with a debugger and starts the processor.

115.8.3 Debug master access

The debug module provides a set of ports for AHB masters to connect to for debug access. Standard debug masters from the GRLIB library can be connected to these debug ports.

The debug module implements a transparent bridge between each debug master and the processor AHB bus, where the debug master can make accesses as if it was on the processor bus directly. However the bridge provides pipeline registers so that there are no direct combinatorial paths between the debug masters and the processor bus, or between the debug masters.

In addition to having access to the processor bus, the debug masters also have debug access to the processors themselves via a register interface that can only be accessed through the debug ports.

The debug masters can be blocked from accessing the system using the dsu_en signal.

115.8.4 Debug module

The debug module is designed according to the RISC-V External Debug Support standard (see section 115.1.2).

[Note: The debug module interface (DMI) is adapted to be used with any debug link available in GRLIB together with the GRMON debug software]

The debug module supports the following features:

- All mandatory control and status registers. This enables the run control functionality.
- Register access via Abstract command. Access support for both GPR and CSR registers.
- Program buffer (implied ebreak)
- Single step (supported via CSR access and run control)

There is no direct access to the FPU registers. The FPU register data needs to be copied to a GPR or CSR register before it can be read out. This can be done by using the program buffer.

The debug link has direct access to the AMBA bus without using the System Bus Access feature described in the standard.

115.9 AMBA bus fabric

The subsystem includes an AMBA 2.0 AHB and APB bus fabric, using the GRLIB AHBCTRL and APBCTRL IP cores. Ports are included to connect external AHB masters and AHB slaves to the processor bus, and to connect additional APB slaves to the APB peripheral bus. The system uses the AMBA Plug and play support and interrupt steering capabilities provided by the GRLIB infrastructure (see GRLIB IP Library User's Manual).

*Table 2247.*Processor AHB bus masters

Master index	Connection
0	Processor 0
1	Processor 1
...	...
(ncpu-1)	Processor (ncpu-1)
ncpu	External master 0
ncpu+1	External master 1
...	...
(ncpu+nextmst-1)	External master (nextmst-1)
ncpu+nextmst	Debug bus bridge

*Table 2248.*Processor AHB bus slaves

Slave index	
0	External slave 0
...	...
nextslv-1	External slave (nextslv-1)
nextslv	APB bridge controller (APBCTRL)
nextslv+1	Core Local interrupt controller
nextslv+2	Platform interupt controller

*Table 2249.*APB bus slaves

Slave index	
0	External APB slave 0
...	...
(nextapb-1)	External APB slave (nextapb-1)
nextapb	Console UART (APBUART)
nextapb+1	Version (GRVERSION)
nextapb+2	Timer (GPTIMER)

115.10 Standard peripherals

The following standard peripherals are included in the subsystem block:

- Core Local / Platform interrupt controller, used to deliver interrupts to the processors
- General purpose timer (GPTIMER), used for time keeping
- Console UART (APBUART), used for standard console output

The peripherals are connected to the subsystem's APB bus

115.11 Memory map

115.11.1Physical memory map

The subsystem implements the following memory map as seen from the processor (using physical addresses) or from other masters on the processor bus, or from masters on the debug ports. NOTE: different memory map are used depending on GRLIB release.

*Table 2250.*Physical bus memory map

Range	Function	Comment
0x00000000 - 0xDFFFFFFF	For use by external AHB slaves (main memory)	Typically, RAM at 0x00000000, ROM at 0xC0000000
0xE0000000 - 0xE000FFFF	Core Local interrupt controller	
0xE0010000 - 0xF7FFFFFF	For use by external AHB slaves	
0xF8000000 - 0xFBFFFFFF	Platform Interrupt controller (PLIC)	
0xFC000000 - 0xFF8FFFFF	For use by external AHB slaves	
0xFF900000 - 0xFF9000FF	Console UART registers	See APBUART documentation
0xFF900100 - 0xFF907FFF	For use by external APB slaves	
0xFF908000 - 0xFF9080FF	Timer registers	See GPTIMER documentation
0xFF908100 - 0xFF9FFFFF	For use by external APB slaves	
0xFFA00000 - 0xFDFFFFFF	For use by external AHB slaves	
0xFE000000 - 0xFEFFFFFF	Debug module	
0xFF000000 - 0xFFEFEFFF	For use by external AHB slaves	
0xFFF00000 - 0xFFF1FFFF	AHB trace buffer	See AHBTRACE documentation
0xFFF20000 - 0xFFFFEFFF	For use by external AHB slaves	
0xFFFFF000 - 0xFFFFFFFF	AMBA Plug'n'play information on the debug bus	Read only.

*Table 2251.*Physical bus memory map (for GRLIB older then 2024.4-b4295)

Range	Function	Comment
0x00000000 - 0xDFFFFFFF	For use by external AHB slaves (main memory)	Typically, RAM at 0x00000000, ROM at 0xC0000000
0xE0000000 - 0xE000FFFF	Core Local interrupt controller	
0xE0010000 - 0xF7FFFFFF	For use by external AHB slaves	
0xF8000000 - 0xFBFFFFFF	Platform Interrupt controller (PLIC)	
0xFC000000 - 0xFC0000FF	Timer registers	See GPTIMER documentation
0xFC001000 - 0xFC0010FF	Console UART registers	See APBUART documentation
0xFC002000 - 0xFC0FFFFF	For use by external APB slaves	
0xFC100000 - 0xFDFFFFFF	For use by external AHB slaves	
0xFE000000 - 0xFEFFFFFF	Debug module	
0xFF000000 - 0xFFEFEFFF	For use by external AHB slaves	
0xFFF00000 - 0xFFF1FFFF	AHB trace buffer	See AHBTRACE documentation
0xFFF20000 - 0xFFFFEFFF	For use by external AHB slaves	
0xFFFFF000 - 0xFFFFFFFF	AMBA Plug'n'play information on the debug bus	Read only.

115.12 Processor configuration and status registers (CSR)

This section defines implementation dependent fields in standard CSR and describes implemented custom CSRs.

*Table 2252.*Standard CSR registers (with implementation dependent fields)

CSR address	Register
0x301	MISA (ISA and extensions)
0x305/0x105/0x205	MTVEC/STVEC/VSTVEC (Trap Vector Configuration)
0x343/0x143/0x243	MTVAL/STVAL/VSTVAL (Trap Value Register)
0x34a/0x64a	MINST/HINST (Trap Instruction Register)
0x34b	MTVAL2 (Second Trap Value Register)
0x30a/0x60a	MENVCFG/HENVCFG (Environment Configuration Register)
0x10a	SENVCFG (Supervisor Environment Configuration Register)
0xF11	MVENDORID (Vendor ID)

*Table 2253.*Custom CSR registers

CSR address	Register
0x7C0	Custom FEATURES
0x7C1	Custom CCTRL
0x7C2	Custom TCMICTRL
0x7C3	Custom TCMDCTRL
0x7C4	Custom FT
0x7C5	Custom EINJECT
0x7C6	RESERVED
0x7D0	Custom FEATURESH
0x7D1	Custom CCTRLH
0x7D2	Custom TCMICTRLH
0x7D3	Custom TCMDCTRLH
0x7D4	Custom FTH
0x7D5	Custom EINJECTH
0x7D6	RESERVED
0xFC0	CAPABILITY(Read-only)
0xFD0	CAPABILITYH (Read-only)

The custom CSR register which name ends with the letter H is only available in a 32-bit NOEL-V implementation and holds register bits 63-32.

115.12.1MISA

*Table 2254.*MISA

XLEN-1	XLEN-2	XLEN-3	26	25	24	23	22	21	20	19	18	17	16
xXL									U		S		
*	0			0					*	0	*	0	
r	r			r					r	r	r	r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			M				I	H		F		D	C		A
0			*	0			1	*	0	*	0	*	*	0	*
r			r	r			r	rw	r	r	r	r	r	r	r

XLEN -1: XLEN -2	xXL - XLEN 1: for RV32 2: for RV64
20	U - User mode implemented
18	S - Supervisor mode implemented
12	M - Integer Multiply/Divide extension
8	I - RV32I/64I base ISA
7	H- Hypervisor extension
5	F - Single-precision floating-point extension (when enabled D is also enabled)
3	D - Double-precision floating-point extension (when enabled F is also enabled)
2	C - Compressed extension
0	A - Atomic extension

115.12.2MVENDORID

*Table 2255.*MVENDORID

31	11	0
-	ID	
0	0x324	
	r	

31:12	Zero
11:0	Gaisler JEDEC ID (0xA4 bank 7)

115.12.3MTVEC/STVEC/VSTVEC

*Table 2256.*xTVEC

XLEN-1	2	1	0
BASE		MODE	
0		0	
rw		rw	

XLEN -1:2	BASE - trap vector base address When MODE = 0: base address aligned to 4 byte When MODE = 1: base address aligned to 256 byte
1:0	MODE - trap vector mode: 0 : Direct 1 : Vectored

115.12.4MTVAL/STVAL/VSTVAL

The xTVAL register is written with a nonzero value (virtual address) when a breakpoint, address-misaligned, access-fault, or page-fault exception occurs on an instruction fetch, load, or store. On a illegal

instruction exception the xTVAL is written with the faulting instruction bits.

115.12.5MTINST/HINST

When a trap is taken into M-/HS-mode, MINST/HINST is written with a value that provides information about the instruction that trapped, to assist software in handling the trap.

115.12.6MTVAL2

When a trap is taken into M-mode, MTVAL2 is written with additional exception-specific information, alongside mtval, to assist software in handling the trap.

115.12.7MENVCFG/HENVCFG

*Table 2257.*MENVCFG/HENVCFG

63	62	61 … 8	7	6	5 … 4	3 … 1	0
STCE	RES	RESERVED	RES	CBCFE	CBIE	RESERVED	RES
0	0	0	0	0	0	0	0
rw	r	r	r	rw	rw	r	r

63 STCE - (STimecmp Enable) enables stimecmp for S-mode
62 RESERVED
61:8 RESERVED
7 RESERVED
6 CBCFE - Cache Block Clean and Flush instruction Enable
5:4 CBIE - Cache Block Invalidate instruction Enable
3:1 RESERVED
0 RESERVED

115.12.8SENVCFG

*Table 2258.*SENVCFG

63 … 8	7	6	5 … 4	3 … 1	0
RESERVED	RES	CBCFE	CBIE	RESERVED	RES
0	0	0	0	0	0
r	r	rw	rw	r	r

63:8 RESERVED
7 RESERVED
6 CBCFE - Cache Block Clean and Flush instruction Enable
5:4 CBIE - Cache Block Invalidate instruction Enable
3:1 RESERVED
0 RESERVED

115.12.9Custom FEATURES (Features enable/disable)

*Table 2259.*FEATURES

63																																32
RESERVED																																
o																																
r																																

31	30																															0
RESERVED																																
0																																
r																																

31	30				26	25																										0
EPBT	DIST					RESERVED																										
						0																										
						r																										

11	10	9	8	7	6	5	4	3	2	1	0
	DIS	SBP	SBD	MADF	DB2BS	DLA	DLB	DRAS	RES	DBP	DDI
	0	0	0	0	0	0	0	0	0	0	0
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

63:32	RESERVED
31	Enable logging of program buffer execution in trace buffer (EPBT)
30:26	Disable trace buffer / privilege mode (DIST) 30: When ‘1’, trace is disabled for VU-mode 29: When ‘1’, trace is disabled for VS-mode 28: When ‘1’, trace is disabled for U-mode 27: When ‘1’, trace is disabled for S-mode 26: When ‘1’, trace is disabled for M-mode
25:11	RESERVED
10	Disable instruction streaming (DIS)
9	Static branch prediction config (SBP)
8	Enable static branch prrediction (SBD)
7	(MADF)
6	Disable back-to-back stores (any size) (DB2BS)
5	Disable late ALU (DLA)
4	Disable late branch prediction (DLB)
3	Disable RAS (DRAS)
2	Reserved
1	Disable branch prediction (DBP)
0	Disable dual-issue (DDI)

115.12.10Custom CCTRL (Cache control)

*Table 2260.*CCTRL

63																																32
RESERVED																																
0																																
r																																

31	30																															0
RESERVED																																
0																																
r																																

31	30																															0
RESERVED																																
0																																
r																																

Table 2260. CCTRL

14	13	12	11	10	9	8	7	6	5 4	3 2	1 0
	IP	DP	RES	ITWP	DTWP	DS	FD	FI	RES	DCS	ICS
	0	0	0	0	0	0	0	0	0	0	0
	r	r	r	rw	rw	rw	rw	rw	r	rw	rw

63:14 RESERVED

13 Instruction cache flush pending (IP). This bit is set when an instruction cache flush operation is in progress

12 Data cache flush pending (DP). This bit is set when an data cache flush operation is in progress.

11 RESERVED

10 ITCM wipe (ITWP). Set to 1 to clear the contents of the instruction TCM. Reads 1 if wipe is in progress, otherwise 0.

9 DTCM wipe (DTWP). Set to 1 to clear the contents of the data TCM. Reads 1 if wipe is in progress, otherwise 0.

8 Data cache snoop enable (DS) - if set, will enable data cache snooping.

7 Flush data cache (FD). If set, will flush the data cache. Always reads as zero.

6 Flush Instruction cache (FI). If set, will flush the instruction cache. Always reads as zero.

5:4 RESERVED

3:2 Data Cache state (DCS) - Indicates the current data cache state according to the following: X0= disabled, 01 = frozen, 11 = enabled.

1:0 Instruction Cache state (ICS) - Indicates the current data cache state according to the following: X0= disabled, 01 = frozen, 11 = enabled.

115.12.11 Custom TCMICTRL (Control register for instruction tightly coupled memory)

To be added.

115.12.12 Custom TCMDCTRL (Control register for data tightly coupled memory)

To be added.

115.13 Fault tolerance registers (CSR)

115.13.1Custom FT

*Table 2261.*FT

63 62	61 60	59 58	57 56	55 54	53 52	51 50	49 48
RFSEC	FRFSEC	ITCEC	IDCEC	DTCEC	STCEC	DDCEC	ICCEC
(nr)	(nr)	(nr)	(nr)	(nr)	(nr)	(nr)	(nr)
rw	rw	rw	rw	rw	rw	rw	rw

47 46	45 44	43	42	41	40	39	38	37	36	35	34	33
DCCEC	CC	ITUEC	IDUEC	DLUEC	DSUEC	DDUEC	ICUE	DCUE	RES	RFDEC	FRF-DEC	
(nr)	(nr)	(nr)	(nr)	(nr)	(nr)	(nr)	(nr)	(nr)	0	(nr)	(nr)	
rw	rw	rw	rw	rw	rw	rw	rw	rw	r	rw	rw	

21	20 19	18 17	16
RESERVED	CUEM	CCEM	SCREN
0	0	0	0
r	rw	rw	rw

15 0
SCRUBPER
0
rw

63:62	Integer Register file single error counter (RFSEC)
61:60	FPU Register file single error counter (FRFSEC)
59:58	Instruction cache tag correctable error counter (ITCEC)
57:56	Instruction cache data correctable error counter (IDCEC)
55:54	Data cache lookup tag correctable error counter (DTCEC)
53:52	Data cache snoop tag correctable error counter (STCEC)
51:50	Data cache data correctable error counter (DDCEC)
49:48	Instruction TCM correctable error counter (ICCEC)
47:46	Data TCM correctable error counter (DCCEC)
45:44	Cache correction cycle counter (CC)
43	Instruction cache tag uncorrectable error counter (ITUEC)
42	Instruction cache data uncorrectable error counter (IDUEC)
41	Data cache lookup tag uncorrectable error counter (DLUEC)
40	Data cache snoop tag uncorrectable error counter (DSUEC)
39	Data cache data uncorrectable error counter (DDUEC)
38	Instruction TCM uncorrectable error counter (ICUE)
37	Data TCM uncorrectable error counter (DCUE)
36	RESERVED
35	Integer Register file double error counter (RFDEC)
34	FPU Register file double error counter (FRFDEC)
33:21	RESERVED
20:19	Cache memory uncorrectable error handling mode (CUEM). 0: flush only the affected cache line (for data error) or set (for tag error) and resume 1: flush entire affected cache 2,3: reserved for future use
18:17	Cache memory correctable error handling mode (CCEM). 0: correct error, writeback and resume 1: correct data on the fly, no writeback (valid in CACHEFT=2 configuration only) 2: reserved for future use 3: flush entire cache and resume

*Table 2261.*FT

16	Enable hardware scrubber for cache memory and register file (SCREN)
15:0	Scrub interval, in cycles, minus one (SCRUBPER).

115.13.2Custom EINJECT (Error injection register)

*Table 2262.*EINJECT

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
RESERVED															
0															
r															

47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RESERVED															
0															
r															

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EISUPP		RESERVED													
*		0													
r		r													

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		EIBIT2						EIBIT1						EIUE	EIEN
		0						0						0	0
		rw						rw						rw	rw

63:32	RESERVED
31:30	Error injection support: 0:Error injection not supported 1:Can inject correctable error only, in unspecified bit 2:Can inject uncorrectable error or correctable error, in unspecified bits 3:Can inject in arbitrary bits selected via EIBIT1,EIBIT2 fields
29:14	RESERVED
13:8	Error injection bit number 2 (EIBIT2). The bit selected by this index will be inverted in error injection in addition to bit selected with EIBIT1 field. Should be set to the same value as EIBIT1 when injecting single bit error. Field is only implemented if EISUPP=3
7:2	Error injection bit number (EIBIT1). The bit selected by this index will be inverted in error injection. Field is only implemented if EISUPP=3
1	Inject uncorrectable error (EIUE). Only implemented if EISUPP=2
0	Enable error injection (EIEN). The next read access to diagnostic cache interface (using custom instructions) will cause an error injection to the read address.

115.14 Custom instructions (diagnostics)

115.14.1Overview

115.14.2Custom diagnostic load instructions

*Table 2263.*Custom Load instructions

31	25	24	23	22	21	20	19	
0000000		0	INST				RS1	
0000000		0	*				*	

15	14	13	12	11	7	6	0
	LD	Size		RD		OP	
	0	10/11		*		0001011	

23:20 INST - diagnostic instructions
0000: I-cache tags load (diag.lw.ict or diag.ld.ict)
0001: I-cache data load (diag.lw.icd or diag.ld.icd)
0010: D-cache tags load (diag.lw.dct or diag.ld.dct)
0011: D-cache data load (diag.lw.dcd or diag.ld.dcd)

19:15 RS1 - source register

14 LD - 0: load instruction

13:12 Size - Size of the access
10: 32-bit word access
11: 64-bit double word access

11:7 RD - destination register

6:0 Custom0 op-code

115.14.3Custom diagnostic store instructions

*Table 2264.*Custom Store instructions

31	25	24	23	22	21	20	19	
0000000		RS2					RS1	
0000000		*					*	

15	14	13	12	11	10	7	6	0
	LD	Size		0	INST		OP	
	0	10/11		0	*		0001011	

24:20 RS2 - source register2

19:15 RS1 - source register1

14 LD - 1: store instruction

13:12 Size - Size of the access
10: 32-bit word access
11: 64-bit double word access

10:7 INST - diagnostic instructions
0000: I-cache tags store(diag.sw.ict or diag.sd.ict)
0001: I-cache data store (diag.sw.icd or diag.sd.icd)
0010: D-cache tags store (diag.sw.dct or diag.sd.dct)
0011: D-cache data store (diag.sw.dcd or diag.sd.dcd)

6:0 Custom0 op-code

115.14.4Assembly macros for custom diagnostic instructions

```
.macro diag.ld.ict rd, rs
  .insn i CUSTOM_0, 3, \rd, \rs, 0
.endm
.macro diag.ld.icd rd, rs
  .insn i CUSTOM_0, 3, \rd, \rs, 1
.endm
.macro diag.ld.dct rd, rs
  .insn i CUSTOM_0, 3, \rd, \rs, 2
.endm
.macro diag.ld.dcd rd, rs
  .insn i CUSTOM_0, 3, \rd, \rs, 3
```

```
.endm
.macro diag.lw.ict rd, rs
  .insn i CUSTOM_0, 2, \rd, \rs, 0
.endm
.macro diag.lw.icd rd, rs
  .insn i CUSTOM_0, 2, \rd, \rs, 1
.endm
.macro diag.lw.dct rd, rs
  .insn i CUSTOM_0, 2, \rd, \rs, 2
.endm
.macro diag.lw.dcd rd, rs
  .insn i CUSTOM_0, 2, \rd, \rs, 3
.endm
.macro diag.sd.ict rs2, rs1
  .insn s CUSTOM_0, 7, \rs2, 0(\rs1)
.endm
.macro diag.sd.icd rs2, rs1
  .insn s CUSTOM_0, 7, \rs2, 1(\rs1)
.endm
.macro diag.sd.dct rs2, rs1
  .insn s CUSTOM_0, 7, \rs2, 2(\rs1)
.endm
.macro diag.sd.dcd rs2, rs1
  .insn s CUSTOM_0, 7, \rs2, 3(\rs1)
.endm
.macro diag.sw.ict rs2, rs1
  .insn s CUSTOM_0, 6, \rs2, 0(\rs1)
.endm
.macro diag.sw.icd rs2, rs1
  .insn s CUSTOM_0, 6, \rs2, 1(\rs1)
.endm
.macro diag.sw.dct rs2, rs1
  .insn s CUSTOM_0, 6, \rs2, 2(\rs1)
.endm
.macro diag.sw.dcd rs2, rs1
  .insn s CUSTOM_0, 6, \rs2, 3(\rs1)
.endm
```

115.15 Software considerations

115.15.1Register file initialization on power up

It is recommended that the boot code for the processor writes all registers in the IU and FPU register files before launching the main application. This allows software to be portable to both FT and non-FT versions of the NOEL-V processor.

115.15.2Start-up

To be documented.

115.16 Vendor and device identifiers

The core has vendor identifiers 0x01 (Frontgrade Gaisler) and device identifiers 0x0BD for the NOEL-V AHB master interface on the processor bus, and 0xBE for the NOEL-V RISC-V debug module (register area) on the debug ports. For description of vendor and device identifiers, see GRLIB IP Library User's Manual.

115.17 Implementation

115.17.1Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_async_reset_enable* is set.

[Note: currently only synchronous reset of all registers are supported.]

115.17.2Technology mapping

NOEL-V has two technology mapping VHDL generics, *fabtech* and *memtech*. The *fabtech* generic controls the implementation of some pipeline features, while *memtech* selects which memory blocks will be used to implement cache memories and the IU/FPU register file. *Fabtech* can be set to any of the provided technologies (0 - NTECH) as defined in the TECHMAP.GENCOMP package. See the GRLIB Users's Manual for available settings for *memtech*.

115.17.3RAM usage

The instruction cache will use only single-port syncram instances for both tags and data.

The data cache tags can be configured into three different modes depending on the cmemconf generic. Depending on configuration it will use either syncram, syncram_2p or syncram_dp instantiations and valid bits will be either included in the tag or held separately in flip flops. The different modes are functionally equivalent.

The data cache data memory is always implemented using single-port memories. It can be configured to use memories with individual byte-write stobes if such are available in the technology. When byte strobes are not available, byte/half-word updates are handled by forwarding back read data from the same memory which may impact timing on some technologies.

115.18 Configuration options

Table 2265 shows the configuration options of the core (VHDL generics).

*Table 2265.*Configuration options

Generic	Function	Allowed range	Default
fabtech	Target technology	0 - NTECH	0 (inferred)
memtech	Vendor library for regfile and cache RAMs.	0 - 16#FFFFFFFF#	0 (inferred)
ncpu	Number of processors instantiated in subsystem	1 - 16	1
nextmst	Number of external masters connected to processor AHB bus Note due to limitations of VHDL, this generic must be at least 1. If no external AHB masters are implemented, select one master and use the ahbm_none constant.	1 - NAHBMST	1
nextslv	Number of external slaves connected to processor AHB bus	1 - NAHBSLV	1
nextapb	Number of external slaves connected to APB bus	0 - NAPBSLV	0
ndbgmst	Number of debug masters connected to debug ports Note due to limitations of VHDL, this generic must be at least 1. If no external debug masters are implemented, select one master and use the ahbm_none constant.	1 - NAHBMST	
cached	Fixed cachability mask. See sections 115.4 and 115.4.13 for more information.	0 - 16#FFFF#	0
wbmask	Wide-bus mask. Indicates which address ranges are 64/128 bit capable. Treated as a 16-bit vector with LSB bit (right-most) indicating address 0 - 0x10000000. See section 115.4.13 for more information.	0 - 16#FFFF#	0

Table 2265. Configuration options

Generic	Function	Allowed range	Default
busw	Bus width of the wide bus area (32, 64 or 128). See section 115.4.13 for more information.	64, 128	64
cmemconf	Cache memory configuration. Sum of data cache tag configuration and data cache data configuration. Data cache tag configuration: 0 - two memories (one two-port, one one-port), valid bits in two-port memory 1 - one dual port memory, valid bits in flip flops 2 - two single port memories, valid bits in flip flops 3 - reserved Data cache data configuration: 0 - Use standard 32-bit one-port memories for data 4 - Use one-port memories with byte writes for data Fault tolerance configuration (FT releases only): 0 - No error correction 16 - FPGA builtin error correction on memories 32 - RTL-based error correction on memories Note: Support for RTL-based error correction will be added in future releases	0 - 7, 16, 18, 32, 34	0
fpuconf	FPU configuration: 0: NanoFPU, one per processor 1: GRFPUnv, one per processor	0-1	0
rfconf	Register file implementation 0 - Implemented in technology mapped syncram 1 - Implemented in register In releases with fault tolerance available, one of the following-constants may be added to the rfconf value to configure fault tolerance when using memories: 16: FPGA builtin error correction on memories 32: Reserved 48: RTL-based error correction on memories	0-1 + 16, 32, 48	0
mulconf	Implementation option of Integer multiplier (dual-cycle) 0 - Multiply full-width 1 - Explicitly multiply half-width and then add The same implementation option is available for the floating-point multiplier, by adding this values to the mulconf generic. 16 - Dual-cycle multiplier	0-1 + 16	0
disas	Print instruction disassembly in VHDL simulator console. Does not affect synthesis output	0 - 3	0
ahbtrace	Enable AHB trace output in VHDL simulator console. Does not affect synthesis output	0 - 1	0

*Table 2265.*Configuration options

Generic	Function	Allowed range	Default
cfg	Select standard NOEL-V configuration, see table 2225. The following values is used to select one of the standard configurations: 4*256: **HP** 3*256: **GP** 2*256: **MC** To select sub-configuration options the following constants can be added to the cfg generic: 128: Lite version 2: No FPU 1: Single-issue pipeline	0 - 1024	0
devid	Device ID for AMBA plug'n'play information table		0
nodbus	0 - Debug module and debug links are located at a separate internal bus. 1- Debug module and debug links are located at the processor bus. Note: The debug structure will be redesigned in future releases and this configuration option will be removed.	0 - 1	0

115.19 Signal descriptions

Table 2266 shows the interface signals of the core (VHDL ports).

*Table 2266.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RSTN	N/A	Input	Reset	Low
AHBMI	*	Output	AHB external master input signals	-
AHBMO[(NCPU+NEXTMST-1) : NCPU]	*	Input	AHB external master output signals	-
AHBSI	*	Output	AHB external slave input signals	-
AHBSO [(NEXTSLV-1):0]	*	Input	AHB external slave output signals	-
DBGMI[(NDBGMST-1):0]	*	Output	AHB debug master input signals Note: Unlike typical AHB signals, this is a separate input record per master. Each element must be connected to the corresponding master	-
DBGMO[(NDBGMST-1):0]	*	Input	AHB debug master output signals	-
APBI	*	Output	APB external slave input signals	-
APBO	*	Input	APB external slave output signals Note: Only a subset of the vector is used corresponding to elements 0:(NEXTAPB-1)	-
DSUEN	N/A	Input	Debug module enable, enables the debug module and allow debug master access	High
DSUBREAK	N/A	Input	Debug module break. On reset, disables automatic start-up of CPU0. After reset, rising edge will make the debug module break the processors.	High
CPU0ERRN	N/A	Output	CPU0 error mode indication	Low
UARTI	*	Input	Console UART input/receive signals	-
UARTO	*	Output	Console UART output/transmis signals	-

Table 2266. Signal descriptions

Signal name	Field	Type	Function	Active
CNT	**	Output	Event for performance counters	High
TESTEN	N/A	Input	The scan test signals are propagated into the AHB controller to be distributed via the AMBA records. See the section on scan test mode support in GRLIB IP Library User's Manual.	High
TESTRST	N/A	Input		Low
SCANEN	N/A	Input		High
TESTOEN	N/A	Input		-
TESTSIG[(1+GRLIB_CONFIG_ARRAY(grlib_techmap_testin_extra):0]	N/A	Input	These signals have default values in the component and can be left unconnected when the scantest generic is 0.	-

* see GRLIB IP Library User's Manual
** fields to be defined

115.20 Library dependencies

Table 2267 shows the libraries used when instantiating the core (VHDL libraries).

Table 2267. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	NOELV	Component, signals	NOEL-V component declaration and debug signals declaration

116 PHY - Ethernet PHY simulation model

116.1 Overview

The PHY is a simulation model of an IEEE 802.3 compliant Ethernet PHY. It provides a complete MII and GMII interface with the basic, extended status and extended capability registers accessible through the management interface (MDIO). Not all of the functionality is implemented and many of the register bits are therefore only writable and readable but do not have any effect. Currently only the loopback is supported.

116.2 Operation

The PHY simulation model was designed to make it possible to perform simple simulations on the GRETH and GRETH_GBIT cores in GRLIB. It provides the complete set of basic, extended capability and extended status registers through the MII management interface (MDIO) and a loopback mode for data transfers. Figure 1 shows a block diagram of a typical connection.

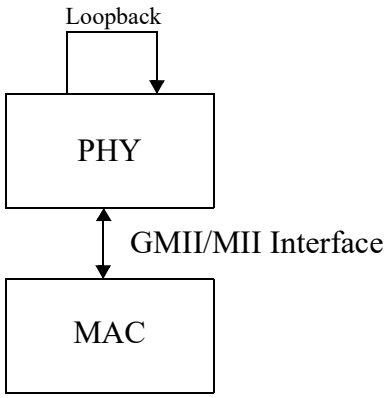


Figure 308. Block diagram of the PHY simulation model connected to a MAC.

The PHY model provides the complete GMII and MII interface as defined by the IEEE 802.3 standard. The model can be used in any of the following modes: 10 Mbit half- or full duplex, 100 Mbit half- or full-duplex and 1000 Mbit half- or full-duplex. This support refers only to the configuration settings available through the MDIO registers. Since the datapath implementation is loopback no collisions will ever be seen on the network and operation will essentially be full-duplex all the time. In loopback mode, rx_clk and tx_clk are identical in both frequency and phase and are driven by the PHY when not in gigabit mode. In gigabit mode the gtx_clk input is used as the transmitter clock and it also drives rx_clk.

When not configured to loopback mode the PHY just sits idle and ignores transmitted packet and does not insert any activity on the receive interface. Clocks are still generated but in this case rx_clk and tx_clk does have the same frequency but not the same phase when not in gigabit mode.

A simple auto-negotiation function is provided and the supported and advertised modes are set through vhdl generics. The generic values will be directly reflected in the reset values and read-only values of all corresponding MII management registers.

116.3 Configuration options

Table 2268 shows the configuration options of the model (VHDL generics).

*Table 2268.*Configuration options

Generic	Function	Allowed range	Default
address	Address of the PHY on the MII management interface	0 - 31	0
extended_regs	Include extended register capability	0 - 1	1
aneg	Enable auto-negotiation functionality	0 - 1	1
base100_t4	Enable support for 100Base-T4	0 - 1	0
base100_x_fd	Enable support for 100Base-X full-duplex	0 - 1	1
base100_x_hd	Enable support for 100Base-X half-duplex	0 - 1	1
fd_10	Enable support for 10Base-T full-duplex	0 - 1	1
hd_10	Enable support for 10Base-T half-duplex	0 - 1	1
base100_t2_fd	Enable support for 100Base-T2 full-duplex	0 - 1	1
base100_t2_hd	Enable support for 100Base-T2 half-duplex	0 - 1	1
base1000_x_fd	Enable support for 1000Base-X full-duplex	0 - 1	0
base1000_x_hd	Enable support for 1000Base-X half-duplex	0 - 1	0
base1000_t_fd	Enable support for 1000Base-T full-duplex	0 - 1	1
base1000_t_hd	Enable support for 1000Base-T half-duplex	0 - 1	1
rmii	Set PHY in RMII mode	0 - 1	0

116.4 Signal descriptions

Table 2269 shows the interface signals of the model (VHDL ports).

*Table 2269.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	-	Input	Reset	Low
MDIO	-	Input/ Output	Data signal for the management interface (Currently not used)	-
TX_CLK	-	Output	Transmitter clock	-
RX_CLK	-	Output	Receiver clock	-
RXD	-	Output	Receiver data	-
RX_DV	-	Output	Receiver data valid	High
RX_ER	-	Output	Receiver error	High
RX_COL	-	Output	Collision	High
RX_CRS	-	Output	Carrier sense	High
TXD	-	Input	Transmitter data	-
TX_EN	-	Input	Transmitter enable	High
TX_ER	-	Input	Transmitter error	High
MDC	-	Input	Management interface clock (Currently not used)	-
GTX_CLK	-	Input	Gigabit transmitter clock	-

see the IEEE 802.3 standard for a description of how the signals are used.

116.5 Library dependencies

Table 2270 shows the libraries used when instantiating the model (VHDL libraries).

*Table 2270.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	SIM	Component	Component declaration

116.6 Instantiation

This example shows how the model can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library gaisler;
use gaisler.sim.all;

entity phy_ex is
  port (
rst : std_ulogic;
clk : std_ulogic;
    );
end;

architecture rtl of phy_ex is

  -- Signals

 signal etx_clk   : std_logic;
 signal gtx_clk   : std_logic;
 signal erx_clk   : std_logic;
 signal erxd      : std_logic_vector(7 downto 0);
 signal erx_dv    : std_logic;
 signal erx_er    : std_logic;
 signal erx_col   : std_logic;
 signal erx_crs   : std_logic;
 signal etxd      : std_logic_vector(7 downto 0);
 signal etx_en    : std_logic;
 signal etx_er    : std_logic;
 signal emdc      : std_logic;

begin

  -- Other components are instantiated here
  ...


  -- PHY model
  phy0 : phy
  generic map (address => 1)
  port map(resetn => rst, mdio => open, tx_clk => etx_clk, rx_clk => erx_clk, rxd => erxd,
  rx_dv => erx_dv, rx_er => erx_er,
  rx_col => erx_col, rx_crs => erx_crs, txd => etxd, tx_en => etx_en,
  tx_er => etx_er, mdc => emdc, gtx_clk => gtx_clk);
end;
```

117 RGMII - Reduced Ethernet Media Access Controller

117.1 Overview

Frontgrade Gaisler's RGMII IP provides the RGMII adaptation layer between the Ethernet physical media device and the GRETH or GRETH_GBIT IP core.

The RGMII adaptation layer supports version 2.0 of the Reduced Gigabit Media Independent interface and can dynamically switch between different speed modes of operation.

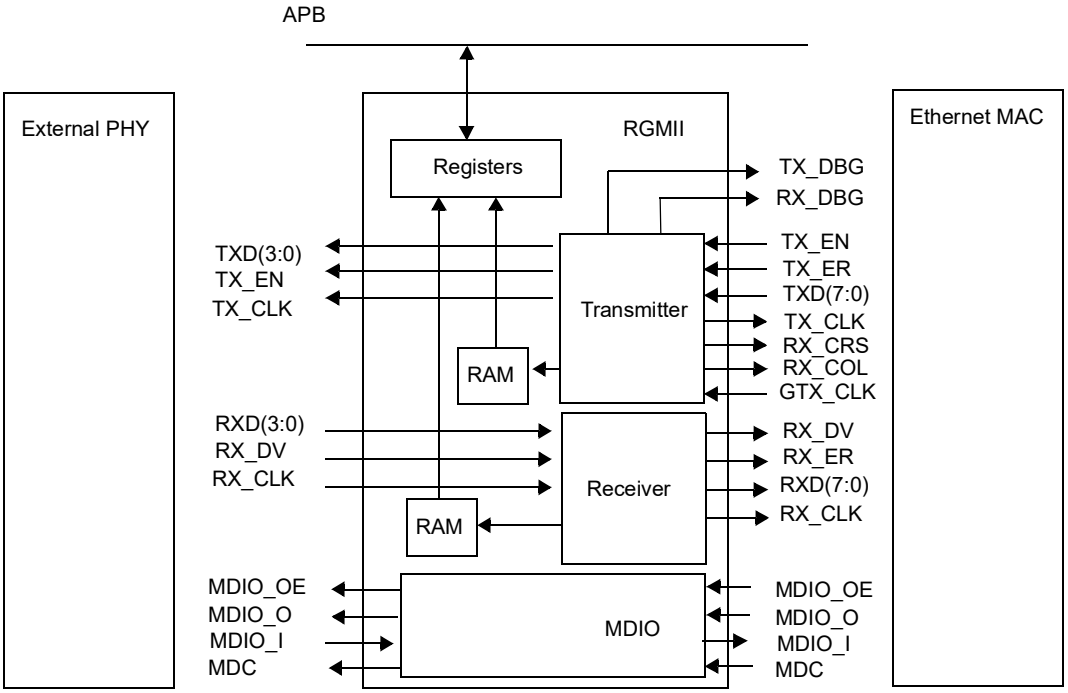


Figure 309. Block diagram of the internal structure of the RGMII.

117.2 Operation

117.2.1 Protocol support

The RGMII adaptation layer supports version 2.0 of the Reduced Gigabit Media Independent interface and IEEE standard 802.3-2002 for GMII.

117.2.2 Transmit clock

The transmitter clock is used by the transmit logic of the core. The transmit clock is also an output from the RGMII IP and will be used as the TX clock in the Ethernet MAC. There are three options of generating and using the transmit clock inside the IP. The VHDL generics ‘no_clock_mux’ and ‘use90degtxclk’ determines the clock mode used by the RGMII IP.

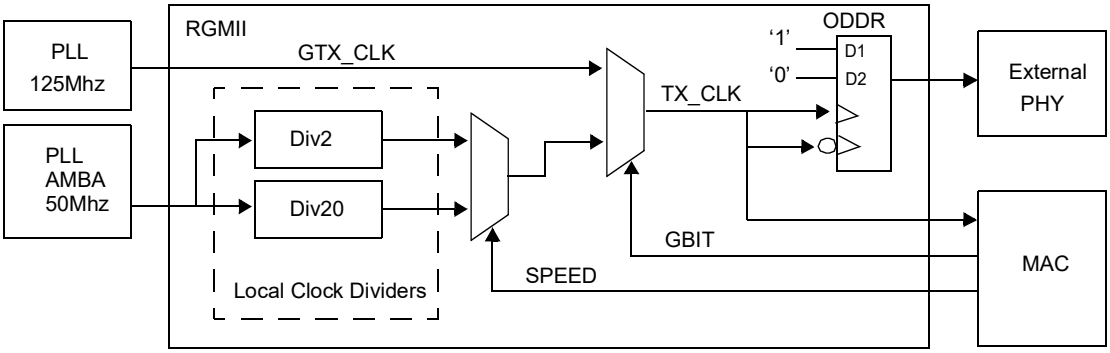


Figure 310. 10/100 Mb/s transmitter clocks generated by internal logic

Figure 310 shows the clocking scheme when the transmit clock for 10Mb/s and 100Mb/s mode is generated internally. This clock mode is used when the VHDL generic 'no_clock_mux' is set to '0'. In this case user must ensure that the GMII clock frequency and AMBA clock frequency is appropriate for the line speed. That is, 50Mhz AMBA clock for 10/100 Mb/s and 125Mhz for 1000Mb/s.

Local clock resources will provide 2.5MHz and 25MHz frequency clocks for 10 Mb/s and 100 Mb/s speeds of operation, respectively. The local clock logic generates the 2.5MHz and 25MHz TX_CLK clock from the 50MHz AHB_CLK.

The SPEED and GBIT signals from the Ethernet MAC are used as selection pins of the local clock muxes.

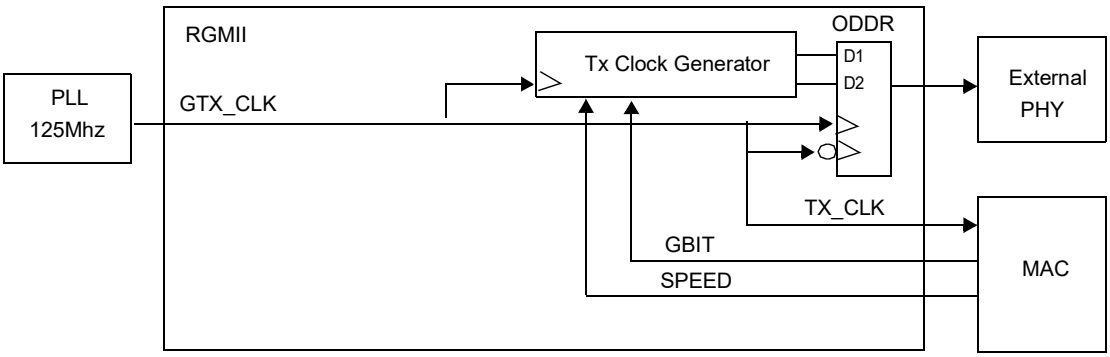


Figure 311. All transmit clocks generated by internal logic

Figure 311 show an alternative clock scheme using only one clock domain for the transmitter. This clock mode is used when setting the VHDL generic 'no_clock_mux' to 1. The local clock divider is replaced by a counter in the GTX_CLK clock domain to generate bit pattern for clock generation in 10 Mb/s, 100 Mb/s, and 1 Gb/s speeds of operation.

The bit pattern is connected to the DDR output buffers data inputs and 125MHz GTX_CLK clock is connected to the DDR output buffer clock in order to generate the transmit clock to the external PHY.

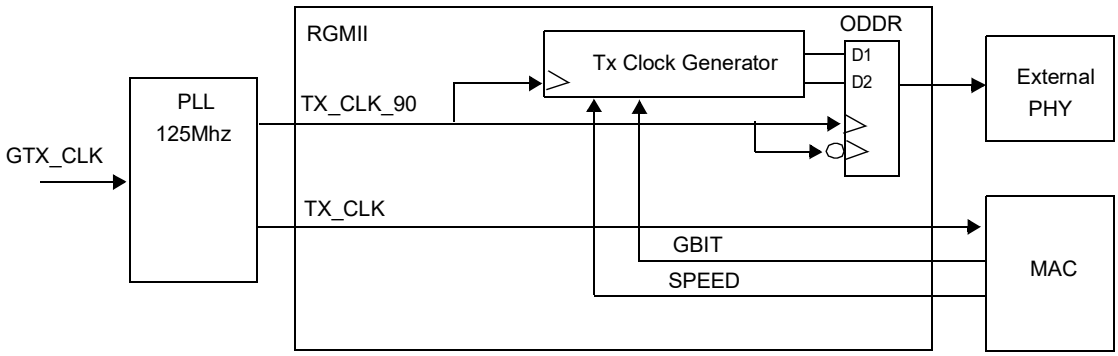


Figure 312. External PHY transmit clock 90deg phase shift

The RGMII v2.0 standard specifies that the external PHY TX clock to have a setup of 2 ns with respect to the TX data. The 2ns setup can be achieved by phase shifting the external transmit clock to the external PHY by 90 degrees relative the internal MAC transmit clock. This mode is shown in figure 312 and is used when setting the VHDL generic 'no_clock_mux' to 1 and 'use90degtxclk' to 1.

117.2.3 Transmitter Operation

The transmitter is enabled when the GMII transmitter enable is high. Data is transferred from the GMII to RGMII as long as the GMII transmitter enable signal is high.

The transmitter enable signal is expected to be high during the complete transmission of the Ethernet MAC frame.

117.2.4 Side-band information

The RGMII receiver samples input data at all time. Side-band information is stored in the RGMII status register accessible via the APB interface. The status register has an address offset of 0x0.

117.2.5 MDIO Management

The RGMII IP only forwards the MDIO signals from the MDIO bus master (GRTEH/GRETH_GBIT) to the external PHY. The RGMII does not affect any MDIO transactions.

117.2.6 RAM debug support

The IP RGMII can optionally be build to include debug memories. The debug memories will record and store the last received and transmitted MAC frame. The RAM debug support is used when the VHDL generic ’debugmem’ to 1.

The transmit frame buffer is accessed starting from the APB address offset 0x400 and the receive frame buffer is located at APB address offset 0x800.

117.2.7 External debug support

Interface for optionality connection to an on-chip logic analyzer for tracing and displaying of on-chip signals e.g. LOGAN IP included in the GRIP.

117.3 Media Independent Interfaces

There are several interfaces defined between the MAC sublayer and the Physical layer. The GRETH_GBIT supports the Media Independent Interface (MII) and the Gigabit Media Independent Interface (GMII).

The GMII is used in 1000 Mbit mode and the MII in 10 and 100 Mbit. These interfaces are defined separately in the 802.3-2002 standard but in practice they share most of the signals. The GMII has 9 additional signals compared to the MII. Four data signals are added to the receiver and transmitter data interfaces respectively and a new transmit clock for the gigabit mode is also introduced.

The RGMII interface uses the same signal with the additional signal tx_clk_90.

*Table 2271.*Signals in RGMII, GMII and MII.

MII and GMII	GMII Only	RGMII Only
txd[3:0]	txd[7:4]	tx_clk_90
tx_en	rxd[7:4]	
tx_er	gtx_clk	
rx_col		
rx_crs		
rxd[3:0]		
rx_clk		
rx_er		
rx_dv		
rx_en		
tx_dv		

117.4 Registers

The core is programmed through registers mapped into APB address space.

*Table 2272.*RGMII registers

APB address offset	Register
0x0	Status register
0x4	Interrupt-source register
0x8	Interrupt mask register
0xC	25Mhz bit pattern wrap register
0x10	25Mhz bit pattern first edge register
0x14	25Mhz bit pattern second edge register
0x18	2.5Mhz bit pattern wrap register
0x1C	2.5Mhz bit pattern first edge register
0x20	2.5Mhz bit pattern second edge register
0x24	Bit pattern register
0x28	Data bit 4 to 7 (negative clocked) delay register
0x2C	Data bit 0 to 3 (positive clocked) delay register
0x30	Data bit swap register
0x400 - 0x7FC	Transmit RAM buffer debug access
0x800 - 0xBFC	Receiver RAM buffer debug access

117.4.1 Status / Interrupt / Mask Register

*Table 2273.*RGMII status/interrupt/mask register

31 16	15	14	13	12 10	9	8	7	6	5	4	3	2 1	0
RESERVED	NM	RD	GA	RESERVED	GB	SP	CS	CE	CX	FC	DS	CS	LS

15 No Clock Mux Mode (NM) - If this bit always reads as a 1 no internal clock logic has been used

14 RAM debug enable (RD) - If this bit always reads as a 1 the debug ram is available.

13 Gigabit MAC available (GA) - If this bit always reads as a 1 the MAC has 1000 Mbit capability.

9 Gigabit (GB) - 1 shows the current speed mode is1000 Mbit in the MAC and when set to 0, the speed mode is selected with bit 7 (SP). Reset value: ‘0’.

8 Speed (SP) - Shows the current speed mode of the MAC. 0 = 10 Mbit, 1 = 100 Mbit. Must not be set to 1 at the same time as bit 8 (GB). Reset valuie: ‘0’.

7 Carrier Sense (CS) - Carrier detected

6 Carrier Extend Error (CE) - Carrier Extend frames Error

5 Carrier Extend (CX) - Carrier Extend frames

4 False Carrier (FC) - False Carrier is detected

3 Duplex Status (DS) - Indicates duplex status 0=half-duplex, 1=full duplex

2: 1 Link Clock Speed (CS) - Indicates RXC clock speed 00=2.5Mhz, 01=25Mhz, and 10=125Mhz, 11=reserved

0 Links status (LS) - Indicates link status 0=down, 1=up

117.4.2 25 MHz Clock Warp Register

*Table 2274.*RGMII 25MHz clock warp register

31 16	15 6	5 0
RESERVED		WRAP

31: 6 RESERVED

5: 0 25Mhz warp register (WARP) - Number of GTX_CLK cycles to generate 25Mhz clock pattern

117.4.3 25 MHz First Edge Register

*Table 2275.*RGMII 25MHz first edge register.

31 16	15 6	5 0
RESERVED		EDGE1

31: 6 RESERVED

5: 0 25Mhz first edge (EDGE1) - Number of GTX_CLK cycles to before clock change state

117.4.4 25 MHz Second Edge Register

Table 2276. RGMII 25MHz Second edge register.

31 16	15 6	5 0
RESERVED		EDGE2

31: 6 RESERVED

5: 0 25Mhz Second edge (EDGE2) - Number of GTX_CLK cycles after the first edge the clock change state

117.4.5 Bit Pattern Register

Table 2277. RGMII bit pattern register.

31 18	17 16	15 14	13 8	7 6	5 0
Not Used	P125M	Not Used	P25M	Not Used	P2M5

17: 16 125Mhz clock pattern (P125M) - Pattern for generating 125Mhz clock

9: 3 25Mhz clock pattern (P25M) - Pattern for generating 25Mhz clock

5: 0 2.5Mhz clock pattern (P2M5) - Pattern for generating 2.5Mhz clock

117.4.6 Positive / Negative Clocked Data Receiver Delay Register

Table 2278. RGMII positive / negative clocked data receiver delay register.

31 2	1 0
Not Used	DEL

2: 0 Delay input (DEL) - Number of RX clock cycles to delay input data

117.4.7 Receiver Data Swap Register

Table 2279. RGMII receiver data swap register

31 1	0
Not Used	SW

0 Swap receiver input clock edge (SW) - Swaps data between negative and positive clocked data input

117.5 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x093. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

117.6 Implementation

117.6.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

117.7 Configuration options

Table 2280 shows the configuration options of the core (VHDL generics).

*Table 2280.*Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the RGMII.	0 - NAHBIRQ-1	0
tech	Technology used for the DDR buffers.	0 - NTECH	inferred
debugmem	Enables debug access to the core's RAM blocks through the APB interface.	0 - 1	0
abits	Selects the number of APB address bits used to decode the register addresses	3 - 8	8
no_clk_mux	Dont't generate 10Mb and 100Mb mode clock in RGMII logic	0 - 1	0
use90degtxclk	Use external generated trasnmit clock to to have a setup of 2 ns with respect to the TX data. VHDL generic 'no_clk_mux' must be set to '1' to use this option.	0 - 1	0

117.8 Signal descriptions

Table 2281 shows the interface signals of the core (VHDL ports).

*Table 2281.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	System reset	Low
APB_RSTN	N/A	Input	AMBA Reset	Low
APB_CLK	N/A	Input	AMBA Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-

*Table 2281.*Signal descriptions

Signal name	Field	Type	Function	Active
GMIII	gtx_clk	Output	Ethernet gigabit transmit clock.	-
	rmii_clk	Output	Ethernet RMII clock.	-
	tx_clk	Output	Ethernet transmit clock.	-
	tx_dv	Output	Ethernet transmitter enable	-
	rx_clk	Output	Ethernet receive clock.	-
	rxd	Output	Ethernet receive data.	-
	rx_dv	Output	Ethernet receive data valid.	High
	rx_er	Output	Ethernet receive error.	High
	rx_col	Output	Ethernet collision detected. (Asynchronous, sampled with tx_clk)	High
	rx_crs	Output	Ethernet carrier sense. (Asynchronous, sampled with tx_clk)	High
	rx_en	Output	Ethernet receiver enable.	-
	mdio_i	Output	Ethernet management data input	-
	mdint	Output	Ethernet management interrupt	-
GMIIO	reset	Input	Ethernet reset (asserted when the MAC is reset).	Low
	txd	Input	Ethernet transmit data.	-
	tx_en	Input	Ethernet transmit enable.	High
	tx_er	Input	Ethernet transmit error.	High
	mdc	Input	Ethernet management data clock.	-
	mdio_o	Input	Ethernet management data output.	-
	mdio_oe	Input	Ethernet management data output enable.	Set by the oepol generic in MAC.
RGMIII	gtx_clk	Input	Ethernet gigabit transmit clock.	-
	rx_clk	Input	Ethernet receive clock.	-
	tx_clk	Input	Ethernet receive clock.	-
	tx_clk_90	Input	Ethernet receive clock phase shifted 90 deg.	-
	rxd	Input	Ethernet receive data.	-
	rx_dv	Input	Ethernet receive data valid.	High
	mdint	Input	Ethernet management data input	-
	mdio_i	Input	Ethernet management interrupt	-
RGMIIO	reset	Output	External RGMII PHY reset	Low
	tx_clk	Output	Ethernet transmit clock output	-
	txd	Output	Ethernet transmit data.	-
	tx_en	Output	Ethernet transmit data valid.	-
	mdio_o	Output	Ethernet management data output.	-
	mdio_oe	Output	Ethernet management data output enable.	Set by the oepol generic.
	mdc	Output	Ethernet management data output enable.	Set by the oepol generic in MAC.

*Table 2281.*Signal descriptions

Signal name	Field	Type	Function	Active
DEBUG **	debug_rgmii_phy_tx	Output	Ethernet transmit debug port.	-
	debug_rgmii_phy_rx	Output	Ethernet receive debug port.	-

* see GRLIB IP Library User's Manual

** Leave port unconnected if not used.

117.9 Library dependencies

Table 2282 shows libraries used when instantiating the core (VHDL libraries).

*Table 2282.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	ETHERNET_MAC	Signals, component	GRETH_GBIT component declarations, GRETH_GBIT signals.
GAISLER	NET	Signals	Ethernet signals

117.10 Instantiation

The first example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.ethernet_mac.all;

entity rgmii_ex is
  port (
    clk    : in std_ulogic;
    rstn   : in std_ulogic;

    -- ethernet signals
    rgmiii :  in  eth_in_type;
    rgmiio :  in  eth_out_type
    );
end;

architecture rtl of rgmii_ex is

  -- AMBA signals
  signal apbi : apb_slv_in_type;
  signal apbo : apb_slv_out_vector := (others => apb_none);
  signal ethi : eth_in_type;
  signal etho : eth_out_type;

begin

  -- AMBA Components are instantiated here
  ...

  -- RGMII
  rgmii0 : rgmii
  generic map (
    pindex        => 11,
    paddr         => 16#010#,
```

```
    pmask         => 16#ff0#,
    tech          => fabtech,
    gmii          => CFG_GRETH1G,
    debugmem      => 1,
    abits         => 8,
    no_clk_mux    => 1,
    pirq          => 11,
    use90degtxclk => 1)
  port map (
    rstn     => rstn,
    gmiii    => ethi,
    gmii0    => etho,
    rgmiii   => rgmiii,
    rgmiio   => rgmii0,
    apb_clk  => apb_clk,
    apb_rstn => apb_rstn,
    apbi     => apbi,
    apbo     => apbo(11));
```

118 REGFILE_3P 3-port RAM generator (2 read, 1 write)

118.1 Overview

The 3-port register file has two read ports and one write port. Each port has a separate address and data bus. All inputs are latched on the rising edge of clk. The read data appears on dataout directly after the clk rising edge. Note: on most technologies, the register file is implemented with two 2-port RAMs with combined write ports. Address width, data width and target technology is parametrizable through generics.

Write-through is supported if the function *syncram_2p_write_through(tech)* returns 1 for the target technology.

118.2 Configuration options

Table 2283 shows the configuration options of the core (VHDL generics).

*Table 2283.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	see table 2284	-
dbits	Data width	see table 2284l	-
wrfst	Write-first (write-through). Only applicable to inferred technology	0 - 1	0
numregs	Not used		

Table 2284 shows the supported technologies for the core.

*Table 2284.*Supported technologies

Tech name	Technology	RAM cell	abit range	dbit range
axcel / axdsp	Actel AX/RTAX & RTAX-DSP	RAM64K36	2 - 12	unlimited
altera	All Altera devices	altsyncram	unlimited	unlimited
ihp25	IHP 0.25	flip-flops	unlimited	unlimited
inferred	Behavioural description	synthesis tool dependent		
rhumc	Rad-hard UMC 0.18	flip-flops	unlimited	unlimited
virtex	Xilinx Virtex, Virtex-E, Spartan-2	RAMB4_Sn	2 - 10	unlimited
virtex2	Xilinx Virtex2, Spartan3, Virtex4	RAMB16_Sn	2 - 14	unlimited
proasic3	Actel Proasic3	ram4k9	2 - 12	unlimited
lattice	Lattice XP/EC/ECP	dp8ka	2 - 13	unlimited
memvirage	Virage ASIC RAM	hdss2_64x32cm4sw0 hdss2_128x32cm4sw0 hdss2_256x32cm4sw0 hdss2_512x32cm4sw0	6 - 9	32
eclipse	Aeroflex/Quicklogic FPGA	RAM128x18_25um RAM256X9_25um RAM512X4_25um RAM1024X2_25um	2 - 10	unlimited
easic90	eASIC 90 nm Nextreme	eram	2 - 12	unlimited

118.3 Signal descriptions

Table 2285 shows the interface signals of the core (VHDL ports).

Table 2285. Signal descriptions

Signal name	Field	Type	Function	Active
WCLK	N/A	Input	Write port clock	
WADDR	N/A	Input	Write address	
WDATA	N/A	Input	Write data	
WE	N/A	Input	Write enable	High
RCLK	N/A	Input	Read ports clock	-
RADDR1	N/A	Input	Read port1 address	-
RE1	N/A	Input	Read port1 enable	High
RDATA1	N/A	Output	Read port1 data	-
RADDR2	N/A	Input	Read port2 address	-
RE2	N/A	Input	Read port2 enable	High
RDATA2	N/A	Output	Read port2 data	-

118.4 Library dependencies

Table 2286 shows libraries used when instantiating the core (VHDL libraries).

Table 2286. Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Constants	Technology contants

118.5 Component declaration

The core has the following component declaration.

```
library techmap;
use techmap.gencomp.all;

component regfile_3p
  generic (tech : integer := 0; abits : integer := 6; dbits : integer := 8;
           wrfst : integer := 0; numregs : integer := 64);
  port (
    wclk   : in  std_ulogic;
    waddr  : in  std_logic_vector((abits -1) downto 0);
    wdata  : in  std_logic_vector((dbits -1) downto 0);
    we     : in  std_ulogic;
    rclk   : in  std_ulogic;
    raddr1 : in  std_logic_vector((abits -1) downto 0);
    re1    : in  std_ulogic;
    rdata1 : out std_logic_vector((dbits -1) downto 0);
    raddr2 : in  std_logic_vector((abits -1) downto 0);
    re2    : in  std_ulogic;
    rdata2 : out std_logic_vector((dbits -1) downto 0)
  );
  end component;
```

119 RSTGEN - Reset generation

119.1 Overview

The RSTGEN reset generator implements input reset signal synchronization with glitch filtering and generates the internal reset signal. The input reset signal can be asynchronous.

119.2 Operation

The reset generator latches the value of the clock lock signal on each rising edge of the clock. The lock signal serves as input to a five-bit shift register. The three most significant bits of this shift register are clocked into the reset output register. The reset signal to the system is high when both the reset output register and the reset input signal are high. Since the output register depends on the system clock the active low reset output from the core will go high synchronously to the system clock. The raw reset output does not depend on the system clock or clock lock signal and is polarity adjusted to be active low.

The VHDL generic *syncrst* determines how the core resets its shift register and the reset output register. When *syncrst* is set to 1 the core's shift register will have an synchronous reset and no reset signal will be connected to the output reset register, see figure 313. Note that the core's reset output signal will always go low when the input reset signal is activated.

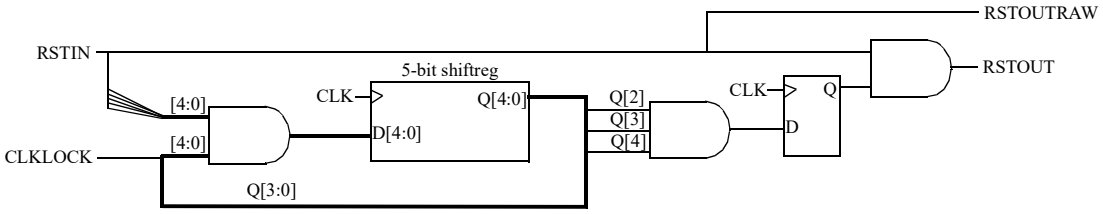


Figure 313. Reset generator with VHDL generic syncrst set to 1

When *syncrst* is 0 the shift register will be reset asynchronously together with the reset output register. Figure 314 shows the reset generator when scan test support is disabled. The shift register reset will be connected to the core's normal reset input and the test reset input will be unused. When scan test support is enabled, the core's test reset input can be connected to the reset input on both registers. The reset signal to use for the registers is selected with the test enable input, see figure 315.

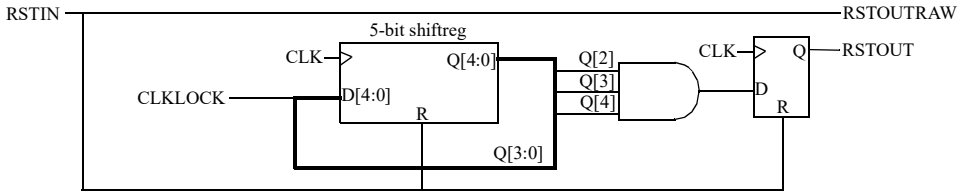


Figure 314. Reset generator with VHDL generic syncrst set to 0 and scan test disabled

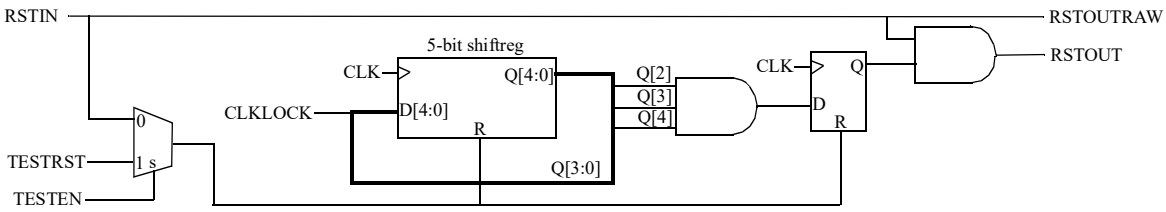


Figure 315. Reset generator with VHDL generic syncrst set to 0 and scan test enabled

When *syncin* is 1 the reset input is synchronized to the same clock domain as the rstgen block.

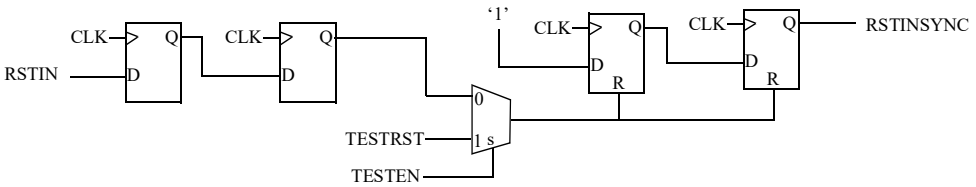


Figure 316. Extra Reset generator logic with VHDL generic syncin set to 1 to synchronize input reset and scan test enabled

119.3 Configuration options

Table 2287 shows the configuration options of the core (VHDL generics).

*Table 2287.*Configuration options

Generic name	Function	Allowed range	Default
acthigh	Set to 1 if reset input is active high. The core outputs an active low reset.	0 - 1	0
syncrst	When this generic is set to 1 the reset signal will use a synchronous reset to reset the filter registers. When this generic is set to 1 the TESTRST and TESTEN inputs will not be used.	0 - 1	0
scanen	Setting this generic to 1 enables scan test support. This connects the TESTRST input via a multiplexer so that the TESTRST and TESTEN signals can be used to asynchronously reset the core's registers. This also requires that the generic syncrst is set to 0.	0 - 1	0
syncin	Setting this generic to 1 will add reset synchronizer to the input reset. This option can be used to break false timing paths in designs when input reset is not generated from same clock domain as the input clock of rstgen	0 - 1	0

119.4 Signal descriptions

Table 2288 shows the interface signals of the core (VHDL ports).

*Table 2288.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTIN	N/A	Input	Reset	-
CLK	N/A	Input	Clock	-
CLKLOCK	N/A	Input	Clock lock	High
RSTOUT	N/A	Output	Filtered reset	Low
RSTOUTRAW	N/A	Output	Raw reset	Low
TESTRST	N/A	Input	Test reset	-
TESTEN	N/A	Input	Test enable	High

119.5 Signal definitions and reset values

The signals and their reset values are described in table 2289.

*Table 2289.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
resetn	Input	Reset	Low	

119.6 Timing

The timing waveforms and timing parameters are shown in figure 317 and are defined in table 2290.

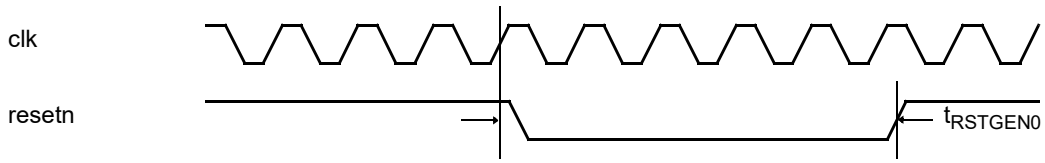


Figure 317. Timing waveforms

*Table 2290.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{RSTGEN0}$	asserted period	-	TBD	-	ns

Note: The *resetn* input is re-synchronized internally. The signals does not have to meet any setup or hold requirements.

119.7 Library dependencies

Table 2291 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2291.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	MISC	Component	Component definition

119.8 Instantiation

This example shows how the core can be instantiated together with the GRLIB clock generator.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;
library gaisler;
use gaisler.misc.all;

entity rstgen_ex is
  port (
    resetn  : in  std_ulogic;
    clk     : in  std_ulogic; -- 50 MHz main clock
    pllref  : in  std_ulogic;
    testrst : in std_ulogic;
    testen  : in std_ulogic
  );
end;

architecture example of rstgen_ex is

signal lclk, clkm, rstn, rstraw, sdclkl, clk50: std_ulogic;
signal cgi   : clkgen_in_type;
```

```
signal cgo   : clkgen_out_type;

begin
  cgi.pllctrl <= "00"; cgi.pllrst <= rstraw;
  pllref_pad : clkpad generic map (tech => padtech) port map (pllref, cgi.pllref);
  clk_pad : clkpad generic map (tech => padtech) port map (clk, lclk);
  clkgen0 : clkgen  -- clock generator
    generic map (clktech, CFG_CLKMUL, CFG_CLKDIV, CFG_MCTRL_SDEN,
                 CFG_CLK_NOFB, 0, 0, 0, BOARD_FREQ)
    port map (lclk, lclk, clkm, open, open, sdclkl, open, cgi, cgo, open, clk50);
  sdclk_pad : outpad generic map (tech => padtech, slew => 1, strength => 24)
    port map (sdclk, sdclkl);

  resetn_pad : inpad generic map (tech => padtech) port map (resetn, rst);

  rst0 : rstgen  -- reset generator
    generic map (acthigh => 0, syncrst => 0, scanen => 1)
    port map (rst, clkm, cgo.clklock, rstn, rstraw, testrst, testen);
end;
```

120 GR(2^4)(68, 60, 8, T=1) - QEC/QED error correction code encoder/decoder

120.1 Overview

The gf4_e1 VHDL package provides functions for encoding and decoding a Bose Chaudhuri Hocquenghem (BCH) type of code. It is a Quad Error Correction/Quad Error Detection (QEC/QED) code.

The data symbols are 4-bit wide, represented as GF(2^4). The has the capability to detect and correct a single symbol error anywhere in the codeword. The data is represented as 60 bits and the checksum is represented as 8 bits, and the code can correct up to four bit errors when located in the same nibble.

120.2 Code

The code has the following definition:

- there are 4 bits per symbol;
- there are 17 symbols per codeword, of which 2 symbols represent the checksum;
- the code is systematic;
- the code can correct one symbol error per codeword;
- the field polynomial is

$$f(x) = x^4 + x + 1$$

- all multiplications are performed as Galois Field multiplications over the above field polynomial
- all additions/subtractions are performed as Galois Field additions (i.e. bitwise exclusive-or)

120.3 Encoding

- a codeword is defined as 17 symbols:

  $[c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15}, c_{16}]$

  where c_0 to c_{14} represent information symbols and c_{15} to c_{16} represent check symbols.
- c_{15} is calculated as follows

$$c_{15} = \sum_{0}^{14} (k_i \times c_i)$$

- c_{16} is calculated as follows

$$c_{16} = \sum_{0}^{14} c_i$$

- where the constant vector k is defined as:

  k_0=0xF, k_1=0xE, ..., k_{14}=0x1 (one can assume k_{15}=0x1 and k_{16}=0x1 for correction purposes)

120.4 Decoding

- the corrupt codeword is defined as 17 symbols:

  $[r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}]$
- the corrupt codeword can also be defined as 17 uncorrupt symbols and an error:

  $[c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15}, c_{16}] + [e_x]$

  where the error is defined as e_x, e being the unknown magnitude and
  x being the unknown index position in the codeword

- recalculated checksum rc_0 is calculated as follows (k_i is as defined above, x being the unknown index)

$$rc_0 = \sum_{0}^{14}(k_i \times r_i) = \sum_{0}^{14}(k_i \times c_i) + (k_x \times e_x)$$

- recalculated rc_1 is calculated as follows

$$rc_1 = \sum_{0}^{14} r_i = \sum_{0}^{14} c_i + e_x$$

- syndrome s_0 is calculated as follows

$$s_0 = rc_0 + r_{15} = \sum_{0}^{14}(k_i \times r_i) + \sum_{0}^{14}(k_i \times c_i) = k_x \times e_x$$

- syndrome s_1 is calculated as follows, which gives the magnitude (not applicable to c_{15} and c_{16})

$$s_1 = rc_1 + r_{16} = \sum_{0}^{14} r_i + \sum_{0}^{14} c_i = e_x$$

- in case s_0 and s_1 are both non-zero, to located the error in range c_0 to c_{14}, multiply error magnitude e_x with each element of the constant vector defined above:

$$k_i e_x = k_i \times s_1 = k_i \times e_x \qquad i = [0,14]$$

- search the resulting vector to find the element matching syndrome s_0, the resulting index i points to the error location (applicable only to i in [0, 14])

$$k_i e_x \Leftrightarrow k_i \times e_x$$

- finally perform the correction (applicable only to i in [0, 14])

$$c_i = r_i - s_1 = r_i - e_x = (c_i - e_x) \times e_x = c_i - e_x + e_x = c_i$$

- when s_0 is zero and s_1 is non-zero, the error is located in checksum r_{15}, no correction is necessary
- when s_1 is zero and s_0 is non-zero, the error is located in checksum r_{16}, no correction is necessary
- when s_0 and s_1 are both zero, no error has been detected, no correction is necessary

120.5 Capability

The decoder has the following capabilities. It detects and corrects up to four bit errors in the same nibble. The described errors can be located anywhere in the codeword.

120.6 Operation

120.6.1 Encoder

The encoder is defined by the gf4_60_8_encode function. The function is called with the 60-bit wide data that should be encoded, and returns a 68-bit wide codeword of which bits 67 downto 8 represent the data and bits 7 downto 0 represent the checksum.

120.6.2 Decoder

The decoder is defined by the gf4_60_8_decode function.

The gf4_60_8_decode function calculates the syndromes, calculates the error magnitude and the error location, and returns a bit indicating whether an error has been detected and corrected, and the corrected data.

The function is called with a 68-bit wide codeword of which bits 67 downto 8 represent the data and bits 7 downto 0 represent the checksum. It returns the record type gf4_60_8_type, containing the 60-bit wide corrected data and an indication if an error was detected and corrected over the complete codeword.

120.7 Type descriptions

Table 2292 shows the type declarations used by the functions in the package (VHDL types).

*Table 2292.*Type declarations

Name	Field	Type	Function	Active
gf4_60_8_type	cerr	Std_Logic	error corrected	
	data	Std_Logic_Vector(59 downto 0)	data	

120.8 Library dependencies

Table 2293 shows the libraries used when instantiating the functions in the package (VHDL libraries).

*Table 2293.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	StdLib	All	Common VHDL functions

120.9 Instantiation

This example shows how the functions in the package can be instantiated. Note that all input and outputs are synchronized to remove any timing constraints for pads in an example design. Timing analysis can then be made purely for the register-to-register paths.

```
library  IEEE;
use      IEEE.Std_Logic_1164.all;

entity gf4_60_8_encode_sync is
   port(
      clk:        in       std_ulogic;
      data:       in       std_logic_vector(59 downto 0);
      codeword:   out      std_logic_vector(67 downto 0));
end entity gf4_60_8_encode_sync;

library  grlib;
use      grlib.gf4_e1.all;

architecture rtl of gf4_60_8_encode_sync is
   signal   int_data:      std_logic_vector(59 downto 0);
   signal   int_codeword:  std_logic_vector(67 downto 0);
begin
   process(clk)
   begin
      if rising_edge(clk) then
          codeword      <= int_codeword;
          int_codeword  <= gf4_60_8_encode(int_data);
          int_data      <= data;
      end if;
   end process;
end architecture;
```

```
library  IEEE;
use      IEEE.Std_Logic_1164.all;

entity gf4_60_8_decode_sync is
   port(
         clk:         in    std_ulogic;
         codeword:    in    std_logic_vector(67 downto 0);
         cerr:        out   std_ulogic;
         data:        out   std_logic_vector(59 downto 0));
end entity gf4_60_8_decode_sync;

library  grlib;
use      grlib.gf4_e1.all;

architecture rtl of gf4_60_8_decode_sync is
   signal   int_codeword:  std_logic_vector(67 downto 0);
   signal   int_result:    gf4_60_8_type;
begin
   process(clk)
   begin
      if rising_edge(clk) then
         cerr           <= int_result.cerr;
         data           <= int_result.data;
         int_result     <= gf4_60_8_decode(int_codeword);
         int_codeword   <= codeword;
      end if;
   end process;
end architecture;
```


Frontgrade Gaisler AB
Kungsgatan | SE-411 19 | Goteborg | Sweden
+46 31 7758650 | frontgrade.com/gaisler

121 RS(24, 16, 8, E=1) - Reed-Solomon encoder/decoder

121.1 Overview

The rs_gf4_e1 VHDL package provides functions for encoding and decoding data with a Reed-Solomon code. It also provides a data type storing intermediate results from the functions.

The Reed-Solomon data symbols are 4-bit wide, represented as GF(2^4). The Reed-Solomon code is a shortened RS(15, 13, 2) code, represented as RS(6, 4, 2). It has the capability to detect and correct a single symbol error anywhere in the codeword. The data is represented as 16 bits and the checksum is represented as 8 bits, and the code can correct 4-bit errors when located in the same nibble.

121.2 Capability

The Reed-Solomon decoder has the following capabilities. The described errors can be located anywhere in the codeword.

It detects and corrects any single bit error.

It detects 63% of all double bit errors and reports them as multiple bit errors.

It detects 27% of all double bit errors and reports them (incorrectly) as single bit errors.

It detects 63,5% of all triple bit errors and reports them as multiple bit errors.

It detects 36% of all triple bit errors and reports them (incorrectly) as single bit errors.

It does not detect 0,5% of all triple bit errors and reports them (incorrectly) as without errors.

It detects 63,5% of all quadruple bit errors and reports them as multiple bit errors.

It detects 36% of all quadruple bit errors and reports them (incorrectly) as single bit errors.

It does not detect 0,5% of all quadruple bit errors and reports them (incorrectly) as without error.

It detects and corrects up to four bit errors in the same nibble.

121.3 Operation

121.3.1 Encoder

The encoder is defined by the rs_16_8_encode function. The function is called with the 16-bit wide data that should be encoded, and returns 24-bit wide codeword of which bits 0 to 15 represent the data and bits 16 to 23 represent the checksum.

121.3.2 Decoder

The decoder is defined by the rs_16_8_check, rs_16_8_precorrect and rs_16_8_correct functions. The decoder has been split in three functions to facilitate pipelining, with each function being fairly balanced with respect to the depth of the resulting combinatorial logic.

The rs_16_8_check function calculates the syndrome and returns a bit indicating whether an error has been detected. Note that it can be any type of error: correctable or uncorrectable. The function is called with a 24-bit wide codeword of which bits 0 to 15 represent the data and bits 16 to 23 represent

the checksum. It returns the record type rs_16_8_type, containing the 16-bit wide data to be corrected, the syndrome and an indication if an error was detected over the complete codeword.

The rs_16_8_precorrect function is called with the intermediate result from the rs_16_8_check function. The input is the record type rs_16_8_type. It returns the record type rs_16_8_type, containing the 16-bit wide data to be corrected, the syndrome, intermediate data and an indication if an error was detected over the complete codeword.

The rs_16_8_correct function is called with the intermediate result from the rs_16_8_precorrect function. The input is the record type rs_16_8_type. It returns the record type rs_16_8_type, containing the corrected 16-bit wide data, an indication if the error was correctable or non-correctable over the complete codeword, and the union of the two.

To pipeline the decoder, the rs_16_8_check function should be called in the first stage and the intermediated result should be stored. Note that the intermediate result contains the input data required for the correction in the next stage. The rs_16_8_precorrect function should be called in the second stage. The rs_16_8_correct function should be called in the third stage.

121.4 Type descriptions

Table 2294 shows the type declarations used by the functions in the package (VHDL types).

*Table 2294.*Type declarations

Name	Field	Type	Function	Active
rs_16_8_type	err	Std_Logic	error detected	High
	cerr	Std_Logic	error corrected	
	merr	Std_Logic	errors uncorrected	
	data	Std_Logic_Vector(0 to 15)	data	
	s_1	Std_Logic_Vector(0 to 3)	-	-
	s_2	Std_Logic_Vector(0 to 3)	-	-
	elp_3_1	Std_Logic_Vector(0 to 3)	-	-

121.5 Library dependencies

Table 2295 shows the libraries used when instantiating the functions in the package (VHDL libraries).

*Table 2295.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	StdLib	All	Common VHDL functions

121.6 Instantiation

This example shows how the functions in the package can be instantiated.

The example design shows a codec for which the decoder is pipelined, with the error flag output one clock cycle earlier than the corrected data. Note that all input and outputs are synchronised to remove any timing constraints for pads in an example design. Timing analysis can be made purely for the register-to-register paths.

```
library  IEEE;
use      IEEE.Std_Logic_1164.all;

entity rs_gf4_16_8_codec is
   port(
      clk:        in    Std_Logic;
```

```
      din:        in    Std_Logic_Vector(0 to 15);       -- encoder input
      cout:       out   Std_Logic_Vector(0 to 23);       -- encoder output

      cin:        in    Std_Logic_Vector(0 to 23);       -- decoder input
      terr:       out   Std_Logic;                       -- intermediate error

      dout:       out   Std_Logic_Vector(0 to 15);       -- decoder output
      err:        out   Std_Logic;                       -- error detected
      cerr:       out   Std_Logic;                       -- error corrected
      merr:       out   Std_Logic);                      -- errors uncorrected
end entity;

library  grlib;
use      grlib.rs_gf4_e1.all;

architecture rtl of rs_gf4_16_8_codec is
   signal   s_din:      Std_Logic_Vector(0 to 15);
   signal   s_cout:     Std_Logic_Vector(0 to 23);

   signal   s_cin:      Std_Logic_Vector(0 to 23);
   signal   s_dout:     Std_Logic_Vector(0 to 15);
   signal   s_err:      Std_Logic;
   signal   s_cerr:     Std_Logic;
   signal   s_merr:     Std_Logic;

   signal   check:      rs_16_8_type;                    -- intermediate
   signal   precorr:    rs_16_8_type;
   signal   corr:       rs_16_8_type;
begin
   SyncronizeInput: process(clk)
   begin
      if Rising_Edge(clk) then
         s_din       <= din;
         s_cin       <= cin;
      end if;
   end process;

   SyncronizeOutput: process(clk)
   begin
      if Rising_Edge(clk) then
         cout        <= s_cout;
         err         <= corr.err;
         cerr        <= corr.cerr;
         merr        <= corr.merr;
         dout        <= corr.data;
         terr        <= check.err;
      end if;
   end process;

   encoder: process(clk)
   begin
      if Rising_Edge(clk) then
         s_cout      <= rs_16_8_encode(s_din);
      end if;
   end process;

   decoder: process(clk)
   begin
      if Rising_Edge(clk) then
         corr        <= rs_16_8_correct(precorr);     -- third phase
         precorr     <= rs_16_8_precorrect(check);    -- second phase
         check       <= rs_16_8_check(s_cin);         -- first phase
      end if;
   end process;
end architecture rtl;
```

122 RS(48, 32, 16, E=1+1) - Reed-Solomon encoder/decoder - interleaved

122.1 Overview

The rs_gf4_e1 VHDL package provides functions for encoding and decoding data with a Reed-Solomon code. It also provides a data type storing intermediate results from the functions.

The Reed-Solomon data symbols are 4-bit wide, represented as GF(2^4). The Reed-Solomon code is a shortened RS(15, 13, 2) code, represented as RS(6, 4, 2). It has the capability to detect and correct a single symbol error anywhere in the codeword. The data is represented as 16 bits and the checksum is represented as 8 bits, and the code can correct 4-bit errors when located in the same nibble.

The gf4_32_16 functions provide an interleaved RS(6, 4, 2) where the data is represented as 32 bits and the checksum is represented as 16 bits, and the code can correct two 4-bit errors when each error is located in a nibble and not in the same original RS(6, 4, 2) codeword. The codewords are interleaved nibble-wise.

122.2 Capability

The Reed-Solomon decoder has the same capabilities as the original RS(6, 4, 2) code, but distributed per original RS(6, 4, 2) codeword.

122.3 Operation

122.3.1 Encoder

The encoder is defined by the rs_32_16_encode function. The function is called with the 32-bit wide data that should be encoded, and returns 48-bit wide codeword of which bits 0 to 31 represent the data and bits 32 to 47 represent the checksum.

122.3.2 Decoder

The decoder is defined by the rs_32_16_check, rs_32_16_precorrect and rs_32_16_correct functions. The decoder has been split in three functions to facilitate pipelining, with each function being fairly balanced with respect to the depth of the resulting combinatorial logic.

The rs_32_16_check function calculates the syndrome and returns a bit indicating whether an error has been detected. Note that it can be any type of error: correctable or uncorrectable. The function is called with a 48-bit wide codeword of which bits 0 to 31 represent the data and bits 32 to 47 represent the checksum. It returns the record type rs_32_16_type, containing the 32-bit wide data to be corrected, the syndrome and an indication if an error was detected over the two complete codewords.

The rs_32_16_precorrect function is called with the intermediate result from the rs_32_16_check function. The input is the record type rs_32_16_type. It returns the record type rs_32_16_type, containing the 32-bit wide data to be corrected, the syndrome, intermediate data and an indication if an error was detected over the two complete codewords.

The rs_32_16_correct function is called with the intermediate result from the rs_32_16_precorrect function. The input is the record type rs_32_16_type. It returns the record type rs_32_16_type, containing the corrected 32-bit wide data, an indication if the error was correctable or non-correctable over the complete two codewords, and the union of the two.

To pipeline the decoder, the rs_32_16_check function should be called in the first stage and the intermediated result should be stored. Note that the intermediate result contains the input data required for the correction in the next stage. The rs_32_16_precorrect function should be called in the second stage. The rs_32_16_correct function should be called in the third stage.

122.4 Type descriptions

Table 2296 shows the type declarations used by the functions in the package (VHDL types).

*Table 2296.*Type declarations

Name	Field	Type	Function	Active
rs_32_16_type	err	Std_Logic	error detected	High
	cerr	Std_Logic	error corrected	
	merr	Std_Logic	errors uncorrected	
	data	Std_Logic_Vector(0 to 31)	data	
	e_0	Std_Logic	-	-
	s_1_0	Std_Logic_Vector(0 to 3)	-	-
	s_2_0	Std_Logic_Vector(0 to 3)	-	-
	elp_3_1_0	Std_Logic_Vector(0 to 3)	-	-
	e_1	Std_Logic	-	-
	s_1_1	Std_Logic_Vector(0 to 3)	-	-
	s_2_1	Std_Logic_Vector(0 to 3)	-	-
	elp_3_1_1	Std_Logic_Vector(0 to 3)	-	-

122.5 Library dependencies

Table 2297 shows the libraries used when instantiating the functions in the package (VHDL libraries).

*Table 2297.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	StdLib	All	Common VHDL functions

123 RS(40, 32, 8, E=1) - Reed-Solomon encoder/decoder

123.1 Overview

The rs_gf4_e1 VHDL package provides functions for encoding and decoding data with a Reed-Solomon code. It also provides a data type storing intermediate results from the functions.

The Reed-Solomon data symbols are 4-bit wide, represented as GF(2^4). The Reed-Solomon code is a shortened RS(15, 13, 2) code, represented as RS(10, 8, 2). It has the capability to detect and correct a single symbol error anywhere in the codeword. The data is represented as 32 bits and the checksum is represented as 8 bits, and the code can correct 4-bit errors when located in the same nibble.

123.2 Operation

123.2.1 Encoder

The encoder is defined by the rs_32_8_encode function. The function is called with the 32-bit wide data that should be encoded, and returns 40-bit wide codeword of which bits 0 to 31 represent the data and bits 32 to 39 represent the checksum.

123.2.2 Decoder

The decoder is defined by the rs_32_8_check, rs_32_8_precorrect and rs_32_8_correct functions. The decoder has been split in three functions to facilitate pipelining, with each function being fairly balanced with respect to the depth of the resulting combinatorial logic.

The rs_32_8_check function calculates the syndrome and returns a bit indicating whether an error has been detected. Note that it can be any type of error: correctable or uncorrectable. The function is called with a 24-bit wide codeword of which bits 0 to 31 representsthe data and bits 32 to 39 represent the checksum. It returns the record type rs_32_8_type, containing the 32-bit wide data to be corrected, the syndrome and an indication if an error was detected over the complete codeword.

The rs_32_8_precorrect function is called with the intermediate result from the rs_32_8_check function. The input is the record type rs_32_8_type. It returns the record type rs_32_8_type, containing the 32-bit wide data to be corrected, the syndrome, intermediate data and an indication if an error was detected over the complete codeword.

The rs_32_8_correct function is called with the intermediate result from the rs_32_8_precorrect function. The input is the record type rs_32_8_type. It returns the record type rs_32_8_type, containing the corrected 32-bit wide data, an indication if the error was correctable or non-correctable over the complete codeword, and the union of the two.

To pipeline the decoder, the rs_32_8_check function should be called in the first stage and the intermediated result should be stored. Note that the intermediate result contains the input data required for the correction in the next stage. The rs_32_8_precorrect function should be called in the second stage. The rs_32_8_correct function should be called in the third stage.

123.3 Type descriptions

Table 2298 shows the type declarations used by the functions in the package (VHDL types).

*Table 2298.*Type declarations

Name	Field	Type	Function	Active
rs_32_8_type	err	Std_Logic	error detected	High
	cerr	Std_Logic	error corrected	
	merr	Std_Logic	errors uncorrected	
	data	Std_Logic_Vector(0 to 31)	data	
	s_1	Std_Logic_Vector(0 to 3)	-	-
	s_2	Std_Logic_Vector(0 to 3)	-	-
	elp_3_1	Std_Logic_Vector(0 to 3)	-	-

123.4 Library dependencies

Table 2299 shows the libraries used when instantiating the functions in the package (VHDL libraries).

*Table 2299.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	StdLib	All	Common VHDL functions

123.5 Instantiation

This example shows how the functions in the package can be instantiated.

The example design shows a codec for which the decoder is pipelined, with the error flag output one clock cycle earlier than the corrected data. Note that all input and outputs are synchronised to remove any timing constraints for pads in an example design. Timing analysis can be made purely for the register-to-register paths.

```
library  IEEE;
use      IEEE.Std_Logic_1164.all;

entity rs_gf4_32_8_codec is
   port(
      clk:        in    Std_Logic;

      din:        in    Std_Logic_Vector(0 to 31);       -- encoder input
      cout:       out   Std_Logic_Vector(0 to 39);       -- encoder output

      cin:        in    Std_Logic_Vector(0 to 39);       -- decoder input
      terr:       out   Std_Logic;                       -- intermediate error

      dout:       out   Std_Logic_Vector(0 to 31);       -- decoder output
      err:        out   Std_Logic;                       -- error detected
      cerr:       out   Std_Logic;                       -- error corrected
      merr:       out   Std_Logic);                      -- errors uncorrected
end entity;

library  grlib;
use      grlib.rs_gf4_e1.all;

architecture rtl of rs_gf4_32_8_codec is
   signal   s_din:      Std_Logic_Vector(0 to 31);
   signal   s_cout:     Std_Logic_Vector(0 to 39);

   signal   s_cin:      Std_Logic_Vector(0 to 39);
   signal   s_dout:     Std_Logic_Vector(0 to 31);
```

```
   signal   s_err:      Std_Logic;
   signal   s_cerr:     Std_Logic;
   signal   s_merr:     Std_Logic;

   signal   check:      rs_32_8_type;                    -- intermediate
   signal   precorr:    rs_32_8_type;
   signal   corr:       rs_32_8_type;
begin
   SyncronizeInput: process(clk)
   begin
      if Rising_Edge(clk) then
         s_din       <= din;
         s_cin       <= cin;
      end if;
   end process;

   SyncronizeOutput: process(clk)
   begin
      if Rising_Edge(clk) then
         cout        <= s_cout;
         err         <= corr.err;
         cerr        <= corr.cerr;
         merr        <= corr.merr;
         dout        <= corr.data;
         terr        <= check.err;
      end if;
   end process;

   encoder: process(clk)
   begin
      if Rising_Edge(clk) then
         s_cout      <= rs_32_8_encode(s_din);
      end if;
   end process;

   decoder: process(clk)
   begin
      if Rising_Edge(clk) then
         corr        <= rs_32_8_correct(precorr);     -- third phase
         precorr     <= rs_32_8_precorrect(check);    -- second phase
         check       <= rs_32_8_check(s_cin);         -- first phase
      end if;
   end process;
end architecture rtl;
```

124 RS(48, 32, 16, E=2) - Reed-Solomon encoder/decoder

124.1 Overview

The rs_gf4_e2 VHDL package provides functions for encoding and decoding data with a Reed-Solomon code. It also provides a data type storing intermediate results from the functions.

The Reed-Solomon data symbols are 4-bit wide, represented as GF(2^4). The Reed-Solomon code is a shortened RS(15, 11, 4) code, represented as RS(12, 8, 4). It has the capability to detect and correct two symbol errors anywhere in the codeword. The data is represented as 32 bits and the checksum is represented as 16 bits, and the code can correct up to two 4-bit errors when located within nibble boundaries.

124.2 Operation

124.2.1 Encoder

The encoder is defined by the rs_32_16_2_encode function. The function is called with the 32-bit wide data that should be encoded, and returns 48-bit wide codeword of which bits 0 to 31 represent the data and bits 32 to 47 represent the checksum.

124.2.2 Decoder

The decoder is defined by the rs_32_16_2_check, rs_32_16_2_precorrect and rs_32_16_2_correct functions. The decoder has been split in three functions to facilitate pipelining, with each function being fairly balanced with respect to the depth of the resulting combinatorial logic.

The rs_32_16_2_check function calculates the syndrome and returns a bit indicating whether an error has been detected. Note that it can be any type of error: correctable or uncorrectable. The function is called with a 48-bit wide codeword of which bits 0 to 31 represent the data and bits 32 to 47 represent the checksum. It returns the record type rs_32_16_2_type, containing the 32-bit wide data to be corrected, the syndrome and an indication if an error was detected.

The rs_32_16_2_precorrect function is called with the intermediate result from the rs_32_16_2_check function. The input is the record type rs_32_16_2_type. It returns the record type rs_32_16_2_type, containing the 32-bit wide data to be corrected, the syndrome, intermediate data and an indication if an error was detected.

The rs_32_16_2_correct function is called with the intermediate result from the rs_32_16_2_precorrect function. The input is the record type rs_32_16_2_type. It returns the record type rs_32_16_2_type, containing the corrected 32-bit wide data, an indication if the error was correctable or non-correctable, and the union of the two.

To pipeline the decoder, the rs_32_16_2_check function should be called in the first stage and the intermediated result should be stored. Note that the intermediate result contains the input data required for the correction in the next stage. The rs_32_16_2_precorrect function should be called in the second stage. The rs_32_16_2_correct function should be called in the third stage.

124.3 Type descriptions

Table 2300 shows the type declarations used by the functions in the package (VHDL types).

*Table 2300.*Type declarations

Name	Field	Type	Function	Active
rs_32_16_2_type	err	Std_Logic	errors detected	High
	cerr	Std_Logic	errors corrected	High
	merr	Std_Logic	errors uncorrected	High
	data	Std_Logic_Vector(0 to 31)	data	
	l_u	Std_Logic_Vector(0 to 1)	indicates number of detected errors, only for rs_32_16_2_correct function	
	s_1	Std_Logic_Vector(0 to 3)		
	s_2	Std_Logic_Vector(0 to 3)		
	s_3	Std_Logic_Vector(0 to 3)		
	s_4	Std_Logic_Vector(0 to 3)		
	elp_5_1	Std_Logic_Vector(0 to 3)	indicates index of detected error in code-word, only for rs_32_16_2_correct function: 0x0 = codeword(0:3) 0x1 = codeword(4:7) 0x2 = codeword(8:11) 0x3 = codeword(12:15) 0x4 = codeword(16:19) 0x5 = codeword(20:23) 0x6 = codeword(24:27) 0x7 = codeword(28:31) 0x8 = codeword(32:35) 0x9 = codeword(36:39) 0xA = codeword(40:43) 0xB = codeword(44:47) 0xF = unidentified (can be used with erasure information)	
	elp_5_2	Std_Logic_Vector(0 to 3)	indicates index of detected error, as above	

124.4 Library dependencies

Table 2301 shows the libraries used when instantiating the functions in the package (VHDL libraries).

*Table 2301.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	StdLib	All	Common VHDL functions

124.5 Instantiation

This example shows how the functions in the package can be instantiated.

The example design shows a codec for which the decoder is pipelined, with the error flag output one clock cycle earlier than the corrected data. Note that all input and outputs are synchronised to remove

any timing constraints for pads in an example design. Timing analysis can be made purely for the register-to-register paths.

```
library  IEEE;
use      IEEE.Std_Logic_1164.all;

entity rs_gf4_32_16_2_codec is
   port(
      clk:        in    Std_Logic;

      din:        in    Std_Logic_Vector(0 to 31);       -- encoder input
      cout:       out   Std_Logic_Vector(0 to 47);       -- encoder output

      cin:        in    Std_Logic_Vector(0 to 47);       -- decoder input
      terr:       out   Std_Logic;                       -- intermediate error

      dout:       out   Std_Logic_Vector(0 to 31);       -- decoder output
      err:        out   Std_Logic;                       -- error detected
      cerr:       out   Std_Logic;                       -- error corrected
      merr:       out   Std_Logic);                      -- errors uncorrected
end entity;

library  grlib;
use      grlib.rs_gf4_e2.all;

architecture rtl of rs_gf4_32_16_2_codec is
   signal   s_din:      Std_Logic_Vector(0 to 31);
   signal   s_cout:     Std_Logic_Vector(0 to 47);
   signal   s_cin:      Std_Logic_Vector(0 to 47);
   signal   s_dout:     Std_Logic_Vector(0 to 31);
   signal   s_err:      Std_Logic;
   signal   s_cerr:     Std_Logic;
   signal   s_merr:     Std_Logic;
   signal   check:      rs_32_16_2_type;                 -- intermediate
   signal   precorr:    rs_32_16_2_type;
   signal   corr:       rs_32_16_2_type;
begin
   SyncronizeInput: process(clk)
   begin
      if Rising_Edge(clk) then
         s_din       <= din;
         s_cin       <= cin;
      end if;
   end process;

   SyncronizeOutput: process(clk)
   begin
      if Rising_Edge(clk) then
         cout        <= s_cout;
         err         <= corr.err;
         cerr        <= corr.cerr;
         merr        <= corr.merr;
         dout        <= corr.data;
         terr        <= check.err;
      end if;
   end process;

   encoder: process(clk)
   begin
      if Rising_Edge(clk) then
         s_cout      <= rs_32_16_2_encode(s_din);
      end if;
   end process;

   decoder: process(clk)
   begin
      if Rising_Edge(clk) then
         corr        <= rs_32_16_2_correct(precorr);     -- third phase
         precorr     <= rs_32_16_2_precorrect(check);    -- second phase
```

```
            check       <= rs_32_16_2_check(s_cin);         -- first phase
          end if;
        end process;
      end architecture rtl;
```

125 SDCTRL - 32/64-bit PC133 SDRAM Controller

125.1 Overview

The SDRAM controller handles PC133 SDRAM compatible memory devices attached to a 32 or 64 bit wide data bus. The controller acts as a slave on the AHB bus where it occupies a configurable amount of address space for SDRAM access. The SDRAM controller function is programmed by writing to a configuration register mapped into AHB I/O address space.

Chip-select decoding is provided for two SDRAM banks.

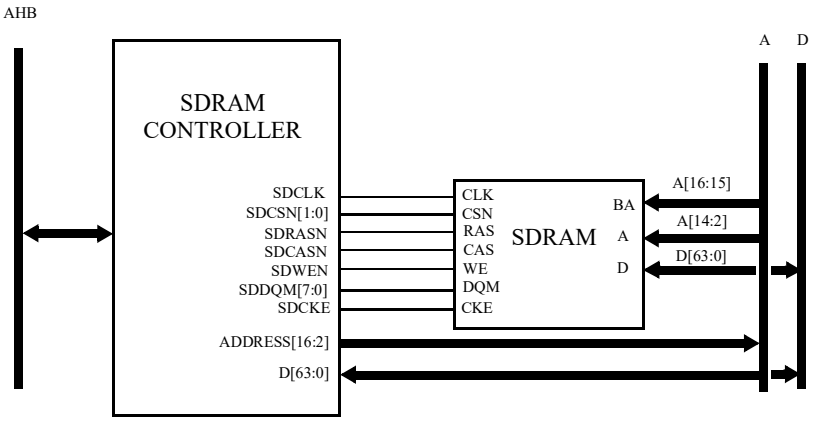


Figure 318. SDRAM Memory controller connected to AMBA bus and SDRAM

125.2 Operation

125.2.1 General

Synchronous dynamic RAM (SDRAM) access is supported to two banks of PC100/PC133 compatible devices. The controller supports 64M, 256M and 512M devices with 8 - 12 column-address bits, up to 13 row-address bits, and 4 banks. The size of each of the two banks can be programmed in binary steps between 4 Mbyte and 512 Mbyte. The operation of the SDRAM controller is controlled through the configuration register SDCFG (see section 125.3). The SDRAM bank's data bus width is configurable between 32 and 64 bits. When the VHDL generic *mobile* is set to a value not equal to 0, the controller supports mobile SDRAM.

125.2.2 Initialization

When the SDRAM controller is enabled, it automatically performs the SDRAM initialization sequence of PRECHARGE, 8x AUTO-REFRESH and LOAD-MODE-REG on both banks simultaneously. When mobile SDRAM functionality is enabled the initialization sequence is appended with a LOAD-EXTMODE-REG command. The controller programs the SDRAM to use page burst on read accesses and single location access on write accesses. If the *pwron* VHDL generic is 1, the initialization sequence is also sent automatically when reset is released. Note that some SDRAM devices require a stable clock of 100 us before any commands might be sent. When using on-chip PLL, this might not always be the case and the *pwron* VHDL generic should be set to 0 in such cases.

125.2.3 Configurable SDRAM timing parameters

To provide optimum access cycles for different SDRAM devices (and at different frequencies), three SDRAM parameters can be programmed through memory configuration register 2 (MCFG2): TCAS, TRP and TRFCD. The value of these fields affect the SDRAM timing as described in table 2302.

*Table 2302.*SDRAM programmable minimum timing parameters

SDRAM timing parameter	Minimum timing (clocks)
CAS latency, RAS/CAS delay (t_{CAS}, t_{RCD})	TCAS + 2
Precharge to activate (t_{RP})	TRP + 2
Auto-refresh command period (t_{RFC})	TRFC + 3
Activate to precharge (t_{RAS})	TRFC + 1
Activate to Activate (t_{RC})	TRP + TRFC + 4

If the TCAS, TRP and TRFC are programmed such that the PC100/133 specifications are fulfilled, the remaining SDRAM timing parameters will also be met. The table below shows typical settings for 100 and 133 MHz operation and the resulting SDRAM timing (in ns):

*Table 2303.*SDRAM example programming

SDRAM settings	t_{CAS}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}
100 MHz, CL=2; TRP=0, TCAS=0, TRFC=4	20	80	20	70	50
100 MHz, CL=3; TRP=0, TCAS=1, TRFC=4	30	80	20	70	50
133 MHz, CL=2; TRP=1, TCAS=0, TRFC=6	15	82	22	67	52
133 MHz, CL=3; TRP=1, TCAS=1, TRFC=6	22	82	22	67	52

When mobile SDRAM support is enabled, one additional timing parameter (TXSR) can be programmed though the Power-Saving configuration register.

*Table 2304.*Mobile SDRAM programmable minimum timing parameters

SDRAM timing parameter	Minimum timing (clocks)
Exit Self Refresh mode to first valid command (t_{XSR})	tXSR

125.2.4 Refresh

The SDRAM controller contains a refresh function that periodically issues an AUTO-REFRESH command to both SDRAM banks. The period between the commands (in clock periods) is programmed in the refresh counter reload field in the SDCFG register. Depending on SDRAM type, the required period is typically 7.8 or 15.6 µs (corresponding to 780 or 1560 clocks at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by setting bit 31 in SDCFG register.

125.2.5 Self Refresh

The self refresh mode can be used to retain data in the SDRAM even when the rest of the system is powered down. When in the self refresh mode, the SDRAM retains data without external clocking and refresh are handled internally. The memory array that is refreshed during the self refresh operation is defined in the extended mode register. These settings can be changed by setting the PASR bits in the Power-Saving configuration register. The extended mode register is automatically updated

when the PASR bits are changed. The supported “Partial Array Self Refresh” modes are: Full, Half, Quarter, Eighth, and Sixteenth array. “Partial Array Self Refresh” is only supported when mobile SDRAM functionality is enabled. To enable the self refresh mode, set the PMODE bits in the Power-Saving configuration register to “010” (Self Refresh). The controller will enter self refresh mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits are cleared. When exiting this mode the controller introduce a delay defined by tXSR in the Power-Saving configuration register and a AUTO REFRESH command before any other memory access is allowed. The minimum duration of this mode is defined by tRAS. This mode is only available when the VHDL generic *mobile* is >= 1.

125.2.6 Power-Down

When entering the power-down mode all input and output buffers, excluding SDCKE, are deactivated. All data in the SDRAM is retained during this operation. To enable the power-down mode, set the PMODE bits in the Power-Saving configuration register to “001” (Power-Down). The controller will enter power-down mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits is cleared. The REFRESH command will still be issued by the controller in this mode. When exiting this mode a delay of one clock cycles are added before issue any command to the memory. This mode is only available when the VHDL generic *mobile* is >= 1.

125.2.7 Deep Power-Down

The deep power-down operating mode is used to achieve maximum power reduction by eliminating the power of the memory array. Data will not be retained after the device enters deep power-down mode. To enable the deep power-down mode, set the PMODE bits in the Power-Saving configuration register to “101” (Deep Power-Down). To exit the deep power-down mode the PMODE bits in the Power-Saving configuration register must be cleared. The controller will respond with an AMBA ERROR response to an AMBA access, that will result in a memory access, during Deep Power-Down mode. This mode is only available when the VHDL generic *mobile* is >= 1 and mobile SDRAM functionality is enabled.

125.2.8 Temperature-Compensated Self Refresh

The settings for the temperature-compensation of the Self Refresh rate can be controlled by setting the TCSR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the TCSR bits are changed. Note that some vendors implements a Internal Temperature-Compensated Self Refresh feature, which makes the memory ignore the TCSR bits. This functionality is only available when the VHDL generic *mobile* is >= 1 and mobile SDRAM functionality is enabled.

125.2.9 Drive Strength

The drive strength of the output buffers can be controlled by setting the DS bits in the Power-Saving configuration register. The extended mode register is automatically updated when the DS bits are changed. The available options are: full, three-quarter, one-half, and one-quarter drive strengths. This functionality is only available when the VHDL generic *mobile* is >= 1 and mobile SDRAM functionality is enabled.

125.2.10SDRAM commands

The controller can issue four SDRAM commands by writing to the SDRAM command field in the SDRAM Configuration register: PRE-CHARGE, AUTO-REFRESH, LOAD-MODE-REG (LMR) and LOAD-EXTMODE-REG (EMR). If the LMR command is issued, the CAS delay as programmed in SDCFG will be used, remaining fields are fixed: page read burst, single location write, sequential burst. If the EMR command is issued, the DS, TCSR and PASR as programmed in Power-Saving configuration register will be used. The command field will be cleared after a command has been exe-

cuted. Note that when changing the value of the CAS delay, a LOAD-MODE-REGISTER command should be generated at the same time.

125.2.11Read cycles

A read cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a READ command with data read after the programmed CAS delay. A read burst is performed if a burst access has been requested on the AHB bus. The read cycle is terminated with a PRE-CHARGE command, no banks are left open between two accesses. Note that only word bursts are supported by the SDRAM controller. The AHB bus supports bursts of different sizes such as bytes and half-words but they cannot be used.

125.2.12Write cycles

Write cycles are performed similarly to read cycles, with the difference that WRITE commands are issued after activation. A write burst on the AHB bus will generate a burst of write commands without idle cycles in-between. As in the read case, only word bursts are supported.

125.2.13Address bus connection

The SDRAM address bus should be connected to SA[12:0], the bank address to SA[14:13], and the data bus to SD[31:0] or SD[63:0] if a 64-bit SDRAM data bus is used.

125.2.14Data bus

The external SDRAM data bus is configurable to either 32 or 64 bits width, using the *sdbits* VHDL generic. A 64-bit data bus allows 64-bit (SO)DIMMs to be connected using the full data capacity of the devices. The polarity of the output enable signal to the data pads can be selected with the oepol generic. Sometimes it is difficult to fulfil the output delay requirements of the output enable signal. In this case, the vbdrive signal can be used instead of bdrive. Each index in this vector is driven by a separate register and a directive is placed on them so that they will not be removed by the synthesis tool.

125.2.15Clocking

The SDRAM controller is designed for an external SDRAM clock that is in phase or slightly earlier than the internal AHB clock. This provides the maximum margin for setup and hold on the external signals, and allows highest possible frequency. For Xilinx and Altera devices, the GRLIB Clock Generator (CLKGEN) can be configured to produce a properly synchronized SDRAM clock. For other FPGA targets, the custom clock synchronization must be designed, or the inverted clock option can be used (see below). For ASIC targets, the SDRAM clock can be derived from the AHB clock with proper delay adjustments during place&route.

If the VHDL generic INVCLK is set, then all outputs from the SDRAM controller are delayed for 1/2 clock. This is done by clocking all output registers on the falling clock edge. This option can be used on FPGA targets where proper SDRAM clock synchronization cannot be achieved. The SDRAM clock can be the internal AHB clock without further phase adjustments. Since the SDRAM signals will only have 1/2 clock period to propagate, this option typically limits the maximum SDRAM frequency to 40 - 50 MHz.

125.2.16Endianness

The core is designed for big-endian systems.

125.3 Registers

The memory controller is programmed through register(s) mapped into the AHB I/O space defined by the controllers AHB BAR1. Only 32-bit single-accesses to the registers are supported.

Table 2305. SDRAM controller registers

AHB address offset	Register
0x0	SDRAM Configuration register
0x4	SDRAM Power-Saving configuration register

Table 2306. 0x00 - SDCFG1 - SDRAM configuration register

31	30	29 27	26	25 23	22 21	20 18	17	16	15	14 0
Refresh	tRP	tRFC	tCD	SDRAM bank size	SDRAM col. size	SDRAM command	Page-Burst	MS	D64	SDRAM refresh load value
0	1	0b111	1	0	0b10	0	*	*	*	NR
rw	rw	rw	rw	rw	rw	rw	rw*	r	r	rw

31 SDRAM refresh. If set, the SDRAM refresh will be enabled.

30 SDRAM tRP timing. tRP will be equal to 2 or 3 system clocks (0/1). When mobile SDRAM support is enabled, this bit also represent the MSB in the tRFC timing.

29: 27 SDRAM tRFC timing. tRFC will be equal to 3 + field-value system clocks. When mobile SDRAM support is enabled, this field is extended with the bit 30.

26 SDRAM CAS delay. Selects 2 or 3 cycle CAS delay (0/1). When changed, a LOAD-MODE-REGISTER command must be issued at the same time. Also sets RAS/CAS delay (tRCD).

25: 23 SDRAM banks size. Defines the decoded memory size for each SDRAM chip select: “000”= 4 Mbyte, “001”= 8 Mbyte, “010”= 16 Mbyte .... “111”= 512 Mbyte.

When configured for 64-bit wide SDRAM data bus (sdbits=64), the meaning of this field doubles so that “000”=8 Mbyte, .., “111”=1024 Mbyte

22: 21 SDRAM column size. “00”=256, “01”=512, “10”=1024, “11”=2048 except when bit[25:23]=˜111˜ then ˜11˜=4096

20: 18 SDRAM command. Writing a non-zero value will generate an SDRAM command: “010”=PRECHARGE, “100”=AUTO-REFRESH, “110”=LOAD-MODE-REGISTER, “111”=LOAD-EXT-MODE-REGISTER. The field is reset after command has been executed.

17 1 = pageburst is used for read operations, 0 = line burst of length 8 is used for read operations. (Only available when VHDL generic pageburst i set to 2)

16 Mobile SDR support enabled. ‘1’ = Enabled, ‘0’ = Disabled (read-only)

15 64-bit data bus (D64) - Reads ‘1’ if memory controller is configured for 64-bit data bus, otherwise ‘0’. Read-only.

14: 0 The period between each AUTO-REFRESH command - Calculated as follows: tREFRESH = ((reload value) + 1) / SYSCLK

Table 2307. 0x04 - SDCFG2 - SDRAM Power-Saving configuration register

31	30	29 24	23 20	19	18 16	15 7	6	5	4	3	2 0
ME	CE	RESERVED	tXSR	R	PMODE	RESERVED		DS	TCSR		PASR
*	*	0	*	0	0	0		0	0		0
rw*	rw*	r	rw*	r	rw	r		rw	rw		rw

31 Mobile SDRAM functionality enabled. ‘1’ = Enabled (support for Mobile SDRAM), ‘0’ = disabled (support for standard SDRAM)

30 Clock enable (CE). This value is driven on the CKE inputs of the SDRAM. Should be set to ‘1’ for correct operation. This register bit is read only when Power-Saving mode is other then none.

29: 24 Reserved

*Table 2307.*0x04 - SDCFG2 - SDRAM Power-Saving configuration register

23: 20	SDRAM tXSR timing. tXSR will be equal to field-value system clocks. (Read only when Mobile SDR support is disabled).
19	Reserved
18: 16	Power-Saving mode (Read only when Mobile SDR support is disabled). “000”: none “001”: Power-Down (PD) “010”: Self-Refresh (SR) “101”: Deep Power-Down (DPD)
15: 7	Reserved
6: 5	Selectable output drive strength (Read only when Mobile SDR support is disabled). “00”: Full “01”: One-half “10”: One-quarter “11”: Three-quarter
4: 3	Reserved for Temperature-Compensated Self Refresh (Read only when Mobile SDR support is disabled). “00”: 70ªC “01”: 45ªC “10”: 15ªC “11”: 85ªC
2: 0	Partial Array Self Refresh (Read only when Mobile SDR support is disabled). “000”: Full array (Banks 0, 1, 2 and 3) “001”: Half array (Banks 0 and 1) “010”: Quarter array (Bank 0) “101”: One-eighth array (Bank 0 with row MSB = 0) “110”: One-sixteenth array (Bank 0 with row MSB = 00)

125.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x009. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

125.5 Implementation

125.5.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User’s Manual). The core makes use of synchronous reset and resets a subset of its internal registers. The registers driving SDRAM chip select and output enables for the SDRAM data bus have asynchronous reset.

125.5.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User’s Manual). The core is designed for big-endian systems.

125.6 Configuration options

Table 2308 shows the configuration options of the core (VHDL generics).

*Table 2308.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR0 defining SDRAM area. Default is 0xF0000000 - 0xFFFFFFFF.	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0 defining SDRAM area.	0 - 16#FFF#	16#F00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space where SDCFG register is mapped.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR1 defining I/O address space.	0 - 16#FFF#	16#FFF#
wprot	Write protection.	0 - 1	0
invclk	Inverted clock is used for the SDRAM.	0 - 1	0
pwron	Enable SDRAM at power-on initialization	0 - 1	0
sdbits	32 or 64-bit data bus width.	32, 64	32
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
pageburst	Enable SDRAM page burst operation. 0: Controller uses line burst of length 8 for read operations. 1: Controller uses pageburst for read operations. 2: Controller uses pageburst/line burst depending on PageBurst bit in SDRAM configuration register.	0 - 2	0
mobile	Enable Mobile SDRAM support 0: Mobile SDRAM support disabled 1: Mobile SDRAM support enabled but not default 2: Mobile SDRAM support enabled by default 3: Mobile SDRAM support only (no regular SDR support)	0 - 3	0

125.7 Signal descriptions

Table 2309 shows the interface signals of the core (VHDL ports).

*Table 2309.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
AHBSI	1)	Input	AHB slave input signals	-
AHBSO	1)	Output	AHB slave output signals	-
SDI	WPROT	Input	Not used	-
	DATA[63:0]	Input	Data	High
SDO	SDCKE[1:0]	Output	SDRAM clock enable	High
	SDCSN[1:0]	Output	SDRAM chip select	Low
	SDWEN	Output	SDRAM write enable	Low
	RASN	Output	SDRAM row address strobe	Low
	CASN	Output	SDRAM column address strobe	Low
	DQM[7:0]	Output	SDRAM data mask: DQM[7] corresponds to DATA[63:56], DQM[6] corresponds to DATA[55:48], DQM[5] corresponds to DATA[47:40], DQM[4] corresponds to DATA[39:32], DQM[3] corresponds to DATA[31:24], DQM[2] corresponds to DATA[23:16], DQM[1] corresponds to DATA[15:8], DQM[0] corresponds to DATA[7:0].	Low
	BDRIVE	Output	Drive SDRAM data bus	Low/High[2]
	VBDRIVE[31:0]	Output	Identical to BDRIVE but has one signal for each data bit. Every index is driven by its own register. This can be used to reduce the output delay.	Low/High[2]
	ADDRESS[16:2]	Output	SDRAM address	Low
	DATA[31:0]	Output	SDRAM data	Low

1) see GRLIB IP Library User's Manual

2) Polarity selected with the oepol generic

125.8 Library dependencies

Table 2310 shows libraries used when instantiating the core (VHDL libraries).

*Table 2310.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

125.9 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the SDRAM controller. The external SDRAM bus is defined on the example designs port map and connected to the SDRAM controller. System clock and reset are generated by GR Clock Generator and Reset Generator.

SDRAM controller decodes SDRAM area:0x60000000 - 0x6FFFFFFF. SDRAM Configuration register is mapped into AHB I/O space on address (AHB I/O base address + 0x100).

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;
sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0) -- optional sdram data
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

signal sdi   : sdctrl_in_type;
  signal sdo   : sdctrl_out_type;

  signal clkm, rstn : std_ulogic;
signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;
  signal gnd : std_ulogic;

begin

  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                                tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  rst0 : rstgen
  port map (resetn, clkm, cgo.clklock, rstn);

  -- SDRAM controller
  sdc : sdctrl generic map (hindex => 3, haddr => 16#600#, hmask => 16#F00#,
    ioaddr => 1, pwron => 0, invclk => 0)
```

```
    port map (rstn, clkm, ahbsi, ahbso(3), sdi, sdo);

  -- input signals
  sdi.data(31 downto 0) <= sd(31 downto 0);

  -- connect SDRAM controller outputs to entity output signals
  sa <= sdo.address; sdcke <= sdo.sdcke; sdwen <= sdo.sdwen;
  sdcsn <= sdo.sdcsn; sdrasn <= sdo.rasn; sdcasn <= sdo.casn;
  sddqm <= sdo.dqm;

--Data pad instantiation with scalar bdrive
sd_pad : iopadv generic map (width => 32)
port map (sd(31 downto 0), sdo.data, sdo.bdrive, sdi.data(31 downto 0));
end;

--Alternative data pad instantiation with vectored bdrive
sd_pad : iopadvv generic map (width => 32)
port map (sd(31 downto 0), sdo.data, sdo.vbdrive, sdi.data(31 downto 0));
end;
```

126 SOCBRIDGE - AHB to AHB interface through byte stream

126.1 Overview

The SOCBRIDGE interfaces an AHB bus with a slave and a master port, and on the other side provides a bidirectional bytestream. Accesses to the slave port are transmitted over the byte stream. Incoming accesses over the byte stream are performed on the master port. By transferring the byte stream to another device with a corresponding SOCBRIDGE, accesses can be tunneled between the two devices. The interface may be used an expansion interface with additional I/O provided in an external companion chip or to implement a simple data transfer interface between the system-on-chip and an external device.

The socbridge can be configured to operate in *32b mode*, where the byte stream is replaced by a 32-bit wide stream instead. This mode is not practical for connecting two separate devices due to the high pin count. Instead, it is useful when connecting two socbridge cores that are located within the same device and if higher throughput is needed than what the normal byte stream mode can provide.

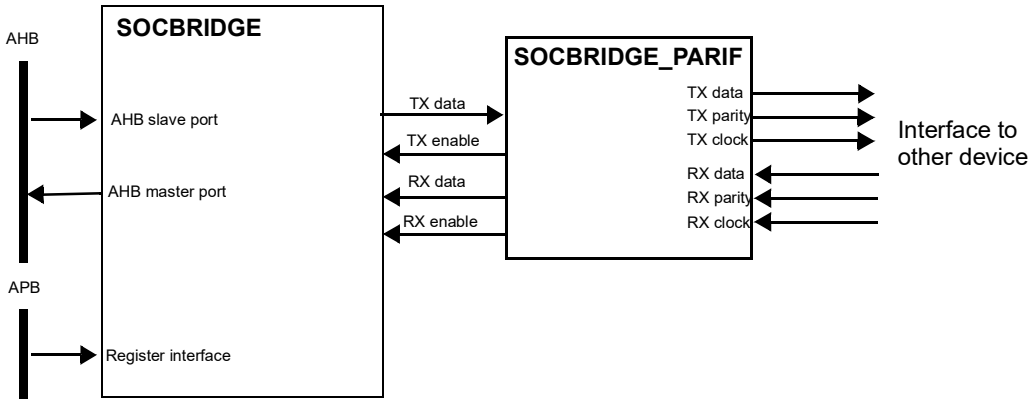


Figure 319. Block diagram of SOCBRIDGE together with parallel interface.

126.2 Communication protocol

126.2.1 Introduction

The socbridge uses an interface based on byte streams to communicate with the other endpoint. A continuous stream of bytes are flowing in both directions. The streams are usually at the same data rate but this is not required, and the streams may have arbitrary latency/pipelining.

126.2.2 Byte stream protocol

Over the byte stream, the socbridge communicates with a simple custom protocol. This section is intended to provide a free-standing description of the protocol that is implemented in this interface.

The byte stream based protocol used by the socbridge allows six different access types: Idle, Write with address and data, write with only data, read with address, read without address, and peer error indication. There are also two types of responses sent back: write acknowledge and read response with data. The format of the commands is tabulated below. Any bit pattern not covered by the table is considered reserved for future use.

*Table 2311.*Command and response bytes

Command	7	6	5	4	3	2	1	0	Followed by address	Followed by data	Response expected
Commands											
Idle	0	0	0	0	0	0	0	0	No	No	None
Write with address	1	0	0	0	0	size[1]			Yes, 4 bytes	Yes, according to size	Write acknowledge
Write without address	1	0	1	0	0	size[1]			No	Yes, according to size	Write acknowledge
Read with address	1	1	0	0	0	size[1]			Yes, 4 bytes	No	Read response
Read without address	1	1	1	0	0	size[1]			No	No	Read response
Peer error	0	1	0	0	1	0	0	0	No	No	None
Responses											
Write acknowledge	0	0	x[2]	0	1	size[1,2]			No	No	-
Read response	0	1	x[2]	0	0	size[1,2]			No	Yes, according to size	-

Notes:
1) size is given as power of two, 000 meaning 1 byte, 001 meaning 2 bytes, 010 meaning 4 bytes, and so on.
2) bit 5 and bits 2:0 in response must match command

Idle is the default command sent when there is nothing to do, or when the socbridge is not yet ready to send a response back.

Writes and read accesses are performed by sending the corresponding command, and then getting a response back from the other end.

An access may be issued either without or with an address, depending on bit 5 of the command. If an address is included, it is sent with four bytes immediately after the command byte with the most significant address byte (address bits 31:24) sent first. The address must always be aligned to the size of the transfer command.

When a transfer command without address is issued, the access is assumed to begin on the first byte after where the previous access ended. This allows for long continuous transfers of data without having to re-send the address in each command. The first read or write transfer made over the link after reset must always provide an address in order to have a known starting point. It is allowed to perform a write without address after a read or a read without address after a write, and the size of the access is not required to match the previous one. However the commands must still be sent so that the effective starting address is aligned to the access size also for accesses without address.

The endpoint performing the access must always provide responses in the same order as the commands were issued, reordering the responses is not permitted. The response must have the same value on bit 5 and bits 2:0 as the command, the purpose of this requirement is to simplify tracking of available buffer space.

The Peer error command is used to inform the other endpoint that a fatal error was detected on the link. The protocol itself does not provide any means to acknowledge or indicate that the error has been recovered, that is managed separately.

The order in which bytes are sent during the data phase respects endianness. This means the most significant data byte is sent first when configured as big endian, and last when configured as little endian.

In *32b mode*, the order in which bytes are sent is the same as in the default configuration, except that four bytes are sent at once instead of one. The first byte to be sent is placed on data lines [31:24], with the following bytes on descending data lines. If less than four bytes of data is to be sent, then the valid data is expected to be replicated over the entire 32b word. Additionally, in *32b mode*, the command

byte is never sent together with any other data. The command is expected to be on data lines [7:0], and the rest of the word is expected to be filled with all zeroes.

126.2.3 Implementation-specific features

Not all endpoints are expected to support the maximum size supported by the protocol, sizes of 1,2 and 4 bytes are expected to be supported by all implementations. The protocol does not provide any way to auto-detect what is supported so this has to be managed separately. The protocol also does not provide information on which address ranges are supported and where different addresses are mapped on the other end, this knowledge has to be managed separately.

Additional commands may be issued before a response has been received to the previous command, but it is the command issuer's responsibility to make sure that the other endpoint has enough space in its receive buffers to queue up the command. The protocol itself does not provide any mechanism to specify whether there is room for more commands. If the capabilities of the other endpoint is unknown then the only strategy that is guaranteed to not overflow the other endpoint is to wait for the response after each access before issuing another access.

126.2.4 Parallel interface

In order to transfer the byte streams between the two devices some form of physical layer has to be provided. The standard parallel interface transfers the bytes over an 8-bit parallel bus, together with an optional parity bit and a clock signal in each direction. In total the interface will communicate with 20 signals over the board. The parity signal implements odd parity, i.e the parity line is the inverse of the exclusive-or of the eight data lines.

The parallel interface is possible to be configured in a *32b mode*, then the data is transfered over an 32-bit parallel bus instead and 68 signals is required in total.

One of the two sides of the parallel interface is designated as the primary side, with the other side designated as the secondary side. The primary side determines which clock frequency is to be used for the interface and provides the clock with rising edge centered in the data eye of each outgoing data byte. The secondary side delivers a data byte in the other direction for each rising edge of the clock received and at the same time toggles its outgoing clock, creating a double data-rate clock with the clock aligned to the beginning of the data eye. The reason for this difference is to allow for simple implementations on the secondary side.

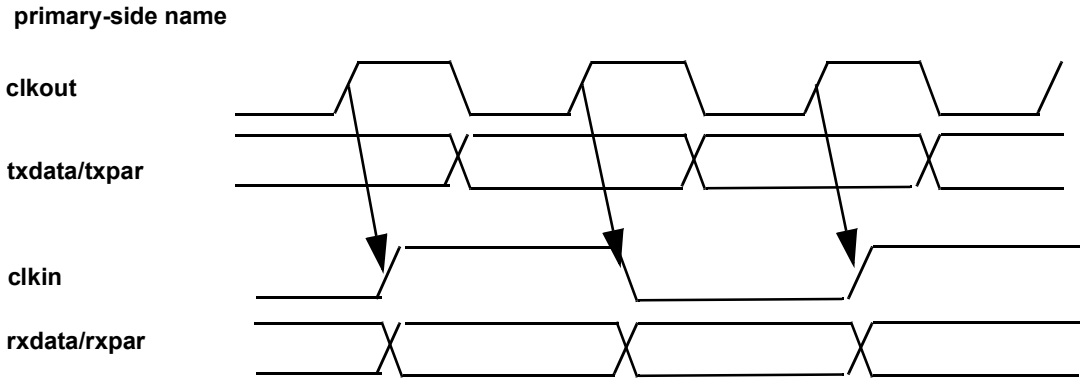


Figure 320. Clocking scheme for parallel interface between primary side and secondary side. Note that on the secondary side the in/out and tx/rx signal names are reversed.

126.3 Operation

126.3.1 AHB slave access

The socbridge provides an AHB slave interface for accesses to be done across the bridge. Two separate BARs can be provided, accesses to both BARs will behave exactly the same except for the address translation (see section 126.3.6).

For reads, the socbridge will send a read command to get the data, and give wait states on the AHB interface until the response has been received. For single accesses, the bridge will send a read command with the same size as the AHB access, unless the AHB access size exceeds the maximum size supported by the other endpoint. In this case, multiple read accesses of the highest size supported will be performed to get the required number of bytes.

In case of burst read access, the socbridge will calculate the optimal size of the read command to be sent across to the other endpoint considering the AHB access size, the number of beats of the burst, and the maximum size supported by both endpoints. If an incremental burst of unspecified length is performed, the core will assume a burst length given by the ahbiblen generic.

For write access, the core uses a write-behind scheme where it will first capture the data into an internal buffer and once the access (single or burst) is received, the write command and data will be transmitted. Any following AHB access to the slave while the write transfer is ongoing will be delayed using wait states.

Certain types of AHB bursts can not be optimally handled by the socbridge but will instead be handled as single accesses. This includes wrapping bursts (both read and write) and byte-sized bursts (with hsize equal to zero). They will still be executed in a functionally correct way but may not be performed with the optimal sequence of commands.

Busy cycles are handled in accordance with the AHB standard but may in some cases during read bursts lead to overrun conditions, where the remaining read data coming in is discarded and another read command is made to fetch the following data again. Busy cycles during write bursts are always handled without any performance overhead.

126.3.2 AHB master access

Incoming read and write commands from the other end of the byte stream will be performed by the AHB master, with a response and read data sent back over the link once the access is done. If the access is larger than the bus width, the AHB master will perform a burst to get all data at once.

126.3.3 Wide buses

The core supports wide AHB buses on both the master and slave ports, the widths are configured using a configuration generic. The core will correctly replicate the data on the data buses for all access sizes up to the configured maximum and pick the expected byte lanes in accordance with the AHB specification, which makes it compatible both with GRLIB's wide bus scheme as well as non-GRLIB slaves using the standard data muxing.

The core will always try to use the largest access possible on the AHB master interface, there is no support to constrain the size below the maximum. In systems with mixed wide and narrow slaves reachable by the bridge's AHB master, it is up to the other side of the link to always access the narrow slaves over the bridge using individual 32-bit accesses.

126.3.4 Maximum command size

In order to support small implementations of the bridge endpoint, the commmand sizes supported can be constrained.

The core has one buffer holding the data for bursts to and from the AHB slave, and one buffer holding the data for bursts to and from the AHB master. The smallest of the local AHB slave burst buffer and

the size of the other end's AHB master burst buffer determines the largest data transfer that can be sent over the link to the other end. The other end's burst buffer size is communicated to the bridge using the OMAXSZ field in the transfer control register. The local buffer sizes are configurable via generics. The local buffer sizes must always be large enough to hold a single access with the bus width configured.

Larger accesses than the maximum command size are still supported over the AHB slave interface but will be broken down into multiple commands.

126.3.5 Command queueing

On the command receiving side, the bridge has a queue to hold any additional transfer requests that come in while a transfer is in progress. This may be used on the command transmitting side to queue up additional commands before previous commands have been handled. A configuration register setting ORXBUFEN and a field ORXBUFSIZE is used to inform the bridge if a command queue is available in the other end and the capacity of the command queue space.

The command queueing feature is used to enqueue additional writes before write acknowledge has been received for previous writes. The improves bandwidth for cases where writes have a long latency on the other end before the acknowledge is sent back. Note that due to the single buffer in the bridge, the AHB write burst and the transmission of the commmand can still not be performed in parallel.

126.3.6 Address translation

By default the address transmitted over the byte stream is the same as the address of the AHB access but the core supports a basic address translation scheme. When enabled, the highest 8 bits of the outgoing address will be replaced by a value configured in a configuration register. Translation can be enabled individually for the two different slave BARs provided.

Translation of incoming accesses can also be enabled. This will in a similar way replace the highest 8 bits of any incoming address by a value from a configuration register.

126.3.7 Deadlock avoidance

When the bridge is used bidirectionally there is a potential deadlock condition when the AHB slaves on both sides of the link are accessed simultaneously. Since the slaves will be issuing wait states and keeping their respective AHB buses occupied, both the masters will wait indefinitely to be granted the bus they are connected to.

In order to recover from this condition, a deadlock detection scheme is implemented inside the core. If the master port is waiting to be granted and at the same time the slave port is waiting for a response from the other side of the link, a counter starts running that increments for every received IDLE byte on the byte stream. After 16 IDLE:s have been received in this condition, the deadlock detection triggers the AHB slave to issue a RETRY response. This frees up the bus for other masters, including the AHB master port of the bridge, to make accesses before the slave port access has completed. The slave port will issue RETRY responses to any other master that tries to access it, and return to operation once the master that originally accessed it comes back with the same access.

In order for the deadlock avoidance function to work as intended, round-robin arbitration policy needs to be used on at least one side of the bridge. This is needed in order to guarantee that the bridge's master port will be granted while the slave port is giving RETRY responses, and that the original master that was accessing the slave will eventually be granted access to the bus.

126.3.8 AHB locking considerations

The core ignores the lock related signals and does not communicate the lock state of the bus across the bridge, therefore locked accesses to the bridge will only be atomic with respect to other accesses on the bus where the access is made.

In bidirectional bridge configuration, a simultaneous locked access to the AHB slave port on both sides of the bridge may result in a deadlock condition which the deadlock avoidance logic can not resolve. This is because the locked state of the bus will take precedence over the arbitration policy and prevent any other master from accessing the bus even if the slave port issues RETRY responses.

126.3.9 AHB error handling

If the master port receives an AHB error response, any read data is replaced by all-one or all-zero data (based on the ERRVAL setting). That the error occurred is otherwise not reported to the other end-point. If detecting ERROR response from the other end is desired, a system designer could for example provide an AHB status register at a given address that could be read out through the bridge.

The bridge can also be configured to go into error state when an AHB error occurs. If this option is used, the error would have to be acknowledged before any further access can be made.

126.3.10Soft reset

The core can be configured to include a synchronization mechanism, called *soft reset*. Using this mechanism it is possible for one core to request a connected core to go into a reset state. This requires that a *softrst* signal is connected in each direction between the two cores. The actual reset request happens with a four-way handshake using these two signals.

Using the *softrsttype* VHDL generic it is possible to configure the core if it should ignore the *softrst* signal, should be able to generate a soft reset, or if it should be able to receive a soft reset. To be able to use the soft reset functionality, for two connected cores, one needs to be configured as a *requesting core* (*softrsttype=2)* and another as a *receiving core* (*softrsttype=1).*

The soft reset handshake between two socbridge cores follows these four coordinated steps:

Step 1) Initiation (Request Asserted)

The process begins when, for a core configured as a *soft reset requester*, either happens:

- The core is reset by the *rstn* signal
- A value of '1' is written to the *srst* register bit.

This causes the core to asserts its *softrst* signal, signaling the start of the soft reset process.

Step 2) Acknowledge (Receiver Asserted)

Upon receiving the asserted *softrst* signal, the *receiving core* must meet the following conditions before it can acknowledge the request:

1. Any ongoing AHB access to its slave interface must be completed with an AHB error response. For bursts, the entire burst must be served.
2. Any ongoing access on its master interface must be completed.
3. A minimum of 32 cycles must have passed since the request was received.

Once all three conditions are satisfied, the *receiving core* asserts its own *softrst* signal to acknowledge the reset request.

Step 3) Completion (Request Deasserted)

When the *requesting core* sees that the *softrst* signal it receives is asserted (i.e., acknowledged by the receiver), and at least 32 cycles have passed since it sent the request, it deasserts its own *softrst* signal.

Step 4: Final Acknowledge (Receiver Deasserted)

When the *receiving core* detects that the *softrst* signal it receives is deasserted, it considers the soft reset process complete and deasserts its own *softrst* signal. At this point, both cores resume normal operation and allow processing of new accesses.

Any new AHB access to the slave interface of either core which happens after the soft reset process has started will be held with waitstates until the handshake has completed. Additionally, all traffic over the byte stream is ignored during the process of a soft reset.

When a soft reset is detected by a core, it will reset a subset of its internal registers immediately. Included in this subset is the error status register, meaning it is possible to get back to a synchronized state if a error happens in either socbridge core by starting the soft reset process in the *requesting core.*

Currently the *soft reset* functionality is only supported for the cross-connection interconnects, which are described in chapter 126.4.4.

126.3.11Link error handling

The core can check for parity errors, if parity is implemented, and also detect certain protocol violations on the incoming data. If any of these errors occur then the bridge will go into “error state”. In error state, any access to the slave port will be given ERROR response and commands coming in from the other endpoint will be ignored.

When the bridge is in error state, the bridge will continuously send the PeerError command to the other end to indicate that it is in error state. Receiving the PeerError command will make the other end also go into error state. To avoid loops of peer error commands, the endpoint that has received a PeerError will only send regular Idle commands.

The error state can be cleared via the register interface. Once the error condition is cleared, the bridge returns to an idle state. The endpoint that had the original error needs to be cleared first in order to stop the transmission of PeerError messages. If the other endpoint is cleared first, it will immediately see another PeerError and go back into error state.

126.3.12Link startup

After reset has been released, the core will ignore the first 16 data bytes received over the link and the core will transmit at least 32 IDLE bytes before the first command. This is intended to simplify the design by providing some margin in case reset is released at slightly different times on the two sides of the bridge, and removes the need for asynchronous resets in registers that are in the data path.

126.3.13Interrupts

The core can be configured provide an interrupt when it enters error state.

126.3.14Register interface

Configuration registers are accessed through an APB slave interface.

In order not to require a second APB interface for the interface layer, some register fields to configure the interface are also included in the socbridge. Depending on static signals in from the interface layer, these may get forced to a constant value in order to optimized out in designs where they are not needed.

126.4 Implementation

126.4.1 Physical interfaces

The main socbridge block runs entirely in the AHB bus clock domain. It expects external logic to provide the incoming data bytes from the byte stream and valid signals to indicate on which cycles new data is available and on which cycles a transmit data byte has been consumed.

126.4.2 Oversampling parallel interface

The socbridge_parif block provides an implementation of the parallel interface as described in section 126.2.4. The block is clocked by the same AHB clock used for the socbridge and can handle external data rates up to 1/8 of the AHB clock rate. The incoming clock is captured using synchronization registers for edge detection.

The interface can be configured at run time to act as either primary or secondary side of the interface. When configured as primary side, it uses a counter to divide the AHB clock down to create the outgoing clock. The timing of when the transmit data changes in relation to the outgoing clock edge can be configured in order to allow timing optimizations. For the incoming clock, a counter is used to time when the data is sampled relative to detected edges on the clock.

When the interface is configured to run as secondary side, it will detect rising edges on the incoming clock and use there to sample data, deliver outgoing data and toggle the outgoing clock. When configured as secondary side there are no counters or configurable delays. If the physical interface is built with only support for secondary side operation, most registers needed for primary side operation can be removed.

126.4.3 Link-clocked parallel interface

An alternative interface block called socbridge_parif_comclk is provided for simple companion chip implementations where the clock coming in on the socbridge link is also used to clock the entire system-on-chip. This block only support secondary side operation.

Having the same clock removes the need for synchronization registers and the interface block reduces down to just a register stage.

126.4.4 Cross-connection

For prototyping and verification purposes, crossbars to directly interconnect two socbridges in the same system are provided. socbridge_xcon simply interconnect two socbridges inputs and outputs with no additional logic and works when the two bridges are in the same clock domain. For the case where the bridges are in different clock domains socbridge_xcon_cdc is provide that uses asynchronous FIFOs to transfer the data between the domains.

126.4.5 Internal buffers

The socbridge contains the following data buffers, all implemented in registers:

- master-side burst buffer, number of bytes given by mmaxburst
- master-side request queue, number of bytes given by rxqsize
- slave-side burst buffer, number of bytes given by smaxburst
- common request/response transmit buffer, fixed size of 5 bytes.

126.4.6 Endianness

The core support operation on both little-endian and big-endian systems and will adjust depending on the endian side-band signal in the AMBA records

Regardless of endian-ness, the read and write data over the link is transferred byte-wise in incrementing address order (the byte at the initial address A transmitted first, followed by the byte at address A+1, the byte at address A+2, etc.). When using the bridge with opposing endian-ness on both sides, the bridge will therefore provide byte consistent access to data.

126.4.7 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_async_reset_enable* is set.

126.5 Registers

The bridge provides a configuration register area accessed over the APB bus. For forward compatibility, reserved fields should be written either with 0 or with the last read-out value, and reserved registers should not be accessed at all.

*Table 2312.*Configuration registers for SOCBRIDGE

Register address offset	Name	R/W	Reset value	Notes
0x00	Transfer configuration register	R/W	-	1
0x04	AHB configuration register	R/W	0x00000000	
0x08	Status register	R/W	0x00000000	
0x0C	Interrupt configuration register	R/W	0x00000001	
0x10	Parallel interface configuration register 1	R/W	-	1
0x14	Parallel interface configuration register 2	R/W	-	1
0x18 - 0xFC	RESERVED			

Note 1: Reset value depends on IP configuration options or external interface configuration

126.5.1 Transfer configuration register

Table 2313. 0x00 - SOCBRIDGE transfer configuration register

31	18	17	16
RESERVED		MDIS	PCHK
0		0	*
r		rw	rw

15	14	3	2	0
ORQEN	ORQSIZE		OMAXSZ	
0	111111111111		-	
rw	rw		rw	

31 : 18 Reserved

17 Master disable (MDIS) - when set to 1 any incoming commands from the other end will be ignored.

16 Parity check enable (PCHK) - when set to 1 the bridge checks parity and enters error state when a parity is detected, when set to 0 parity errors are ignored.

If parity is not implemented on the interface layer, this field will be constant zero.

15 Other end request queue enable (ORBEN) - set to 1 when it is known that the other end has a request queue that can take in additional commands while handling a command. When set to 0 the bridge will always wait for a response before sending any additional command.

14 : 3 Other end request queue size (ORQSIZE) - the size of the other end's request queue. Used only when ORQEN is set to 1.

2 : 0 Other end maximum access size. This is set to the largest size command that the other end can handle (corresponding to the mmaxburst generic setting on the other side). Size in power of two, default is set by the *omaxsz* VHDL generic.

126.5.2 AHB configuration register

Table 2314. 0x04 - SOCBRIDGE AHB configuration register

31	30	29	28	27	26	25	24	23	16
RESERVED		MEVAL	MEFAT	SIGWR	S2MOD	S1MOD	MMOD	S2MODADDR	
0		1	0	0	0	0	0	00000000	
r		rw	rw	rw	rw	rw	rw	rw	

15	8	7	0
S1MODADDR		MMODADDR	
00000000		00000000	
rw		rw	

31 : 30	Reserved
29	AHB master error value (MEVAL) - value to substitute in read data in case the AHB master access gets an ERROR response. If set to 1 an all-1 pattern is used, if set to 0 an all-0 pattern is used.
28	AHB master error fatal (MEFAT) - if set to 1, an ERROR response to an access by the AHB master will be treated as a fatal error and the link is put into error state.
27	Slave ignore writes (SIGWR) - when set to 1 the slave port will ignore writes and instantly give an OKAY response to the master.
26	Slave area 2 address modification enable (S2MOD) - when set to 1 accesses to the second slave BAR (if implemented) will have the most significant byte changed to the value of the S2MO-DADDR field. When set to 0, the AHB access address will be used directly as the command's address. This field is only writable if the second BAR of the slave port is implemented.
25	Slave area 1 address modification enable (S1MOD) - when set to 1 accesses to the first slave BAR (if implemented) will have the most significant byte changed to the value of the S1MODADDR field. When set to 0, the AHB access address will be used directly as the command's address. This field is only writable if the first BAR of the slave port is implemented
24	Master address modification enable (MMOD) - when set to 1, incoming accesses will have the most significant byte of the address used by the AHB master changed to the value of the MMODADDR field. When set to 0, the incoming address will be used directly as the AHB master address. This field is only writable if the master port is implemented
23 : 16	Slave area 2 address modification address value (S2MODADDR) This field is only writable if the second BAR of the slave port is implemented.
15: 8	Slave area 1 address modification address value (S1MODADDR) This field is only writable if the first BAR of the slave port is implemented.
7 : 0	Master address modification address value (MMODADDR) This field is only writable if the master port is implemented

126.5.3 Status register

Table 2315. 0x08 - SOCBRIDGE status register

31			28	27									19	18	17	16
IFID				RESERVED										SRTYPE		SRST
*				0										-		0
r				r										r		r / ws
15	14	13	12	11	10	9	8	7	6	5	4	3		2	1	0
RXINIT	TXINIT	RXQNE	OCP	MBUSY	SBUSY	RXCMD	TXIP	ERRS	MCERR	ASERR	AMERR	RINC		RXOVR	PARE	PEERE
1	1	0	0	0	0	0	0	0	0	9	0	0		0	0	0
r	r	r	r	r	r	r	r	wc	wc	wc	wc	wc		wc	wc	wc

31 : 28 Interface ID - read only field to identify which type of physical interface is attached to the bridge.

0: oversampling parallel interface

1: stream-clocked parallel interface

2: cross-connection

3: cross-connection with CDC

4-14: Reserved

15: Reserved for custom implementation

27 : 19 Reserved

18:17 SRTYPE - Mirrors the VHDL generic *softrsttype*

0: Does not implement soft reset functionality

1: The core can receive and respond to a soft reset.

2: The core can initiate a soft reset.

3: Reserved

16 SRST - set to 1 while a soft reset is in progress. If the core is configured to be able to initiate soft resets (VHDL generic *softrsttype=2),* then writing a 1 to the register bit will start the soft reset process, otherwise the register bit is read only.

15 RXINIT - set to 1 after reset during receive hold-off period

14 TXINIT - set to 1 after reset during transmission hold-off period.

13 Receive command queue non-empty

12 Other side command in progress (OCP) - Set to 1 if we are waiting for one or more responses from the other side.

11 Master busy (MBUSY) - indicate that the AHB master/incoming commmand state machine is in a busy state.

10 Slave busy (SBUSY) - indicate that the AHB slave/outgoing command state machine is in a busy state.

9 Receiving commmand (RXCMD) - set to 1 while in the middle of receiving a command or reply from the other side of the link

8 TXIP - Set to 1 when transmission in progress and not transmitting idle

7 Error state (ERRS) - set to 1 when the socbridge has entered error state. Write 1 to clear error state status and return to idle state

6 Malformed command error (MCERR) - set to 1 when a command has been received which does not comply with the socbridge protocol in some way. This will also set the error state indication and put the bridge in error state.

The following errors can trigger the MCERR bit:

* The address of a command is not aligned to the specified access size.

* The command access size is larger than what the *mmaxburst* VHDL generic specifies.

* The command byte does not match with any known command as specified in Ch. 126.2.2.

5 AHB slave error (ASERR) - set to 1 when the slave receives an AHB access request where the address is unaligned with regards the to given access size or if the access size is larger than what is allowed by the *sbusw* VHDL generic. This will also set the error state indication and put the bridge in error state.

Table 2315. 0x08 - SOCBRIDGE status register

4	AHB master error (AMERR) - set to 1 when the master receives an AHB error if the master error fatal register bit has been set to one. This will also set the error state indication and put the bridge in error state.
3	Response inconsistency (RINC) - set to 1 when the other side buffer accounting underflows, indicating that the command replies received do not match the sizes of the commands given. Will also make the bridge go into error state
2	Receive queue overflow (RXOVR) - set to 1 when the bridge receives more commands than it has space for in the receive queue. Will also make the bridge go into error state.
1	Parity error (PARE) - set to 1 when the bridge detects a parity error.
0	Peer error indication (PERRE) - set to 1 when the bridge has received a peer error indication from the other side. The peer error will also set the error state indication.

126.5.4 Interrupt configuration register

Table 2316. 0x0C - SOCBRIDGE interrupt configuration register

31	16
RESERVED	
0	
r	

15	8	7	6	0
RESERVED		ESIRQ	RESERVED	
0		0	0	
r		rw	r	

31 : 8 Reserved

7 Error state interrupt enable (ESIRQ) - set to 1 to enable delivery of interrupts when entering error state.

6 : 0 Reserved

126.5.5 Parallel interface configuration register 1

Table 2317. 0x10 - SOCBRIDGE parallel interface configuration register 1

31	16
RESERVED	
0	
r	

15	4	3	1	0
CLKODIVF		RESERVED		PRIM
111111111111		0		*
rw		r		rw

31 : 16 Reserved

15 : 4 Clock output division factor.

3 : 1 Reserved

0 Primary mode (PRIM) - set to 1 when configured as primary

Note: The fields in this register are only writable if a parallel interface is attached

126.5.6 Parallel interface configuration register 2

Table 2318. 0x14 - SOCBRIDGE parallel interface configuration register 2

31	24	23	16
RESERVED		RXPHASE(11:4)	
0		10000000	
r		rw	

15	12	11	0
RXPHASE(3:0		TXPHASE	
0000		100000000000	
rw		rw	

31 : 24 Reserved

23 : 12 (RXPHASE) -

11 : 0 (TXPHASE) -

Note: The fields in this register are only writable if a parallel interface is attached.

126.6 Vendor and device identifiers

This bridge is identified with vendor ID 0x001 (Frontgrade Gaisler) and device ID 0x0C4. The ver-

sion described in this document is version 1.

126.7 Configuration options

Tables 2319 shows the configuration options (VHDL generics) of the core.

*Table 2319.*Configuration options for SOCBRIDGE

Generic	Function	Allowed range	Default
hmindex	AHB master index for master port	0 - NAHBMST-1	0
hsindex	AHB slave index for slave port	0 - NAHBSLV-1	0
hsaddr	ADDR field of AHB BAR0 for the slave port	0 - 16#FFF#	0
hsmask	MASK field of AHB BAR0 for the slave port	0 - 16#FFF#	0
hsaddr2	ADDR field of AHB BAR1 for the slave port	0 - 16#FFF#	0
hsmask2	MASK field of AHB BAR1 for the slave port	0 - 16#FFF#	0
pindex	APB index for APB slave port	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR for the register port.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR for the register port.	0 - 16#FFF#	0
pirq	Interrupt number for the PIRQ bus	0 - 16#FFF#	0
mbusw	Maximum AHB access size (bus width) for the master port	32, 64, 128, 256	0
sbusw	Maximum AHB access size (bus width) for the slave port	32, 64, 128, 256	0
rxqsize	Size of receive command queue, in bytes.	5 - 65535	5
mmaxburst	Size of master burst buffer and maximum receive command size, in bytes. Must be power of 2, at least 4 bytes.	4, 8, 16, 32, ..., 128	16#FFF#
smaxburst	Size of slave burst buffer and maximum transmit command size, in bytes. Must be power of 2, at least 4 bytes.	4, 8, 16, 32, ...,, 128	16#000#
ahbiblen	Assumed burst length for unspecified incremental burst, in beats. Must be power of 2	1, 2, 4, 8, 16, 32	4
pifdefmode	Default parallel interface mode (1=primary, 0=secondary). Used only with parallel interface.	0 - 1	0
mstdis	Set to 1 to disable the master port	0 - 1	0
omaxsz	Determines the reset value of the *omaxsz* register field.	0 - 7	2
mode_32b	*32b mode* enabled. Set the core to use a 32-bit wide interface instead of the default 8-bit.	0, 1	0
softrsttype	Determines what *soft reset* functionality is supported the by the core. 0: Does not implement soft reset functionality 1: The core can receive and respond to a soft reset. 2: The core can initiate a soft reset. Only supported when using the cross-connection interconnect.	0, 1, 2	0

126.8 Signal descriptions

Table 2320 shows the interface signals (VHDL ports) of the of the core.

*Table 2320.*Signal descriptions for SOCBRIDGE

Signal name	Field	Type	Function	Active
CLK	N/A	Input	AHB clock	Rising
RSTN	N/A	Input	Reset input for AHB clock domain	Low
SBRI	TXEN	Input	Signal socbridge to update transmit bus on next clock edge	High
	RXEN		Signal socbridge that receive data bus contains valid data to be captured on next clock edge	High
	RXDATA[31:0]		Input data bus. Bits [31:8] is only used when *32b mode* is enabled.	-
	RXPAR		Input parity for RXDATA	-
	SOFTRST		Input signal used for the *soft reset* handshake	High
	RXPAREN		Static signal from interface layer to indicate that parity is available.	High
	CFGPRIMEN		Static signal from interface layer to indicate that the PRIM configuration field should be implemented in the core and output on the SBRO record.	High
	CFGDIVEN		Static signal from interface layer to indicate that the configuration fields for the CLKODIVF, RXPHASE and TXPHASE should be implemented in the core and output on the SBRO record.	High
	IFID[3:0]		Static signal from interface layer to indicate what interface implementation is used, the value provided in this field is readable through the status register.	-
SBRO	TXDATA[31:0]	Output	Output data bus. Bits [31:8] is only used when *32b mode* is enabled.	-
	TXPAR		Parity corresponding to TXDATA bus	-
	SOFTRST		Output signal used for the *soft reset* handshake	High
	CFGPRIM		Copy of PRIM configuration register field, if implemented	-
	CFGCLKODIVF		Current value of CLKODIVF configuration register field, if implemented	-
	CFGTXPHASE[11:0]		Copy of TXPHASE configuration register field, if implemented	-
	CFGRXPHASE[11:0]		Copy of RXPHASE configuration register field, if implemented	-
MSTI	*	Input	Inputs for AHB master port	-
MSTO	*	Output	Outputs for AHB master port	-
SLVI	*	Input	Inputs for AHB slave port	-
SLVO	*	Output	Outputs for AHB slave port	-
APBI	*	Input	Inputs for APB port	-
APBO	*	Output	Outputs for APB port	-

* see GRLIB IP Library User's Manual

126.9 Library dependencies

Tables 2321 shows libraries used when instantiating the core (VHDL libraries).

*Table 2321.*Library dependencies for instantiating ftaddr_gr top level

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	Signal record definitions
GAISLER	SOCBRIDGE_PKG	Component, Types, Constants	Component declaration for socbridge and interfaces
TECHMAP	GENCOMP	Constants	Constants for tech generic

126.10 Component declaration

The component declaration for the core is provided below.

```
component socbridge is
    generic (
      hmindex    : integer := 0;
      hsindex    : integer := 0;
      hsaddr     : integer := 16#000#;
      hsmask     : integer := 16#000#;
      hsaddr2    : integer := 16#000#;
      hsmask2    : integer := 16#000#;
      pindex     : integer := 0;
      paddr      : integer := 16#000#;
      pmask      : integer := 16#fff#;
      pirq       : integer := 0;
      mbusw      : integer := AHBDW;
      sbusw      : integer := AHBDW;
      rxqsize    : integer := 5;
      mmaxburst  : integer := 32;
      smaxburst  : integer := 32;
      ahbiblen   : integer := 32;
      pifdefmode : integer := 0;
      mstdis     : integer := 0;
      omaxsz     : integer := 2;
      mode_32b   : integer := 0;
      softrsttype: integer := 0
      );
    port (
      clk         : in  std_ulogic;
      rstn        : in  std_ulogic;
      sbri        : in  socbridge_in_type;
      sbro        : out socbridge_out_type;
      msti        : in  ahb_mst_in_type;
      msto        : out ahb_mst_out_type;
      slvi        : in  ahb_slv_in_type;
      slvo        : out ahb_slv_out_type;
      apbi        : in  apb_slv_in_type;
      apbo        : out apb_slv_out_type
      );
  end component;
```

127 SPI2AHB - SPI to AHB bridge

127.1 Overview

The SPI to AHB bridge is an SPI slave that provides a link between a SPI bus (that consists of two data signals, one clock signal and one select signal) and AMBA AHB. On the SPI bus the slave acts as an SPI memory device where accesses to the slave are translated to AMBA accesses. The core can translate SPI accesses to AMBA byte, half-word or word accesses. The access size to use is configurable via the SPI bus.

The core synchronizes the incoming clock and can operate in systems where other SPI devices are driven by asynchronous clocks.

GRLIB also contains a SPI master/slave controller core, without an AHB interface, where the transfer of each individual byte is controlled by software via an APB interface, see the SPICTRL core documentation for more information.

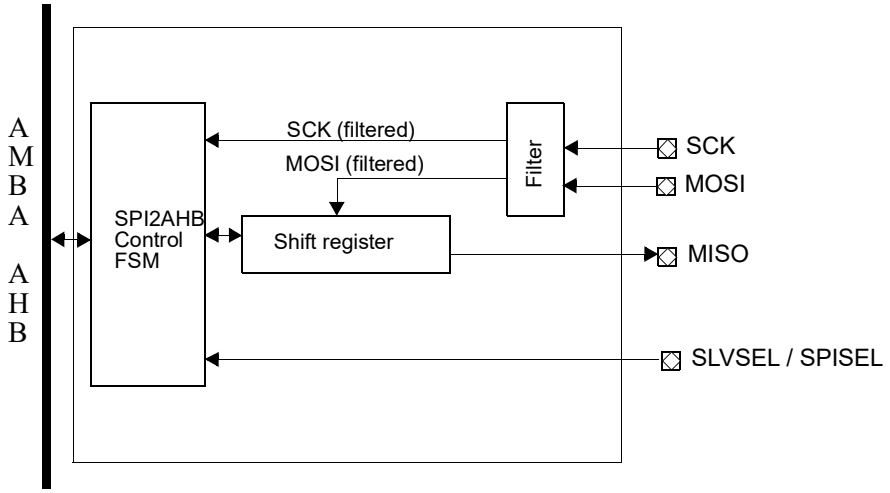


Figure 321. Block diagram, optional APB interface not shown

127.2 Transmission protocol

The SPI bus is a full-duplex synchronous serial bus. Transmission starts when a master selects a slave through the slave's Slave Select (SLVSEL) signal and the clock line SCK transitions from its idle state. Data is transferred from the master through the Master-Output-Slave-Input (MOSI) signal and from the slave through the Master-Input-Slave-Output (MISO) signal. In some systems with only one master and one slave, the Slave Select input of the slave may be always active and the master does not need to have a slave select output. This does not apply to this SPI to AHB bridge, the slave select signal must be used to mark the start and end of an operation.

During a transmission on the SPI bus data is either changed or read at a transition of SCK. If data has been read at edge n, data is changed at edge n+1. If data is read at the first transition of SCK the bus is said to have clock phase 0, and if data is changed at the first transition of SCK the bus has clock phase 1. The idle state of SCK may be either high or low. If the idle state of SCK is low, the bus has clock polarity 0 and if the idle state is high the clock polarity is 1. The combined values of clock polarity (CPOL) and clock phase (CPHA) determine the mode of the SPI bus. Figure 322 shows one byte (0x55) being transferred MSb first over the SPI bus under the four different modes. Note that the idle state of the MOSI line is '1' and that CPHA = 0 means that the devices must have data ready before the first transition of SCK. The figure does not include the MISO signal, the behavior of this line is the same as for the MOSI signal. However, due to synchronization the MISO signal will be delayed for a period of time that depends on the system clock frequency.

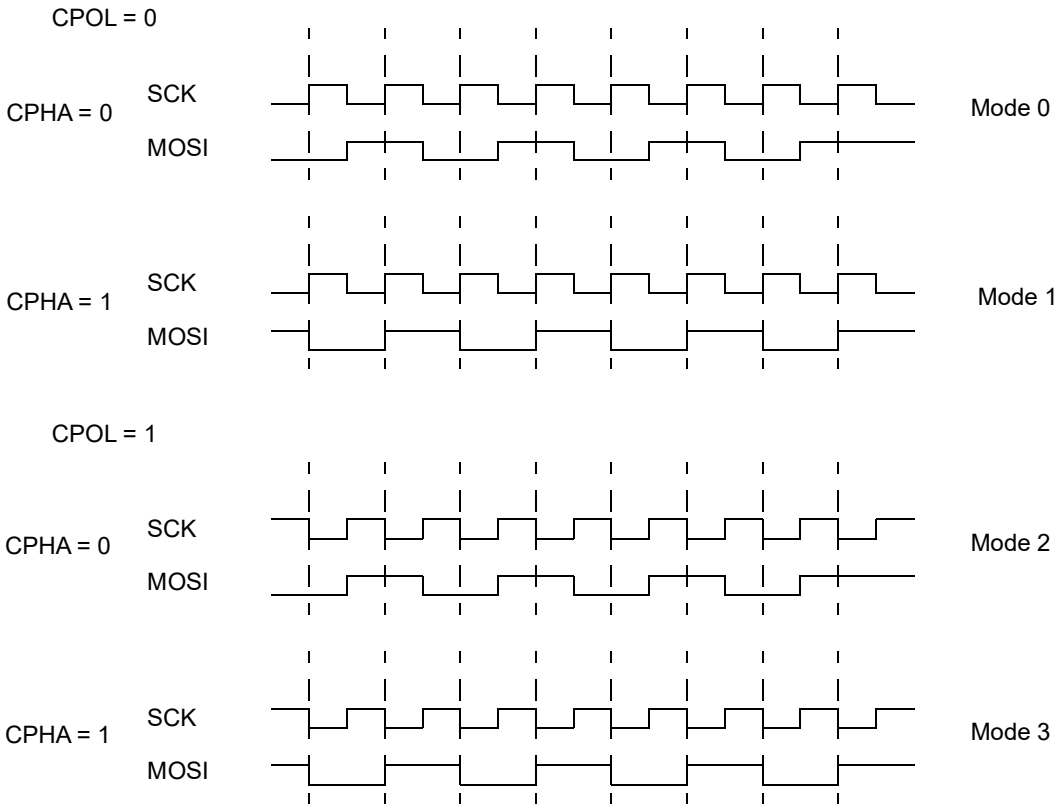


Figure 322. SPI transfer of byte 0x55 in all modes

The SPI to AHB bridge makes use of a protocol commonly used by SPI Flash memory devices. A master first selects the slave via the slave select signal and then issues a one-byte instruction. The instruction is then followed by additional bytes that contain address or data values. All instructions, addresses and data are transmitted with the most significant bit first. All AMBA accesses are done in big endian format. The first byte sent to or from the slave is the most significant byte.

127.3 System clock requirements and sampling

The core samples the incoming SPI SCK clock and does not introduce any additional clock domains into the system. Both the SCK and MOSI lines first pass through two stage synchronizers and are then filtered with a low pass filter.

The synchronizers and filters constrain the minimum system frequency. The core requires the SCK signal to be stable for at least two system clock cycles before the core accepts the SCK value as the new clock value. The core's reaction to transitions will be additionally delayed since both lines are taken through two-stage synchronizers before they are filtered. In order for the slave to be able to output data on the SCK 'change' transition and for this data to reach the master before the next edge the SCK frequency should not be higher than one tenth of the system frequency of core (with the standard VHDL generic *filter* setting of 2).

The slave select input should be asserted at least two system clock cycles before the SCK line starts transitioning.

127.4 SPI instructions

127.4.1 Overview

The core is controlled from the SPI bus by sending SPI instructions. Some commands require additional bytes in the form of address or data. The core makes use of the same instructions as commonly available SPI Flash devices. Table 2322 summarizes the available instructions.

*Table 2322.*SPI instructions

Instruction	Description	Instruction code	Additional bytes
RDSR	Read status/control register	0x05	Core responds with register value
WRSR	Write status/control register	0x01	New register value
READ	AHB read access	0x03	Four address bytes, after which core responds with data.
READD	AHB read access with dummy byte	0x0B	Four address bytes and one dummy byte, after which core responds with data
WRITE	AHB write access	0x02	Four address bytes followed by data to be written

All instructions, addresses and data are transmitted with the most significant bit first. All AMBA accesses are done in big endian format. The first byte sent to or from the slave is the most significant byte.

127.4.2 SPI status/control register accesses (RDSR/WRSR)

The RDSR and WRSR instructions access the core's SPI status/control register. The register is accessed by issuing the wanted instruction followed by the data byte to be written (WRSR) or any value on the byte in order to shift out the current value of the status/control register (RDSR). The fields available in the SPI status/control register are shown in table 2323.

*Table 2323.*SPI2AHB SPI status/control register

7	6	5	4	3	2	1	0
Reserved	RAHEAD	PROT	MEXC	DMAACT	MALF	HSIZE	

7	Reserved, always zero (read only)
6	Read ahead (RAHEAD) - When this bit is set the core will make a new access to fetch data as soon as the last current data bit has been moved. Otherwise the core will not attempt the new access until the 'change' transition on SCK. Setting this bit to '1' allows higher SCK frequencies to be used but will also result in a data fetch as soon as the current data has been read out. This means that RAHEAD may not be suitable when accessing FIFO interfaces. (read/write)
5	Memory protection triggered (PROT) - '1' if last AHB access was outside the allowed memory area. Updated after each AMBA access (read only). Note that since this bit is updated after each access the RAHEAD = '1' setting may hide errors.
4	Memory exception (MEXC) - '1' if core receives AMBA ERROR response. Updated after each AMBA access (read only). Note that since this bit is updated after each access the RAHEAD = '1' setting may hide errors.
3	DMA active (DMAACT) - '1' if core is currently performing a DMA operation.
2	Malfunction (MALF): This bit is set to one by the core is DMA is not finished when a new byte starts getting shifted. If this bit is set to '1' then the last AHB access was not successful.
1:0	AMBA access size (HSIZE) - Controls the access size that the core will use for AMBA accesses. 0: byte, 1: half-word, 2: word. HSIZE = "11" is illegal.

Reset value: 0x42

127.4.3 Read and write instructions (WRITE and READ/READD)

The READD is the same as the READ instruction with an additional dummy byte inserted after the four address bytes. To perform a read operation on AHB via the SPI bus the following sequence should be performed:

1. Assert slave select
2. Send READ instruction
3. Send four byte AMBA address, the most significant byte is transferred first

3a. Send dummy byte (if READD is used)

4. Read the wanted number of data bytes
5. De-assert slave select

To perform a write access on AHB via the SPI bus, use the following sequence:

1. Assert slave select
2. Send WRITE instruction
3. Send four byte AMBA address, the most significant byte is transferred first
4. Send the wanted number of data bytes
5. De-assert slave select

During consecutive read or write operations, the core will automatically increment the address. The access size (byte, halfword or word) used on AHB is set via the HSIZE field in the SPI status/control register.

The core always respects the access size specified via the HSIZE field. If a write operation writes fewer bytes than what is required to do an access of the specified HSIZE then the write data will be dropped, no access will be made on AHB. If a read operation reads fewer bytes than what is specified by HSIZE then the remaining read data will be dropped when slave select is de-asserted.

The core will not mask any address bits. Therefore it is important that the SPI master respects AMBA rules when performing half-word and word accesses. A half-word access must be aligned on a two byte address boundary (least significant bit of address must be zero) and a word access must be aligned on a four byte boundary (two least significant address bits must be zero).

The core can be configured to generate interrupt requests when an AHB access is performed if the core is implemented with the APB register interface, see the APB register documentation for details.

127.4.4 Memory protection

The core is configured at implementation time to only allow accesses to a specified AHB address range (which can be the full 4 GiB AMBA address range). If the core has been implemented with the optional APB register interface then the address range is soft configurable and the reset value is specified with VHDL generics.

The VHDL generics *ahbaddrh* and *ahbaddrl* define the base address for the allowed area. The VHDL generics *ahbmaskh* and *ahbmaskl* define the size of the area. The generics are used to assign the memory protection area's address and mask in the following way:

Protection address, bits 31:16 (*protaddr[31:16]*): ahbaddrh
Protection address, bits 15:0 (*protaddr[15:0]*): ahbaddrl
Protection mask, bits 31:16 (*protmask[31:16]*): ahbmaskh
Protection mask, bits 15:0 (*protmask[15:0]*): ahbmaskl

Before the core performs an AMBA access it will perform the check:

(((incoming address) xor (protaddr)) and protmask) /= 0x00000000

If the above expression is true (one or several bits in the incoming address differ from the protection address, and the corresponding mask bits are set to '1') then the access is inhibited. As an example, assume that *protaddr* is 0xA0000000 and *protmask* is 0xF0000000. Since *protmask* only has ones in the most significant nibble, the check above can only be triggered for these bits. The address range of allowed accessed will thus be 0xA0000000 - 0xAFFFFFFF..

The core will set the configuration register bit PROT if an access is attempted outside the allowed address range. This bit is updated on each AHB access and will be cleared by an access inside the allowed range. Note that the (optional) APB status register has a PROT field with a slightly different behavior.

127.5 Registers

The core can optionally be implemented with an APB interface that provides registers mapped into APB address space.

Table 2324. APB registers

APB address offset	Register
0x00	Control register
0x04	Status register
0x08	Protection address register
0x0C	Protection mask register

127.5.1 Control Register

Table 2325. 0x00 - CTRL - Control register

31 2	1	0
RESERVED	IRQEN	EN
0	0	*
r	rw	rw

31 : 2 RESERVED

1 Interrupt enable (IRQEN) - When this bit is set to '1' the core will generate an interrupt each time the DMA field in the status register transitions from '0' to '1'.

0 Core enable (EN) - When this bit is set to '1' the core is enabled and will respond to SPI accesses. Otherwise the core will not react to SPI traffic.

127.5.2 Status Register

Table 2326. 0x04 - STAT - Status register

31 3	2	1	0
RESERVED	PROT	WR	DMA
0	0	0	0
r	wc	r	wc

31 : 3 RESERVED

2 Protection triggered (PROT) - Set to '1' if an access has triggered the memory protection. This bit will remain set until cleared by writing '1' to this position. Note that the other fields in this register will be updated on each AHB access while the PROT bit will remain at '1' once set.

1 Write access (WR) - Last AHB access performed was a write access. This bit is read only.

0 Direct Memory Access (DMA) - This bit gets set to '1' each time the core attempts to perform an AHB access. By setting the IRQEN field in the control register this condition can generate an interrupt. This bit can be cleared by software by writing '1' to this position.

127.5.3 Protection Address Register

Table 2327. 0x08 - PADDR - Protection address register

31 0
PROTADDR
*
rw

31 : 0 Protection address (PROTADDR) - Defines the base address for the memory area where the core is allowed to make accesses.

127.5.4 Protection Mask Register

Table 2328. 0x0C - PMASK - Protection mask register

31 0
PROTMASK
*
rw

31 : 0 Protection mask (PROTMASK) - Selects which bits in the Protection address register that are used to define the protected memory area.

Reset value: Implementation dependent

127.6 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x05C. For a description of vendor and device identifiers see the GRLIB IP Library User's Manual.

127.7 Implementation

127.7.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

127.7.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

127.8 Configuration options

Table 2329 shows the configuration options of the core (VHDL generics). Two different top level entities for the core is available. One with the optional APB interface (spi2ahb_apb) and one without the APB interface (spi2ahb). The entity without the APB interface has fewer generics as indicated in the table below.

*Table 2329.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST	0
ahbaddrh	Defines bits 31:16 of the address used for the memory protection area	0 - 16#FFFF#	0
ahbaddrl	Defines bits 15:0 of the address used for the memory protection area	0 - 16#FFFF#	0
ahbmaskh	Defines bits 31:16 of the mask used for the memory protection area	0 - 16#FFFF#	0
ahbmaskl	Defines bits 15:0 of the mask used for the memory protection area	0 - 16#FFFF#	0
resen	Reset value for core enable bit (only available on the spi2ahb_apb entity).	0 - 1	0
pindex	APB slave index (only available on the spi2ahb_apb entity).	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR (only available on the spi2ahb_apb entity).	0 - 16#FFF#	0
pmask	MASK field of the APB BAR (only available on the spi2ahb_apb entity).	0 - 16#FFF#	16#FFF#
pirq	Interrupt line driven by APB interface (only available on the spi2ahb_apb entity).	0 - NAHBIRQ-1	0
oepol	Output enable polarity	0 - 1	0
filter	Low-pass filter length. This generic should specify, in number of system clock cycles plus one, the time of the shortest pulse on the SCK clock line to be registered as a valid value.	2 - 512	2
cpol	Clock polarity of SPI clock (SCK)	0 - 1	0
cpha	Clock phase of SPI communication	0 - 1	0

127.9 Signal descriptions

Table 2330 shows the interface signals of the core (VHDL ports).

*Table 2330.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBI	*	Input	AHB master input signals	-
AHBO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
SPII	SCK	Input	SPI clock line input	-
	MOSI	Input	SPI data line input	-
	SPISEL	Input	SPI slave select input	
	Other fields	Input	Unused	
SPIO	MISO	Output	SPI data line output	-
	MISOOEN	Output	SPI data line output enable	Low**
	Other fields	Output	Unused	-

* see GRLIB IP Library User's Manual
** depends on value of OEPOL VHDL generic.

127.10 Signal definitions and reset values

The signals and their reset values are described in table 2331.

*Table 2331.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
sck	Input	SPI clock line	-	Hi-Z
miso	InputOutput	SPI master-input, slave-output line	-	Hi-Z
mosi	Input	SPI master-output, slave-input line	-	-
spisel	Input	SPI slave select	Low	-

127.11 Library dependencies

Table 2332 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2332.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	SPI	Component, signals	Component declaration, SPI signal definitions

127.12 Instantiation

The example below shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;
library gaisler;
```

```
use gaisler.misc.all;

entity spi2ahb_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- SPI signals
    miso : inout std_logic;
    mosi : in    std_logic;
    sck  : in    std_logic;
    sel  : in    std_logic;
    );
end;

architecture rtl of spi2ahb_ex is
  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector;
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector;
  -- SPI signals
  signal spislvi : spi_in_type;
  signal spislvo : spi_out_type;
begin

  -- AMBA Components are instantiated here
  ...
  -- SPI to AHB bridge
  spibridge : if CFG_SPI2AHB /= 0 generate
    withapb : if CFG_SPI2AHB_APB /= 0 generate
      spi2ahb0 : spi2ahb_apb
        generic map(hindex => 10,
          ahbaddrh => CFG_SPI2AHB_ADDRH, ahbaddrl => CFG_SPI2AHB_ADDRL,
          ahbmaskh => CFG_SPI2AHB_MASKH, ahbmaskl => CFG_SPI2AHB_MASKL,
          resen => CFG_SPI2AHB_RESEN, pindex => 11, paddr => 11, pmask => 16#fff#,
          pirq => 11, filter => CFG_SPI2AHB_FILTER, cpol => CFG_SPI2AHB_CPOL,
          cpha => CFG_SPI2AHB_CPHA)
        port map (rstn, clkm, ahbmi, ahbmo(10),
                  apbi, apbo(11), spislvi, spislvo);
    end generate;
    woapb : if CFG_SPI2AHB_APB = 0 generate
      spi2ahb0 : spi2ahb
        generic map(hindex => 10,
          ahbaddrh => CFG_SPI2AHB_ADDRH, ahbaddrl => CFG_SPI2AHB_ADDRL,
          ahbmaskh => CFG_SPI2AHB_MASKH, ahbmaskl => CFG_SPI2AHB_MASKL,
          filter => CFG_SPI2AHB_FILTER,
          cpol => CFG_SPI2AHB_CPOL, cpha => CFG_SPI2AHB_CPHA)
        port map (rstn, clkm, ahbmi, ahbmo(10),
                  spislvi, spislvo);
    end generate;
    spislv_miso_pad : iopad generic map (tech => padtech)
      port map (miso, spislvo.miso, spislvo.misooen, spislvi.miso);
    spislvl_mosi_pad : inpad generic map (tech => padtech)
      port map (miso, spislvi.mosi);
    spislv_sck_pad  : inpad generic map (tech => padtech)
      port map (sck, spislvi.sck);
    spislv_slvsel_pad : iopad generic map (tech => padtech)
      port map (sel, spislvi.spisel);
  end generate;
  nospibridge : if CFG_SPI2AHB = 0 or CFG_SPI2AHB_APB = 0 generate
    apbo(11) <= apb_none;
  end generate;
end;
```

128 SPICTRL - SPI Controller

128.1 Overview

The core provides a link between the AMBA APB bus and the Serial Peripheral Interface (SPI) bus and can be dynamically configured to function either as a SPI master or a slave. The SPI bus parameters are highly configurable via registers. Core features also include configurable word length, bit ordering, clock gap insertion, automatic slave select and automatic periodic transfers of a specified length. All SPI modes are supported and optionally also dual SPI, quad SPI, and a 3-wire protocol where one bidirectional data line is used. In slave mode the core synchronizes the incoming clock and can operate in systems where other SPI devices are driven by asynchronous clocks.

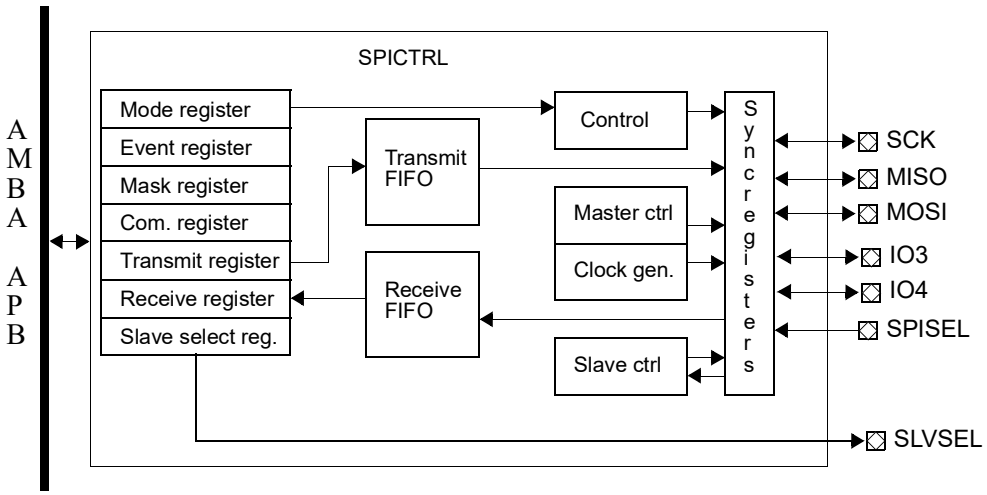


Figure 323. Block diagram

128.2 Operation

128.2.1 SPI transmission protocol

The SPI bus is a full-duplex synchronous serial bus. Transmission starts when a master selects a slave through the slave's Slave Select (SLVSEL) signal and the clock line SCK transitions from its idle state. Data is transferred from the master through the Master-Output-Slave-Input (MOSI) signal and from the slave through the Master-Input-Slave-Output (MISO) signal. In a system with only one master and one slave, the Slave Select input of the slave may be always active and the master does not need to have a slave select output. If the core is configured as a master it will monitor the SPISEL signal to detect collisions with other masters, if SPISEL is activated the master will be disabled.

During a transmission on the SPI bus data is either changed or read at a transition of SCK. If data has been read at edge n, data is changed at edge n+1. If data is read at the first transition of SCK the bus is said to have clock phase 0, and if data is changed at the first transition of SCK the bus has clock phase 1. The idle state of SCK may be either high or low. If the idle state of SCK is low, the bus has clock polarity 0 and if the idle state is high the clock polarity is 1. The combined values of clock polarity (CPOL) and clock phase (CPHA) determine the mode of the SPI bus. Figure 324 shows one byte (0x55) being transferred MSb first over the SPI bus under the four different modes. Note that the idle state of the MOSI line is '1' and that CPHA = 0 means that the devices must have data ready before the first transition of SCK. The figure does not include the MISO signal, the behavior of this line is the same as for the MOSI signal. However, due to synchronization issues the MISO signal will be delayed when the core is operating in slave mode, please see section 128.2.6 for details.

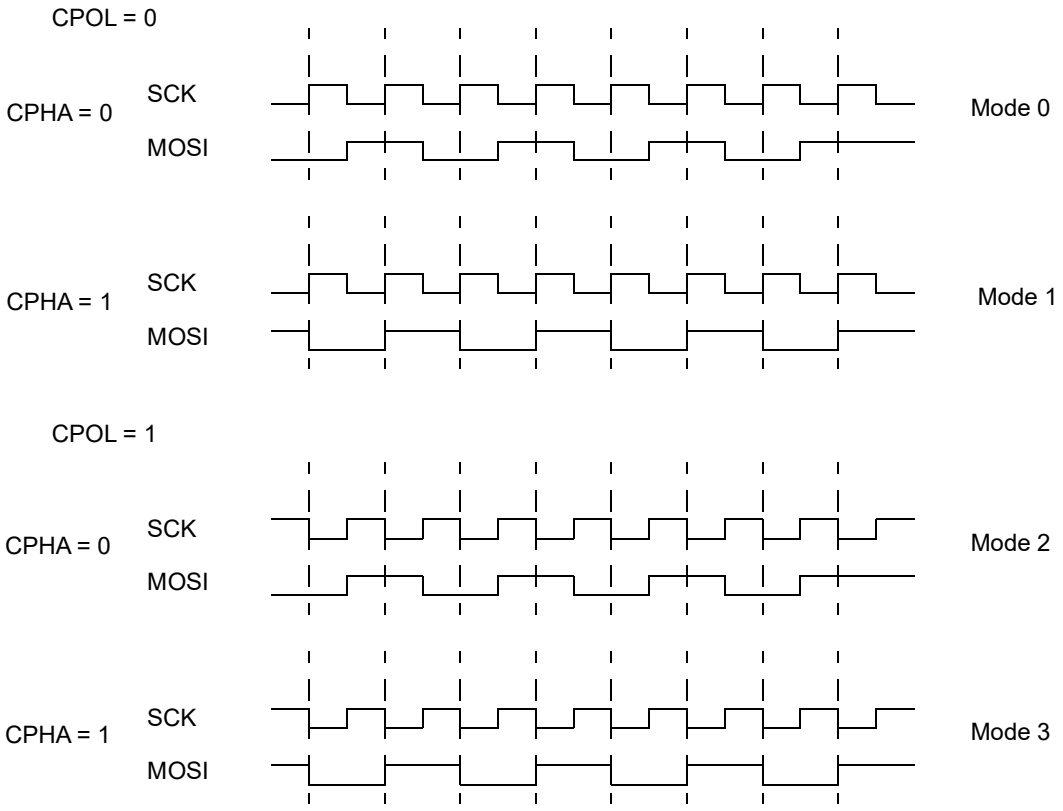


Figure 324. SPI transfer of byte 0x55 in all modes

128.2.2 3-wire transmission protocol

The core can be configured to use a 3-wire protocol if the TWEN field in the core's Capability register 0 is set to '1', where the controller uses a bidirectional dataline instead of separate data lines for input and output data. In 3-wire protocol the bus is thus a half-duplex synchronous serial bus. Transmission starts when a master selects a slave through the slave's Slave Select (SLVSEL) signal and the clock line SCK transitions from its idle state. Only the Master-Output-Slave-Input (MOSI) signal is used for data transfer in the 3-wire protocol. The MISO signal is not used.

The direction of the first data transfer is determined by the value of the 3-wire Transfer Order (TTO) field in the core's Mode register. If TTO is '0', data is first transferred from the master (through the MOSI signal). After a word has been transferred, the slave uses the same data line to transfer a word back to the master. If TTO is '1' data is first transferred from the slave to the master. After a word has been transferred, the master uses the MOSI line to transfer a word back to the slave.

The data line transitions depending on the clock polarity and clock phase in the same manner as in SPI mode. The aforementioned slave delay of the MISO signal in SPI mode will affect the MOSI signal when using 3-wire protocol, when the core operates as a slave.

128.2.3 Dual and quad transmission protocols

The core can be configured to use dual and quad SPI protocols if support for these protocols is indicated in the PROT field of the core's Capability register 1. The SPROT field of the core's Command register determines the protocol to use.

In standard SPI mode, a master always transmits on the MOSI line and receives on the MISO line and vice-versa for a slave. With dual SPI protocol the slave or master may use both of MISO and MOSI for reception or transmission. In quad SPI mode the slave or master will use MISO, MOSI, IO2 and IO3 for either transmission or reception. Since this all traffic in this controller is driven via the trans-

mit queue it becomes necessary to mark the words to be transmitted so that the controller knows which direction that should be used for the MISO, MOSI, IO2 and IO3 lines in the corresponding transfer. The direction associated with the transfer is determined by the setting of the DIR field in the core's Command register.

128.2.4 Receive and transmit queues

The core's transmit queue consists of the transmit register and the transmit FIFO. The receive queue consists of the receive register and the receive FIFO. The total number of words that can exist in each queue is thus the FIFO depth plus one. When the core has one or more free slots in the transmit queue it will assert the Not full (NF) bit in the event register. Software may only write to the transmit register when this bit is asserted. When the core has received a word, as defined by word length (LEN) in the Mode register, it will place the data in the receive queue. When the receive queue has one or more elements stored the Event register bit Not empty (NE) will be asserted. The receive register will only contain valid data if the Not empty bit is asserted and software should not access the receive register unless this bit is set. If the receive queue is full and the core receives a new word, an overrun condition will occur. The received data will be discarded and the Overrun (OV) bit in the Event register will be set.

The core will also detect underrun conditions. An underrun condition occurs when the core is selected, via SPISEL, and the SCK clock transitions while the transmit queue is empty. In this scenario the core will respond with all bits set to '1' and set the Underrun (UN) bit in the Event register. An underrun condition will never occur in master mode. When the master has an empty transmit queue the bus will go into an idle state.

128.2.5 Clock generation

The core only generates the clock in master mode, the generated frequency depends on the system clock frequency and the Mode register fields DIV16, FACT, and PM. Without DIV16 the SCK frequency is:

$$SCKFrequency = \frac{AMBAclockfrequency}{(4-(2 \cdot FACT)) \cdot (PM+1)}$$

With DIV16 enabled the frequency of SCK is derived through:

$$SCKFrequency = \frac{AMBAclockfrequency}{16 \cdot (4-(2 \cdot FACT)) \cdot (PM+1)}$$

Note that the fields of the Mode register, which includes DIV16, FACT and PM, should not be changed when the core is enabled. If the FACT field is set to 0 the core's register interface is compatible with the register interface found in MPC83xx SoCs. If the FACT field is set to 1, the core can generate an SCK clock with higher frequency.

128.2.6 Slave operation

When the core is configured for slave operation it does not drive any SPI signal until the core is selected, via the SPISEL input, by a master. If the core operates in SPI mode when SPISEL goes low the core configures MISO as an output and drives the value of the first bit scheduled for transfer. If the core is configured to use 3-wire protocol then the core will first listen to the MOSI line and when a word has been transferred drive the response on the MOSI line. If the core is selected when the transmit queue is empty it will transfer a word with all bits set to '1' and the core will report an underflow.

Since the core synchronizes the incoming clock it will not react to transitions on SCK until two system clock cycles have passed. This leads to a delay of three system clock cycles when the data output line should change as the result of a SCK transition. This constrains the maximum input SCK frequency of the slave to (system clock) / 8 or less. The controlling master must also allow the decreased setup time on the slave data out line.

The core can also filter the SCK input. The value of the PM field in the Mode register defines for how many system clock cycles the SCK input must be stable before the core accepts the new value. If the PM field is set to zero, then the maximum SCK frequency of the slave is, as stated above, (system clock) / 8 or less. For each increment of the PM field the clock period of SCK must be prolonged by two times the system clock period as the core will require longer time discover and respond to SCK transitions.

128.2.7 Master operation

When the core is configured for master operation it will transmit a word when there is data available in the transmit queue. When the transmit queue is empty the core will drive SCK to its idle state. If the SPISEL input goes low during master operation the core will abort any active transmission and the Multiple-master error (MME) bit will be asserted in the Event register. If a Multiple-master error occurs the core will be disabled. Note that the core will react to changes on SPISEL even if the core is operating in loop mode and that the core can be configured to ignore SPISEL by setting the IGSEL field in the Mode register.

128.2.8 Automated periodic transfers

The core supports automated periodic transfers if the AMODE field in the core's Capability register is '1'. In this mode the core will perform transfers with a specified period and length. The steps below outline how to set up automated transfers:

1. Configure the core's Mode register as a master and set the AMEN field (bit 31) to '1'. Possibly also configure the automatic slave select settings.
2. Write to the AM Mask registers to configure which parts of the AM transmit queue that will be used. The number of bits in the AM Mask registers that are set to one together with the word length (set in the Mode register) defines how long the transfer should be.
3. Write data to the AM transmit queue (AM Transmit registers). Only those registers that correspond to a bit that is set to one in the AM Mask registers need to be written.
4. Set the transfer period in the AM Period register.
5. Set the options for the automated transfers in the AM Configuration register
6. Set the ACT or EACT field in the AM Configuration register.
7. Wait for the Not Empty field to be set in the Event register
8. Read out the AM Receive queue (AM Receive registers). If lock bit (LOCK) in AM Configuration register is set then all registers which have a bit in the AM Mask registers set must be read. If the lock bit is not set software does not need to read out any data, the core can write new data to the AM Receive registers anyway.
9. Go back to step 7.

When an automated transfer is performed, data is not immediately placed in AM receive queue. Instead the data is placed in a temporary queue to ensure that a full transfer can be read out atomically without interference from incoming data.

The AM receive queue is filled with the data from the temporary queue if the AM receive queue is empty, or if it is full and Sequential transfers (SEQ) is disabled in the AM Configuration register. It is possible to configure the core not to place new data in the AM receive queue while software is reading

out data from the queue. This is done by setting the lock bit (LOCK) in the AM Configuration register.

If the AM Configuration register's SEQ bit is set the core will not move data from the temporary queue until the AM receive queue has been cleared. Demanding Sequential transfers means that the AM receive queue's data will never be overwritten. However, data may still be lost, depending on the settings that determine how the temporary queue handles overflow conditions.

The controller will attempt to place data into the temporary receive queue when the automated transfer period counter reaches zero. If the temporary queue is filled, which can occur if the controller is prevented from moving the data to the receive queue, the core's behavior will depend on the setting of the Strict Period (STRICT) field in the AM Configuration register:

If the value of STRICT is '0' the core will delay the transfer and wait until the temporary queue has been cleared.

If the value of STRICT is '1', and the contents of the temporary queue can not be moved to the AM receive queue, there will be an overflow condition in the temporary queue. The core's behavior on a temporary queue overflow is defined by the AM Configuration register fields Overflow Transfer Behavior (OVTB) and Overflow Data Behavior (OVDB). If there is a temporary queue overflow and OVTB is set, the transfer will be skipped and the core's internal period counter will be reloaded. If the OVTB bit is not set the transfer will be performed. If the transfer is performed and the OVDB bit is set the data will be disregarded. If the OVDB bit is not set the data will be placed in the temporary receive queue and the previous data will be overwritten.

A series of automated transfers can be started by an external event. If the AM Configuration register field EACT is set, the core will activate Automated transfers when its internal ASTART input signal goes high. When the core detects that EACT and ASTART are both set, it will set the AM Configuration register ACT bit and reset the EACT bit. Note that subsequent automated transfers will be started when the period counter reaches zero, if ERPT field of the AM Configuration register is set to zero. If the ERPT field is set to one then the ASTART input is used to start subsequent transfers instead.

When automated transfers are enabled by setting the AM Configuration register ACT bit, the core will send a pulse on its internal ASTART output signal. This means that several cores can be connected together and have their start event synchronized. To synchronize a start event, set the EACT bit in all cores, except in the last core which is activated by setting the AM Configuration register ACT field. The last core will then pulse its ASTART output and trigger the start event in all the other connected cores. When this has been done the cores' transfers will be synchronized. However this synchronization may be lost if a core's receive queues are filled and STRICT transfers are disabled, since this will lead to a delay in the start of the core's next transfer.

When the core operates in AM mode, the Receive and Transmit registers should not be accessed. Nor should the AM transmit registers be updated when automatic transfers are enabled.

128.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 2333.*SPI controller registers

APB address offset	Register
0x00	Capability register 0
0x04	Capability register 1
0x08-0x1C	Reserved
0x20	Mode register
0x24	Event register
0x28	Mask register
0x2C	Command register
0x30	Transmit register
0x34	Receive register
0x38	Slave Select register (optional)
0x3C	Automatic slave select register*
0x40	AM Configuration register**
0x44	AM Period register**
0x48-0x4C	Reserved
0x50-0x5C	AM Mask register(s)***
0x200-0x3FC	AM Transmit register(s)****
0x400-0x5FC	AM Receive register(s)****

*Only available if ASEL (bit 17) in the SPI controller Capability register is set.
**Only available if AMODE (bit 18) in the SPI controller Capability register is set.

***Only available if AMODE (bit 18) in the SPI controller Capability register is set. Number of implemented registers depend on FDEPTH (bits 15:8) in the SPI controller Capability register in the following way: Number of registers = (FDEPTH-1)/32 + 1.

****Only available if AMODE (bit 18) in the SPI controller Capability register is set. Number of implemented registers equals FDEPTH (bits 15:8) in the SPI controller Capability register.

128.3.1 SPI Controller Capability Register 0

Table 2334. 0x00 - CAP0 - SPI controller Capability register 0

31	24	23	20	19	18	17	16
SSSZ		MAXWLEN		TWEN	AMODE	ASELA	SSEN
*		*		*	*	*	‘
r		r		r	r	r	r

15	8	7	6	5	4	0
FDEPTH		SR	FT		REV	
*		*	*		5	
r		r	r		r	

31 : 24 Slave Select register size (SSSZ) - If the core has been configured with slave select signals this field contains the number of available signals. This field is only valid is the SSEN bit (bit 16) is ‘1

23 : 20 Maximum word Length (MAXWLEN) - The maximum word length supported by the core:
0b0000 - 4-16, and 32-bit word length
0b0011-0b1111 - Word length is MAXWLEN+1, allows words of length 4-16 bits.
The core must not be configured to use a word length greater than what is defined by this register.

19 3-wire Protocol Enable (TWEN) - If this bit is ‘1’ the core supports 3-wire protocol. See also the PROT field in Capability register 1.

18 Auto mode (AMODE) - If this bit is ‘1’ the core supports Automated transfers.

17 Automatic slave select available (ASELA) - If this bit is set, the core has support for setting slave select signals automatically.

16 Slave Select Enable (SSEN) - If the core has a slave select register, and corresponding slave select lines, the value of this field is one. Otherwise the value of this field is zero.

15 : 8 FIFO depth (FDEPTH) - This field contains the depth of the core’s internal FIFOs. The number of words the core can store in each queue is FDEPTH+1, since the transmit and receive registers can contain one word each.

7 SYNCRAM (SR) - If this field is ‘1’ the core has buffers implemented with SYNCRAM components.

6 : 5 Fault-tolerance (FT) - This field signals if the core has any fault-tolerant capabilities. “00” - No fault-tolerance. “01” - Parity DMR, “10” - TMR.

4 : 0 Core revision (REV) - This manual applies to core revision 5.

128.3.2 SPI Controller Capability Register 1

Table 2335. 0x04 - CAP1 - SPI controller Capability register 1

31	2	1	0
R		PROT	
0		*	
r		r	

31 : 2 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

1 : 0 Protocols (PROT) - This field shows which SPI protocols that the core supports. Note that support for the 3-wire protocol is shown using a separate bit in Capability register 0. The values for the PROT field are decoded in the following way:
0 - Support for standard SPI protocol
1 - Support for standard and dual SPI protocols
2 - Support for standard, dual and quad SPI protocols

128.3.3 SPI Controller Mode Register

*Table 2336.*0x20 - MODE - SPI controller Mode register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AMEN	LOOP	CPOL	CPHA	DIV16	REV	MS	EN	LEN				PM			
0	0	0	0	0	0	0	0	0				0			
rw*	rw	rw	rw	rw	rw	rw	rw	rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TWEN	ASEL	FACT	OD	CG					ASELDEL		TAC	TTO	IGSEL	CITE	R
0	0	0	0	0					0		0	0	0	*	0
rw*	rw*	rw	rw*	rw					rw*		rw	rw	rw	rw	r

31 Auto mode enable (AMEN) - When this bit is set to '1' the core will be able to perform automated periodic transfers. See the AM registers below. The core supports this mode if the AMODE field in the capability register is set to '1'. Otherwise writes to this field has no effect. When this bit is set to '1' the core can only perform automated transfers. Software is allowed to initialize the transmit queue and to read out the receive queue but no transfers except the automated periodic transfers may be performed. The core must be configured to act as a master (MS field set to '1') when performing automated transfers.

30 Loop mode (LOOP) - When this bit is set, and the core is enabled, the core's transmitter and receiver are interconnected and the core will operate in loopback mode. The core will still detect, and will be disabled, on Multiple-master errors.

29 Clock polarity (CPOL) - Determines the polarity (idle state) of the SCK clock.

28 Clock phase (CPHA) - When CPHA is '0' data will be read on the first transition of SCK. When CPHA is '1' data will be read on the second transition of SCK.

27 Divide by 16 (DIV16) - Divide system clock by 16, see description of PM field below and see section 128.2.5 on clock generation. This bit has no significance in slave mode.

26 Reverse data (REV) - When this bit is '0' data is transmitted LSB first, when this bit is '1' data is transmitted MSB first. This bit affects the layout of the transmit and receive registers.

25 Master/Slave (MS) - When this bit is set to '1' the core will act as a master, when this bit is set to '0' the core will operate in slave mode.

24 Enable core (EN) - When this bit is set to '1' the core is enabled. No fields in the mode register should be changed while the core is enabled. This can bit can be set to '0' by software, or by the core if a multiple-master error occurs.

23 : 20 Word length (LEN) - The value of this field determines the length in bits of a transfer on the SPI bus. Values are interpreted as:

0b0000 - 32-bit word length

0b0001-0b0010 - Illegal values

0b0011-0b1111 - Word length is LEN+1, allows words of length 4-16 bits.

The value of this field must never specify a word length that is greater than the maximum allowed word length specified by the MAXWLEN field in the Capability register.

19 : 16 Prescale modulus (PM) - This value is used in master mode to divide the system clock and generate the SPI SCK clock. The value in this field depends on the value of the FACT bit.

If bit 13 (FACT) is '0':The system clock is divided by 4*(PM+1) if the DIV16 field is '0' and 16*4*(PM+1) if the DIV16 field is set to '1'. The highest SCK frequency is attained when PM is set to 0b0000 and DIV16 to '0', this configuration will give a SCK frequency that is (system clock)/4. With this setting the core is compatible with the SPI register interface found in MPC83xx SoCs.

If bit 13 (FACT) is '1': The system clock is divided by 2*(PM+1) if the DIV16 field is '0' and 16*2*(PM+1) if the DIV16 field is set to '1'. The highest SCK frequency is attained when PM is set to 0b0000 and DIV16 to '0', this configuration will give a SCK frequency that is (system clock)/2.

In slave mode the value of this field defines the number of system clock cycles that the SCK input must be stable for the core to accept the state of the signal. See section 128.2.6.

15 3-wire protocol (TW) - If this bit is set to '1' the core will operate using 3-wire protocol. This bit can only be set if the TWEN field of the Capability register is set to '1'.

Table 2336. 0x20 - MODE - SPI controller Mode register

14	Automatic slave select (ASEL) - If this bit is set to '1' the core will swap the contents in the Slave select register with the contents of the Automatic slave select register when a transfer is started and the core is in master mode. When the transmit queue is empty, the slave select register will be swapped back. Note that if the core is disabled (by writing to the core enable bit or due to a multiple-master-error (MME)) when a transfer is in progress, the registers may still be swapped when the core goes idle. This bit can only be set if the ASELA field of the Capability register is set to '1'. Also see the ASELDEL field which can be set to insert a delay between the slave select register swap and the start of a transfer.
13	PM factor (FACT) - If this bit is 1 the core's register interface is no longer compatible with the MPC83xx register interface. The value of this bit affects how the PM field is utilized to scale the SPI clock. See the description of the PM field.
12	Open drain mode (OD) - If this bit is set to '0', all pins are configured for operation in normal mode. If this bit is set to '1' all pins are set to open drain mode. The implementation of the core may or may not support open drain mode. If this bit can be set to '1' by writing to this location, the core supports open drain mode. The pins driven from the slave select register are not affected by the value of this bit.
11 : 7	Clock gap (CG) - The value of this field is only significant in master mode. The core will insert CG SCK clock cycles between each consecutive word. This only applies when the transmit queue is kept non-empty. After the last word of the transmit queue has been sent the core will go into an idle state and will continue to transmit data as soon as a new word is written to the transmit register, regardless of the value in CG. A value of 0b00000 in this field enables back-to-back transfers.
6 : 5	Automatic Slave Select Delay (ASELDEL) - If the core is configured to use automatic slave select (ASEL field set to '1') the core will insert a delay corresponding to ASELDEL*(SPI SCK cycle time)/2 between the swap of the slave select registers and the first toggle of the SCK clock. As an example, if this field is set to "10" the core will insert a delay corresponding to one SCK cycle between assigning the Automatic slave select register to the Slave select register and toggling SCK for the first time in the transfer. This field can only be set if the ASELA field of the Capability register is set to '1'.
4	Toggle Automatic slave select during Clock Gap (TAC) - If this bit is set, and the ASEL field is set, the core will perform the swap of the slave select registers at the start and end of each clock gap. The clock gap is defined by the CG field and must be set to a value >= 2 if this field is set. This field can only be set if the ASELA field of the Capability register is set to '1'.
3	3-wire Transfer Order (TTO) - This bit controls if the master or slave transmits a word first in 3-wire protocol. If this bit is '0', data is first transferred from the master to the slave. If this bit is '1', data is first transferred from the slave to the master. This bit can only be set if the TWEN field of the Capability register is set to '1'.
2	Ignore SPISEL input (IGSEL) - If this bit is set to '1' then the core will ignore the value of the SPISEL input.
1	Require Clock Idle for Transfer End (CITE) - If this bit is '0' the core will regard the transfer of a word as completed when the last bit has been sampled. If this bit is set to '1' the core will wait until it has set the SCK clock to its idle level (see CI field) before regarding a transfer as completed. This setting only affects the behavior of the TIP status bit, and automatic slave select toggling at the end of a transfer, when the clock phase (CP field) is '0'.
0	RESERVED (R) - Read as zero and should be written as zero to ensure forward compatibility.

128.3.4 SPI Controller Event Register

*Table 2337.*0x24 - EVENT - SPI controller Event register

31	30 16	15	14	13	12	11	10	9	8	7 0
TIP	R	AT	LT	R	OV	UN	MME	NE	NF	R
0	0	0	0	0	0	0	0	0	0	0
r	r	r	wc	r	wc	wc	wc	r	r	r

31 Transfer in progress (TIP) - This bit is '1' when the core has a transfer in progress. Writes have no effect. This bit is set when the core starts a transfer and is reset to '0' once the core considers the transfer to be finished. Behavior affected by setting of CITE field in Mode register.

30 : 16 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

15 Automated transfers (AT) - This bit is '1' when the core has an automated transfer in progress. This bit is cleared automatically by the core. Writes have no effect.

14 Last character (LT) - This bit is set when a transfer completes if the transmit queue is empty and the LST bit in the Command register has been written. This bit is cleared by writing '1', writes of '0' have no effect.

13 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

12 Overrun (OV) - This bit gets set when the receive queue is full and the core receives new data. The core continues communicating over the SPI bus but discards the new data. This bit is cleared by writing '1', writes of '0' have no effect.

11 Underrun (UN) - This bit is only set when the core is operating in slave mode. The bit is set if the core's transmit queue is empty when a master initiates a transfer. When this happens the core will respond with a word where all bits are set to '1'. This bit is cleared by writing '1', writes of '0' have no effect.

10 Multiple-master error (MME) - This bit is set when the core is operating in master mode and the SPISEL input goes active. In addition to setting this bit the core will be disabled. This bit is cleared by writing '1', writes of '0' have no effect.

9 Not empty (NE) - This bit is set when the receive queue contains one or more elements. It is cleared automatically by the core, writes have no effect.

8 Not full (NF) - This bit is set when the transmit queue has room for one or more words. It is cleared automatically by the core when the queue is full, writes have no effect. This bit is only updated when the core is enabled (EN field of Mode register is set to '1').

7 : 0 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

128.3.5 SPI Controller Mask Register

*Table 2338.*0x28 - MASK - SPI controller Mask register

31	30 16	15	14	13	12	11	10	9	8	7 0
TIPE	R	AT	LTE	R	OVE	UNE	MMEE	NEE	NFE	R
0	0	0	0	0	0	0	0	0	0	0
rw	r	rw	rw	rw	rw	rw	rw	rw	rw	r

31	Transfer in progress enable (TIPE) - When this bit is set the core will generate an interrupt when the TIP bit in the Event register transitions from ‘0’ to ‘1’.
30 : 16	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
15	Automated transfers (AT) - When this bit is set, the core will generate an interrupt when an automated transfer is completed.
14	Last character enable (LTE) - When this bit is set the core will generate an interrupt when the LT bit in the Event register transitions from ‘0’ to ‘1’.
13	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
12	Overrun enable (OVE) - When this bit is set the core will generate an interrupt when the OV bit in the Event register transitions from ‘0’ to ‘1’.
11	Underrun enable (UNE) - When this bit is set the core will generate an interrupt when the UN bit in the Event register transitions from ‘0’ to ‘1’.
10	Multiple-master error enable (MMEE) - When this bit is set the core will generate an interrupt when the MME bit in the Event register transitions from ‘0’ to ‘1’.
9	Not empty enable (NEE) - When this bit is set the core will generate an interrupt when the NE bit in the Event register transitions from ‘0’ to ‘1’.
8	Not full enable (NFE) - When this bit is set the core will generate an interrupt when the NF bit in the Event register transitions from ‘0’ to ‘1’.
7 : 0	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

128.3.6 SPI Controller Command Register

Table 2339. 0x2C - CMD - SPI controller Command register

31 23	22	21 4	3	2	1 0
R	LST	R	PGRD	DIR	SPROT
0	0	0	0	0	0
r	w*	r	w*	rw*	rw*

31 : 23 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

22 Last (LST) - After this bit has been written to '1' the core will set the Event register bit LT when a character has been transmitted and the transmit queue is empty. If the core uses the 3-wire protocol then the Event register bit is set when the whole transfer has completed. This bit is automatically cleared when the Event register bit has been set and is always read as zero.

As of revision 6 of this SPI controller the LST bit can only be set to '1'. Previous revisions of the core allowed the LST bit to be set to '0' by writing to this bit. As of revision 6, writes of zero do not affect the core's internal state of the LST bit.

21 : 4 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

3 Protocol guard (PGRD) - This field must be set to '1' to modify the IO and SPROT fields. If the value of this field is 0 for a write operation then no change will be made to IO and SPROT.

2 Direction (DIR) - This field determines the direction for transfers when the SPROT field is non-zero.
0 - Output
1 - Input
This field can only be modified when the PGRD field is set in the write data.

1 : 0 SPI protocol (SPROT) - This field selects between standard, dual and quad SPI mode. The settings are:

0b00 - Standard SPI mode
0bx1 - Dual SPI mode
0b10 - Quad SPI mode

Dual and quad SPI modes can only be enabled if they are supported by the implementation as indicated by the PROT field in Capability register 1. This field must not be changed while the core is transmitting or receiving data. The same protocol is also used for the words received while transmitting.

This field must be set to 0 when using 3-wire protocol (controlled via Mode register).

This field can only be modified when the PGRD field is set in the write data.

128.3.7 SPI Controller Transmit Register

Table 2340. 0x30 - TX - SPI controller Transmit register

31 0
TDATA
0
w

31 : 0 Transmit data (TDATA) - Writing a word into this register places the word in the transmit queue. This register will only react to writes if the Not full (NF) bit in the Event register is set. The layout of this register depends on the value of the REV field in the Mode register:

Rev = '0': The word to transmit should be written with its least significant bit at bit 0.

Rev = '1': The word to transmit should be written with its most significant bit at bit 31.

128.3.8 SPI Controller Receive Register

*Table 2341.*0x34 - RXC - SPI controller Receive register

31	0
RDATA	
0	
r	

31 : 0 Receive data (RDATA) - This register contains valid receive data when the Not empty (NE) bit of the Event register is set. The placement of the received word depends on the Mode register fields LEN and REV:

For LEN = 0b0000 - The data is placed with its MSb in bit 31 and its LSb in bit 0.

For other lengths and REV = '0' - The data is placed with its MSB in bit 15.

For other lengths and REV = '1' - The data is placed with its LSB in bit 16.

To illustrate this, a transfer of a word with eight bits (LEN = 7) that are all set to one will have the following placement:

REV = '0' - 0x0000FF00

REV = '1' - 0x00FF0000

128.3.9 SPI Slave Select Register (optional)

*Table 2342.*0x38 - SLVSEL - SPI Slave select register (optional)

31 SSSZ	SSSZ-1 0
R	SLVSEL
0	all 1
r	rw

31 : SSSZ RESERVED (R) - The lower bound of this register is determined by the Capability register field SSSZ if the SSEN field is set to 1. If SSEN is zero bits 31:0 are reserved.

(SSSZ-1) : 0 Slave select (SLVSEL) - If SSEN in the Capability register is 1 the core's slave select signals are mapped to this register on bits (SSSZ-1):0. Software is solely responsible for activating the correct slave select signals, the core does not assert or deassert any slave select signal automatically.

128.3.10SPI Controller Automatic Slave Select Register

*Table 2343.*0x3C - ASLVSEL - SPI controller Automatic slave select register

31 SSSZ	SSSZ-1 0
R	ASLVSEL
0	0
r	rw

31 : SSSZ RESERVED (R) - The lower bound of this register is determined by the Capability register field SSSZ if the SSEN field is set to 1. If SSEN is zero bits 31:0 are reserved.

(SSSZ-1) : 0 Automatic Slave select (ASLVSEL) - If SSEN and ASELA in the Capability register are both '1' the core's slave select signals are assigned from this register when the core is about to perform a transfer and the ASEL field in the Mode register is set to '1'. After a transfer has been completed the core's slave select signals are assigned the original value in the slave select register.

Note: This register is only available if ASELA (bit 17) in the SPI controller Capability register is set

128.3.11 SPI Controller AM Configuration Register

Table 2344. 0x40 - AMCFG - SPI controller AM configuration register

31	30									16
RESERVED										
0										
r										

15	9	8	7	6	5	4	3	2	1	0
RESERVED		ECGC	LOCK	ERPT	SEQ	STRICT	OVTB	OVDB	ACT	EACT
0		0	0	0	0	0	0	0	0	0
r		rw	rw	rw	rw	rw	rw	rw	rw	rw

31 : 9 RESERVED - This field is reserved for future use and should always be written as zero.

8 External clock gap control (ECGC) - If software sets this bit to '1' then the clock gap between individual transfers in a set of automated transfers is controller by the core's CSTART input instead of the CG field in the Mode registers. Note that the requirement that the CG field must be set to a value >= 2 if the TAC bit is set still applies even if this bit is set. Reset value '0'.

7 Lock bit (LOCK) - If software sets this bit to '1' then the core will not place new data in the AM Receive registers while software is reading out new data.

6 External repeat (ERPT) - When this bit is set the core will use the input signal astart to start a new periodic transfer. If this bit is cleared, the period counter will be used instead.

5 Sequential transfers (SEQ) - When this bit is set the core will not update the receive queue unless the queue has been emptied by reading out its contents. Note that the contents in the temporary FIFO may still be overwritten with incoming data, depending on the setting of the other fields in this register.

4 Strict period (STRICT) - When this bit is set the core will always try to perform a transfer when the period counter reaches zero, if this bit is not set the core will wait until the receive FIFO is empty before it tries to perform a new transfer.

3 Overflow Transfer Behavior (OVTB) - If this bit is set to '1' the core will skip transfers that would result in data being overwritten in the temporary receive queue. Note that this bit only decides if the transfer is performed. If this bit is set to '0' a transfer will be performed and the setting of the Overflow Data Behavior bit (OVDB) will decide if data is actually overwritten.

2 Overflow Data Behavior (OVDB) - If this bit is set to '1' the core will skip incoming data that would overwrite data in the receive queues. If this bit is '0' the core will overwrite data in the temporary queue.

1 Activate automated transfers (ACT) - When this bit is set to '1' the core will start to decrement the AM period register and perform automated transfers. The system clock cycle after this bit has been written to '1' there will be a pulse on the core's ASTART output.

Automated transfers can be deactivated by writing this bit to '0'. The core will wait until any ongoing transfer has finished before deactivating automated transfers. Software should not perform any operation on the core before this bit has been read back as '0'. The data in the last transfer(s) will be lost if there is a transfer in progress when this bit is written to '0'. All words present in the transmit queue will also be dropped.

0 External activation of automated transfers (EACT) - When this bit is set to '1' the core will activate automated transfers when the core's ASTART input goes HIGH. When the core has been activated by the external signal this bit will be reset and the ACT field (bit 1 will be set).

Note: This register is only available if AMODE (bit 18) in the SPI controller Capability register is set

128.3.12SPI Controller AM Period Register

*Table 2345.*0x44 - AMPER - SPI controller AM period register

31	0
AMPER	
0	
rw	

31 : 0 AM Period (AMPER) - This field contains the period, in system clock cycles, of the automated transfers. The core has an internal counter that is decremented each system clock cycle. When the counter reaches zero the core will begin to transmit all data in the transmit queue and reload the internal counter, which will immediately begin to start count down again. If the core has a transfer in progress when the counter reaches zero, the core will stall and not start a new transfer, or reload the internal counter, before the ongoing transfer has completed.

The number of bits in this register is implementation dependent. Software should write this register with 0xFFFFFFFF and read back the value to see how many bits that are available.

Note: This register is only available if AMODE (bit 18) in the SPI controller Capability register is set

128.3.13SPI Controller AM Mask Register(s)

*Table 2346.*0x50-0x5C - AMMASK - SPI controller AM Mask register(s)

31	0
AM MASK	
0	
rw	

31 : 0 AM Mask - This field is used as a bit mask to determine which words in the AM Transmit / Receive queues to read from / write to. Bit 0 of the first mask register corresponds to the first position in the queues, bit 1 of the first mask register to the second position, bit 0 of the second mask register corresponds to the 33:d position, etc. The total number of bits implemented equals FDEPTH (bit 15:8) in the SPI controller Capability register. If a bit is set to one then the core will read / write the corresponding position in the queue, otherwise it will be skipped. Software can write these registers at all times. However if a automated transfer is in progress when the write occurs, then the core will save the new value in a temporary register until the transfer is complete. The reset value is all ones.

Note: This register is only available if AMODE (bit 18) in the SPI controller Capability register is set

128.3.14SPI Controller AM Transmit Register(s)

*Table 2347.*0x200-0x3FC - AMTX - SPI controller AM Transmit register(s)

31	0
TDATA	
0	
rw	

31 : 0 Transmit data (TDATA) - Writing a word into these register places the word in the AM Transmit queue. The address of the register determines the position in the queue. Address offset 0x200 corresponds to the first position, offset 0x204 to the second position etc.

The layout of the registers during write depends on the value of the REV field in the Mode register:
Rev = ‘0’: The word to transmit should be written with its least significant bit at bit 0.
Rev = ‘1’: The word to transmit should be written with its most significant bit at bit 31.

The layout of the registers during read is fixed, the word is read with its least significant bit at bit 0.

Note: This register is only available if AMODE (bit 18) in the SPI controller Capability register is set

128.3.15SPI Controller AM Receive Register

*Table 2348.*0x400-0x5FC - SPI controller AM Receive register(s)

31	0
RDATA	
0	
rw	

31 : 0 Receive data (RDATA) - The address of the register determines the position in the queue. Address offset 0x200 corresponds to the first position, offset 0x204 to the second position etc. The placement of the received word depends on the Mode register fields LEN and REV.

For LEN = 0b0000 - The data is placed with its MSb in bit 31 and its LSb in bit 0.

For other lengths and REV = '0' - The data is placed with its MSB in bit 15.

For other lengths and REV = '1' - The data is placed with its LSB in bit 16.

To illustrate this, a transfer of a word with eight bits (LEN = 7) that are all set to one will have the following placement:

REV = '0' - 0x0000FF00

REV = '1' - 0x00FF0000

Note: This register is only available if AMODE (bit 18) in the SPI controller Capability register is set

128.4 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x02D. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

128.5 Implementation

128.5.1 Reset

The core changes reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual).

The core will add reset for all registers if the GRLIB config package setting *grlib_sync_reset_enable_all* is set.

The core will use asynchronous reset for all registers, except synchronization registers, if the GRLIB config package setting *grlib_async_reset_enable* is set.

See also the documentation for the *syncrst* VHDL generic.

128.6 Configuration options

Table 2349 shows the configuration options of the core (VHDL generics).

*Table 2349.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by SPI controller	0 - NAHBIRQ-1	0

*Table 2349.*Configuration options

Generic name	Function	Allowed range	Default
fdepth	FIFO depth. The FIFO depth in the core is 2^{fdepth}. Note that the depth of the transmit and receive queues is FIFO depth + 1 since the Transmit and Receive registers can hold one word. The number of AM Transmit / Receive registers are however 2^{fdepth}.	1 - 7	1
slvselen	Enable Slave Select register. When this value is set to 1 the core will include a slave select register that controls slvselsz slave select signals.	0 - 1	0
slvselsz	Number of Slave Select (slvsel) signals that the core will generate. These signals can be controlled via the Slave select register if the generic slvselen has been set to 1, otherwise they are driven to '1'.	1 - 32	1
oepol	Selects output enable polarity	0 - 1	0
odmode	Open drain mode. If this generic is set to 1, the OD bit in the mode register can be set to 1 and the core must be connected to I/O or OD pads.	0 - 1	0
automode	Enable automated transfers. If this generic is set to 1 the core will include support to automatically perform periodic transfers. The core's receive and transmit queues must not contain more than 128 words if automode is enabled.	0 - 1	0
acntbits	Selects the number of bits used in the AM period counter. This generic is only of importance if the automode generic is set to 1.	1 - 32	32
aslvsel	Enable automatic slave select. If this generic is set to 1 the core will include support for automatically setting the slave select register from the automatic slave select register before a transfer, or queue of transfers, starts. This generic is only significant of the slvselen generic is set to 1.	0 - 1	0
twen	Enable 3-wire protocol. If this generic is set to 1 the core will include support for 3-wire protocol.	0 - 1	1
maxwlen	Determines the maximum supported word length. Values are defined as: 0 - Core will support lengths up to 32-bit words 0-3 - Illegal values 4-15 - Maximum word length is maxlen+1, allows words of length 4-16 bits. This generic sets the size of the slots in the transmit and receive queues. If the core will be used in an application that will never need to perform transfers with words as long as 32-bits, this setting can be used to save area.	0 - 15	0
netlist	If this generic is set to 0 (default) then the RTL version of the core will be used. If this generic is non-zero, the netlist version of the core will be used (if available) and the value of *netlist* will specify the target technology.	0 - NTECH	0
syncram	When this generic is set to 1 the core will instantiate SYNCRAM_2P components for the receive and transmit queues. The use of SYNCRAM_2P components can reduce area requirements, particularly when automode is enabled.	0 - 1	1
memtech	Selects memory technology for SYNCRAM_2P components.	0 - NTECH	0 (inferred)

*Table 2349.*Configuration options

Generic name	Function	Allowed range	Default
ft	Enables fault tolerance for receive and transmit queues. 0 - No fault tolerance, 1 - Parity DMR, 2 - TMR. This generic only has effect if generic syncram is non-zero.	0 - 2	0
scantest	Enable scan test support. Only applicable if generic syncram is /= 0.	0 - 1	0
syncrst	Use only synchronous reset. If this generic is 0 then the spio.sckoen, spio.misooen, spio.mosioen and slvsel output will have asynchronous reset. If synchronous reset for these registers is wanted then set this generic to 1. Note that if grlib_async_reset_enable in the GRLIB configuration package is enabled then this will override the syncrst setting ans all registers, except synchronization registers in the code will have asynchronous reset.	0 - 1	0
ignore	Enable AIGNORE/CIGNORE inputs (experimental)	0 - 1	0
prot	Selects what protocols (note that support for 3-wire protocol is selected by the *twen* generic). The value of prot has the following effect: 0 - Support for standard SPI protocol 1 - Support for standard and dual SPI protocols 2 - Support for standard, dual and quad SPI protocols	0 - 2	0

128.7 Signal descriptions

Table 2350 shows the interface signals of the core (VHDL ports).

*Table 2350.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-

Table 2350. Signal descriptions

Signal name	Field	Type	Function	Active
SPII	MISO	Input	Master-Input-Slave-Output data line, not used with 3-wire protocol.	-
	MOSI	Input	Master-Output-Slave-Input data line	-
	SCK	Input	Serial Clock. If the core is instantiated in a system where it will work only as a master then drive this signal constant Low to save some area.	-
	SPISEL	Input	Slave select input. This signal should be driven High if it is unused in the design.	Low
	ASTART	Input	Automated transfer start. The core can be programmed to use this signal to start a set of automated transfers. This signal should be driven low if it is unused in the design. This signal must be synchronous to the CLK input.	High
	CSTART	Input	Automated clock start. This signal can be used to control when an individual transfer in a set of automated transfers should start. This signal doesn't affect the start of the first transfer in the set. Also the core needs to be programmed to use the signal. This signal should be driven low if it is unused in the design. This signal must be synchronous to the CLK input.	High
	AIGNORE	Input	Ignore RX fifo adddress increment, ignore first TX fifo address increment	High
	CIGNORE	Input	Ignore TX fifo address increment	High
	IO2	Input	Data line 2, used with quad SPI protocol	-
	IO3	Input	Data line 3, used with quad SPI protocol	-
SPIO	MISO	Output	Master-Input-Slave-Output data line, not used in with 3-wire protocol.	-
	MISOOEN	Output	Master-Input-Slave-Output output enable, not used with 3-wire protocol.	-
	MOSI	Output	Master-Output-Slave-Input	-
	MOSIOEN	Output	Master-Output-Slave-Input output enable	-
	SCK	Output	Serial Clock	-
	SCKOEN	Output	Serial Clock output enable	-
	SSN	Output	Not used	-
	ASTART	Output	Automated transfer start indicator.	High
	AREADY	Output	Automated transfer ready indicator. Set each time an individual transfer in a set of automated transfers is completed.	High
	IO2	Output	Data line 2, used with quad SPI protocol	-
	IO2OEN	Output	Data line 2 output enable, used with quad SPI protocol	-
	IO3	Output	Data line 3, used with quad SPI protocol	-
	IO3OEN	Output	Data line 3output enable, used with quad SPI protocol	-
SLVSEL [SSSZ-1:0]	N/A	Output	Slave select output(s). Used if the *slvselen* VHDL generic is set to 1. The range of the vector is (slvselsz-1):0	-

* see GRLIB IP Library User's Manual

128.8 Signal definitions and reset values

The signals and their reset values are described in table 2351.

*Table 2351.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
sck	InputOutput	SPI clock	-	Hi-Z
miso	InputOutput	Master-Input-Slave-Output, not used with 3-wire protocol	-	Hi-Z
mosi	InputOutput	Master-Output-Slave-Input	-	Hi-Z
io2	InputOutput	Data line 2, used with quad SPI protocol	-	Hi-Z
io3	InputOutput	Data line 3, used with quad SPI protocol	-	Hi-Z
spisel	Input	Slave select input	Logical 0	-
slvsel[SSSZ-1:0]	Output	Slave select signals	Logical 0	Logical 1

128.9 Timing

The timing waveforms and timing parameters are shown in figure 325 and are defined in table 2352.

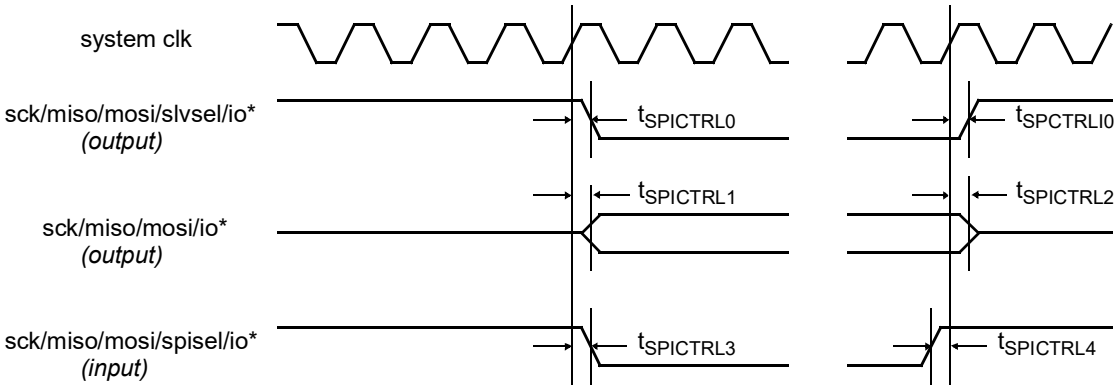


Figure 325. Timing waveforms

*Table 2352.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{SPICTRL0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns
$t_{SPICTRL1}$	clock to non-tri-state delay	rising *clk* edge	TBD	TBD	ns
$t_{SPICTRL2}$	clock to tri-state delay	rising *clk* edge	TBD	TBD	ns
$t_{SPICTRL3}$	input to clock hold	rising *clk* edge	-	-	ns
$t_{SPICTRL4}$	input to clock setup	rising *clk* edge	-	-	ns

Note: The *sck/miso/mosi/io2/io3/spisel* inputs are re-synchronized internally. The signals do not have to meet any setup or hold requirements.

128.10 Library dependencies

Table 2353 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2353.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	SPI	Component, signals	SPI component and signal definitions.
TECHMAP	GENCOMP	Constant values	Technology constants
TECHMAP	NETCOMP	Component	Netlist component

128.11 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.misc.all;

entity spi_ex is
  port (
    clk    : in std_ulogic;
    rstn   : in std_ulogic;

    -- SPI signals
    sck    : inout std_ulogic;
    miso   : inout std_ulogic;
    mosi   : inout std_ulogic;
    spisel : in std_ulogic
    );
end;

architecture rtl of spi_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- SPIsignals
  signal spii : spi_in_type;
  signal spio : spi_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- SPI controller with FIFO depth 2 and no slave select register
  spictrl0 : spictrl generic map (pindex => 10, paddr => 10, pirq => 10,
                                  fdepth => 1, slvselen => 0, slvselsz => 1)
      port map (rstn, clkm, apbi, apbo(10), spii, spio, open);

    misopad : iopad generic map (tech => padtech)
      port map (miso, spio.miso, spio.misooen, spii.miso);
    mosipad : iopad generic map (tech => padtech)
      port map (mosi, spio.mosi, spio.mosioen, spii.mosi);
    sckpad : iopad generic map (tech => padtech)
      port map (sck, spio.sck, spio.sckoen, spii.sck);
    spiselpad : inpad generic map (tech => padtech)
      port map (spisel, spii.spisel);
end;
```

129 SPIMCTRL - SPI Memory Controller

129.1 Overview

The core maps up to four memory devices connected via the Serial Peripheral Interface (SPI) into AMBA address space. Read accesses are performed by performing normal AMBA read operations in the mapped memory area. For certain memory devices, the core can be configured to handle AMBA write operations in the same way, but in most cases writes are instead handled by directly sending SPI commands to the memory device via the core's register interface. The core is highly configurable and supports most SPI Flash memory devices.

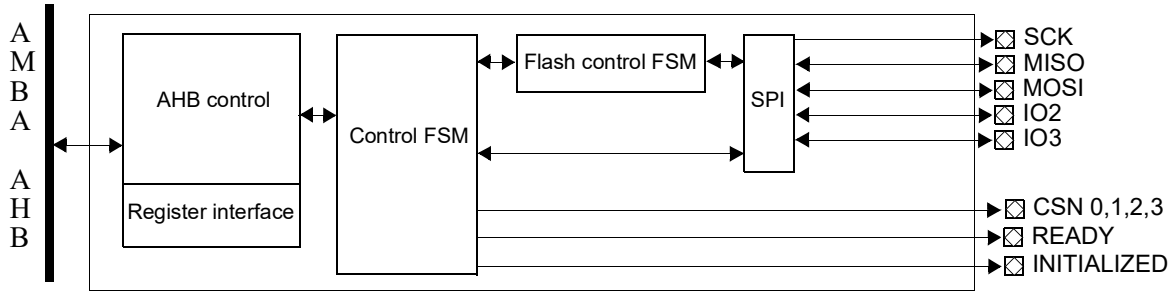


Figure 326. Block diagram

129.2 Operation

129.2.1 Operational model

The core has two memory areas that can be accessed via the AMBA bus; the I/O area and the ROM area. The ROM area maps the memory devices into AMBA address space and the I/O area is utilized for status reporting and to issue user commands to the memory device.

When transmitting SPI commands directly to the device the ROM area should be left untouched. The core will issue an AMBA ERROR response if the ROM area is accessed when the core is busy performing an operation initiated via I/O registers.

Depending on the type of device attached the core may need to perform an initialization sequence. Accesses to the ROM area during the initialization sequence receive AMBA error responses. The core has successfully performed all necessary initialization when the Initialized bit in the core's status register is set, the value of this bit is also propagated to the core's output signal *spio.initialized*.

129.2.2 I/O area

The I/O area contains registers that are used when issuing commands directly to the memory device. By default, the core operates in System mode where it will perform read operations on the memory device when the core's ROM area is accessed. Before attempting to issue commands directly to the memory device, the core must be put into User mode. This is done by setting the User Control (USRC) bit in the core's Control register. Care should be taken to not enter User mode while the core is busy, as indicated by the bits in the Status register. The core should also have performed a successful initialization sequence before User mode accesses (INIT bit in the Status register should be set).

Note that a memory device may need to be clocked when there has been a change in the state of the chip select signal. It is recommended that software transmits a byte with the memory device deselected after entering and before leaving User mode.

The following steps are performed to issue a command to the memory device after the core has been put into User mode:

1. Check Status register and verify that the BUSY and DONE bits are cleared. Also verify that the core is initialized and not in error mode.
2. Optionally enable DONE interrupt by setting the Control register bit IEN.
3. Write command to Transmit register.
4. Wait for interrupt (if enabled) or poll DONE bit in Status register.
5. When the DONE bit is set the core has transferred the command.
6. Clear the Status register's DONE bit by writing one to its position.

To receive data from the device the same process as above need to be done, but instead writing to the Receive register at step 3. The data written will not be used by the controller, the write only signals to the core that it should clock the memory device until one byte has been received. The received data is then stored in the Receive register.

The core should not be brought out of User mode until the transfer completes. Accesses to ROM address space will receive an AMBA ERROR response when the core is in User mode and when an operation initiated under User mode is active.

129.2.3 ROM area

The ROM area supports AMBA read operations and can be configured to additionally allow write operations. For write operations to be allowed the WRT register bit needs to be set, otherwise all write operations will be ignored and a AMBA ERROR response will be returned as a response. When a read or write access (if allowed) is made to the ROM area, then the core will perform an operation of the same type on the memory device. If the system has support for AMBA SPLIT responses the core will SPLIT the master until the access on the memory device has finished unless the operation is a locked access. A locked access never receives a SPLIT response, and the core inserts wait states instead. If the system lacks AMBA SPLIT support the core will always insert wait states until the read operation on the memory device has finished. The core uses the value of the VHDL generic *spliten* to determine if the system has AMBA SPLIT support.

The ROM area is marked as cacheable and prefetchable. This must be taken into account if the data in the ROM area is modified via the I/O area.

129.2.4 SPI memory device address offset

An offset can be specified at implementation via the core's *offset* VHDL generic. This offset will be added to all accesses to the ROM area before the address is propagated to the SPI memory device. Specifying an offset can be useful when the SPI memory device contains, as an example, FPGA configuration data at the lower addresses. By specifying an offset, the top of the SPI memory device can be used to hold user data. The AMBA system is unaware of the offset being added. An access to address *n* in the ROM area will be automatically translated to an access to address *offset* + *n* on the SPI memory device.

The offset must be accounted for when accessing the SPI memory device via the core's register interface. If data is programmed to the SPI memory device through the register interface, then the data must be written starting at the offset specified by the VHDL generic *offset*.

129.2.5 Multiple memory device support

The core has support to communicate with at most four separate SPI flash memories. The data signals (MISO, MOSI, IO2, IO3) and clock (SCK) is shared among all connected devices, while a separate chip select signal should be supplied to each SPI flash (CSN, CSN1, CSN2, CSN3).

When doing accesses to the I/O area, the user needs to specify to the core which out of the four chip select signals the *csn* register bit should affect using the two bit *device select* (DS) register. The chip select signals which are not currently selected are always driven high. For correct behavior it is expected that this selection either happens before, or at the same time as, the *csn* register bit is driven low for the I/O area access.

For accesses to the ROM area, which chip select is used is determined based on the incoming address and/or the four bit *device select configuration* (DSCONF) register value. Through DSCONF it is possible to configure the core to always use one of the four chip select signal, no matter the value of the incoming address. It is also possible to configure the core to do the device selection by looking at specific bit pairs in the address, with the smallest configuration effectively allocating 128KiB to each SPI device and the largest allocating 256MiB. For the incoming address, the bits where the selection happens, and also any bits above that, are masked out and will not be sent to the SPI device. More information about DSCONF can be found in the register interface description.

Writing to the DS and DSCONF registers is only possible if the *multiple_csn* VHDL generic is set. Otherwise they will be read only and the core will only be able to make use of the first chip select signal.

129.2.6 Supported memory devices

The core supports a wide range of memory devices due to its configuration options of read and write instruction, dummy cycles insertions and choice of input and output mode. These values are at reset set to the values as defined by the VHDL generic and, if the *reconf* VHDL generic is set, can be changed during runtime by writing to a configuration register.

The core is configured to issue the instruction defined by the *readcmd* and *writecmd* values. Following an access to the ROM area, the core will issue the instruction followed by 24 address bits. If the *extaddr* bit is set, 32 bits of address are sent instead. For reads operations, an optional number of dummy cycles can be transmitted to the device, as configured by the *dummybyte* and *dummycycles* values. If the *dummybyte* bit is set, 8 cycles will always be used. Otherwise, the value specified by *dummycycles* is utilized. In the case of write operations, no dummy cycles are sent, regardless of the values of *dummybyte* or *dummycycles*. After the possible dummy cycles, the core expects to either receive data from or send data to the device, depending on whether the access was a read or write operation.

For memory writes to many SPI memory devices there is often some prerequisite conditions that needs to be met before a write can occur by default, such as sending a write enable command. The core adopts a straightforward approach to handling writes and lacks logic to manage these conditions autonomously. It is the user's responsibility to configure the memory device in a manner that eliminates the need for these prerequisite conditions before a write. If configuring the memory device in such a way is not possible then writing to the ROM area of the core is discouraged as the prerequisite conditions would need to be set up using the I/O area before every write. In these cases, it is instead better to handle all writing of data completely through the I/O interface.

129.2.7 Supported SPI protocols

The SPIMCTRL can be configured to support a wide range of SPI instruction types/protocols through the *Quadinput*, *Dualinput*, *Quadoutput*, *Dualoutput*, *DSPI* and *QSPI* bits.

A typical SPI instruction consists of three phases: *command*, *address* and *data*. To easily describe the number of data lines used for each phase a three number representation is used, where each number represent a phase (command-address-data). For example, 1-1-4 means one data line is used during both the command and address phase while four are used during the data phase. For each phase, when one data line is selected MISO is used to receive data and MOSI to transmit data, when two data lines are selected both MISO and MOSI are used simultaneously to either transmit or receive data and when four data lines are selected MISO, MOSI, IO2 and IO3 are used simultaneously to either transmit or receive data.

There are three main modes you can set the controller to be in: Extended SPI (ESPI, 1-x-x), Dual SPI (DSPI, 2-2-2) or Quad SPI (QSPI, 4-4-4). The current selected mode is determined by the *DSPI* and *QSPI* bits; if neither of the two bits are set then the default ESPI mode is used and if both are set QSPI mode is used.

When in ESPI mode, the command phase of a read and also all communication through the I/O area will be sent using one data line (1-x-x).The behavior during the address phase is determined by the *[Dual/Quad]input* bits and the data phase is determined by the *[Dual/Quad]output* bits. If neither bit is set for a phase then one data lines is used, if the dual bit is set then two data lines are used and if the quad bit is set four data lines are used. If both dual and quad bits are set then quad mode is used.

When in DSPI mode, all communication with the memory happens using two data lines (2-2-2). When in the QSPI mode, all communication with the memory happens using four data lines (4-4-4). When in DSPI or QSPI mode the *Quadinput*, *Dualinput*, *Quadoutput* and *Dualoutput* bits are **ignored**.

Many memory devices support communication using more than one SPI protocol. When moving between ESPI modes it is often enough to only change the *readcmd* and one, or several, of the *Quadinput*, *Dualinput*, *Quadoutput*, *Dualoutput* bits. When changing configuration between the three main modes (ESPI, DSPI, QSPI) the memory first needs to be set to work in that mode, often through a device specific command or a write to a configuration register on the memory device (this is done using writes to the I/O area). After that is done, the SPIMCTRL needs to be configured to match the new mode the memory was set to use. If there is a mismatch between the mode the memory device and the controller are in, no meaningful communication can happen between the two devices.

In the table below it is described which configuration values need to be modified, assuming the controller is currently using the default values set in the VHDL generic, to work with some common SPI instruction types / protocols. For all configurations the device specific values *readcmd, writecmd* and *dummycycles* also needs to be set.

TABLE 2354. SPIMCTRL configurations for some common SPI protocols

Mode	1-1-1 (ESPI)	1-1-2 (ESPI)	1-2-2 (ESPI)	2-2-2 (DSPI)	1-1-4 (ESPI)	1-4-4 (ESPI)	4-4-4 (QSPI)
Configuration		*Dualoutput*=1	*Dualoutput*=1 *Dualinput*=1	*DSPI*=1	*Quadoutput*=1	*Quadoutput*=1 *Quadinput*=1	*QSPI*=1

129.2.8 eXecute-In-Place

Some SPI memories have support for commands using eXecute-In-Place (XIP). XIP allows several consecutive commands of the same type to be sent, where only the first access needs to specify the command code. For XIP compatible commands, there is an activation byte or bit that should be sent directly after the address phase to the memory controller, determining whether XIP should be enabled or disabled. For some memories, the activation code is sent during the dummy cycles phase, while for others, additional cycles are added just for the XIP activation code. For example, in the Avalanche AS1001204 memory, one additional byte needs to be sent after the address phase and before the dummy cycles. If 'Axh' is sent then XIP is enabled, while 'Fxh' disabled it. Once XIP has been enabled, subsequent commands will not require sending any command code but can immediately send their address.

The core does not have the capability to efficiently use XIP mode. If possible, XIP compatible commands should be avoided. However, for some memories, especially when trying to utilize more advanced features like QSPI, there might be no choice but to use commands that also require XIP. In these cases, the goal from the user should be to configure the SPIMCTRL so a XIP disable code is sent for every access made. Some settings exist in the core specifically for this purpose, namely *xip_byte* and *xip_polarity*. The *xip_byte* settings make it so one byte of data is sent immediately after the address phase, using the same number of bits/cycle as was used during the address phase. The *xip_polarity* setting determines the bit polarity of the data shifted into the memory device during the XIP-byte and dummy cycles phase. With these settings, the core can be configured to work seamlessly with many commands which require XIP.

129.2.9 Bursts

The SPIMCTRL is designed to efficiently manage read and write bursts request from the AMBA AHB bus. When an access is recognized as a burst, the core does not release the chip select signal after handling the access. Instead, it awaits the next access and continues servicing that request without the need to resend the command or address bytes. The core identifies a burst based on the first AMBA AHB HBURST bit, meaning that only incrementing bursts are efficiently handled. Wrapping bursts are supported but will be treated by the core individually as regular non-burst accesses.

129.2.10 Clock generation and power-up timing

The core generates the device clock by scaling the system clock. The VHDL generic *scaler* selects the divisor to use for the device clock that is used when issuing read instructions.

The alternate clock can be used for all communication by setting the Enable Alternate Scaler (EAS) bit in the Control register. When configuring the core for communication with a device in a system where the target frequency may change it is recommended to set the VHDL generic *scaler* to a conservative value and configure the alternate scaler to produce a faster clock. A boot loader can then set the Enable Alternate Scaler (EAS) bit early in the boot process when it has been determined that the system can use the memory device at a higher frequency.

If the external device needs power-up time before being accessed then this needs to be handled at the system level or by waiting to access the SPI memory controller until the power up time has passed.

129.3 Registers

The core is programmed through registers mapped into AHB address space. Only 32-bit single-accesses to the registers are supported.

Table 2355. SPIMCTRL registers

AHB address offset	Register
0x00	Configuration register
0x04	Control register 1
0x08	Status register
0x0C	Receive register
0x10	Transmit register
0x14	Reserved
0x18	Reserved
0x1C	Control register 2

129.3.1 Configuration register

*Table 2356.*0x00 - CONF - Configuration register

31 24	23-21	20	19	18	17	16	15	14	13	12	11 8	7 0
WRITECMD	RES	XIP	QIN	DIN	QOUT	DOUT	DBYTE	ADDR	QSPI	DSPI	DCYCLES	READCMD
-	0	-	-	-	-	-	-	-	-	-	-	-
r/rw*	r	r/rw*	r/rw*	r/rw*	r/rw*	r/rw*	r/rw*	r/rw*	r/rw*	r/rw*	r/rw*	r/rw*

31 :24 Write instruction (WRITECMD) - Write instruction that the core will use for writing to the memory device.

23 :21 RESERVED

20 XIP Byte (XIP) - Send one additional byte of data after the address phase containing either 0x00 or 0xFF depending on the value of *xip:_polarity*.

19 Quad input (QIN) - Use quad mode during the address phase for memory accesses when in ESP I mode.

18 Dual input (DIN) - Use dual mode during the address phase for memory accesses when in ESP I mode.

17 Quad output (QOUT) - Use quad mode during the data phase for memory accesses when in ESP I mode.

16 Dual output (DOUT) - Use dual mode during the data phase for memory accesses when in ESP I mode.

15 Dummy byte (DBYTE) - Output 8 dummy cycles after the address phase.

14 Extended Address (ADDR) - Use 32b address mode for memory accesses. When this bit is set to '0', then the default 24b address is used instead.

13 Enable QSPI (QSPI) - Enables QSPI mode. All communication to the memory device is done using quad mode. If both the QSPI and DSPI bits are '1', then QSPI mode will be used.

12 Enable DSPI (DSPI) - Enables DSPI mode. All communication to the memory device is done using dual mode. If both the QSPI and DSPI bits are '1', then QSPI mode will be used.

11:8 Dummy cycles (DCYCLES) - Determines the amount of dummy cycles used after the address phase. If the DBYTE bit is set to '1' then this value won't be used and the amount of dummy cycles will always be 8.

7:0 Read instruction (READCMD) - Read instruction that the core will use for reading from the memory device.

* Writing to the configuration register is only enabled when the *reconf* VHDL generic is set.

129.3.2 Control register

*Table 2357.*0x04 - CTRL - Control register

31 7	6 5	4	3	2	1	0
RESERVED	DS	RST	CSN	EAS	IEN	USRC
0	0	0	1	0	0	0
r	r/rw*	rw	rw	rw	rw	rw

31 :5 RESERVED

6:5 Device Select (DS) - Controls which of the chip select signals the *CSN* bit controls. The chip select signals not currently selected by *Device select* are always driven to ‘1’.

*DS is only writable if the *multiple_csn* VHDL generic is 1.

4 Reset core (RST) - By writing ‘1’ to this bit the user can reset the core. This bit is automatically cleared when the core has been reset. Reset core should be used with care. Writing this bit has the same effect as system reset. Any ongoing transactions, both on AMBA and to the SPI device will be aborted.

3 Chip select (CSN) - Controls core chip select signal. This field always shows the level of the core’s internal chip select signal. This bit is always automatically set to ‘1’ when leaving User mode by writing USRC to ‘0’.

2 Enable Alternate Scaler (EAS) - When this bit is set the SPI clock is divided by using the alternate scaler.

Table 2357. 0x04 - CTRL - Control register

1	Interrupt Enable (IEN) - When this bit is set the core will generate an interrupt when a User mode transfer completes.
0	User control (USRC) - When this bit is set to ‘1’ the core will accept SPI data via the transmit register. Accesses to the memory mapped device area will return AMBA ERROR responses.

129.3.3 Status register

*Table 2358.*0x08 - STAT - Status register

31 9	8	7	6	5 3	2	1	0
RESERVED	MCSN	WRTR	RECONF	RESERVED	INIT	BUSY	DONE
0	-	-	-	0	0	0	0
r	r	r	r	r	r	r	wc

31:8 RESERVED

8 Multiple Chip Select (MCSN) - This bit is set to the value of the *multiple_csn* VHDL generic. When it is '1' the *DS* and *DCONF* register values are writeable and it is possible to configure the core to use all four chip select signals.

7 Allow writes reset (WRTR) - This bit is set to the value of the allow_writes VHDL generic. Determines the reset value of the WRT register bit.

6 Reconfigurable (RECONF) - This bit is set to the value of the *reconf* VHDL generic. When it is '1' it is possible to write to the configuration register.

5:3 RESERVED

2 Initialized (INIT) - This read only bit is set to '1' when the SPI memory device has been initialized. Accesses to the ROM area should only be performed when this bit is set to '1'.

1 Core busy (BUSY) - This bit is set to '1' when the core is performing an SPI operation.

0 Operation done (DONE) - This bit is set to '1' when the core has transferred an SPI command in user mode.

129.3.4 Receive register

*Table 2359.*0x0C - RX - Receive register

31 8	7 0
RESERVED	RDATA
0	nr
r	rw

31 :8 RESERVED

7:0 Receive data (RDATA) : Contains received data byte

Reset value: 0x000000UU, where U is undefined

129.3.5 Transmit register

*Table 2360.*0x10 - TX - Transmit register

31 8	7 0
RESERVED	TDATA
0	0
r	rw

31 :8 RESERVED

7:0 Transmit data (TDATA) - Data byte to transmit

129.3.6 Control register 2

*Table 2361.*0x1C - CTRL2 - Control register 2

31 5	4	3 0
RESERVED	WRT	DSCONF
0	-	0
r	rw	r/rw*

*Table 2361.*0x1C - CTRL2 - Control register 2

31 :5	RESERVED
4	Allow writes (WRT) - Determines if writes to the ROM area will be accepted and propagated to the SPI memory device. If set to '0', all writes to the ROM area will return an AHB_ERROR response instead. Has reset value as determined by the *allow_writes* VHDL generic.
3:0	Device Select Configuration (DSCONF) - Controls which chip select is used for accesses to the ROM area. For DSCONF values of 0-3, the chip select signal with the index of the same number as DSCONF will always be used. For DSCONF 4-15, the chip select signal is instead picked depending on the address using the formula *[14+DSCONF :13++DSCONF]*. I.e., with a DSCONF of 4 each SPI memory has a 128KiB address space, and with a DSCONF of 15 it instead is 256MiB. *DSCONF is only writable if the *multiple_csn* VHDL generic is 1.

129.4 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x045. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

129.5 Implementation

129.5.1 Reset

The core does not support *grlib_async_reset_enable*. All registers that react on the reset signal will have a synchronous reset.

129.5.2 Technology mapping

The core does not instantiate any technology specific primitives.

129.5.3 RAM usage

The core does not use any RAM components.

129.5.4 Endianness

The core automatically changes its endianness behavior depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for both big-endian and little-endian systems.

129.6 Configuration options

Table 2362 shows the configuration options of the core (VHDL generics).

*Table 2362.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB slave index	0 - (NAHBSLV-1)	0
hirq	Interrupt line	0 - (NAHBIRQ-1)	0
faddr	ADDR field of the AHB BAR1 defining ROM address space.	0 - 16#FFF#	16#000#
fmask	MASK field of the AHB BAR1 defining ROM address space.	0 - 16#FFF#	16#FFF#
ioaddr	ADDR field of the AHB BAR0 defining register address space.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR0 defining register space.	0 - 16#FFF#	16#FFF#

*Table 2362.*Configuration options

Generic name	Function	Allowed range	Default
spliten	If this generic is set to 1 the core will issue AMBA SPLIT responses when it is busy performing an operation on the memory device. Otherwise the core will insert wait states until the operation completes.	0 - 1	0
oepol	Select polarity of output enable signals. 0 = active low.	0 - 1	0
sdcard	Set to 0	0 - 0	0
readcmd	Read instruction of memory device	0 - 16#FF#	16#0B#
dummybyte	Use 8 dummy cycles after the address phase.	0 - 1	0
dualoutput	Use dual mode during the data phase for memory accesses in ESPI mode.	0 - 1	0
scaler	Clock divisor used when generating device clock is 2^{scaler}	1 - 512	1
altscaler	Clock divisor used when generating alternate device clock is $2^{altscaler}$	1 - 512	1
pwrupcnt	Unused	N/A	0
maxahbaccsz	Maximum supported AHB access size. The core will support accesses ranging from 8-bit (BYTE) to the size set by maxahbaccsz. The maximum access size is 256 bits (8WORD).	32, 64, 128, 256	AHBDW
offset	Specifies offset that will be added to incoming AMBA address before address is propagated to SPI flash device. An access to memory position *n* in the core's ROM area will be translated to an access to SPI memory device address *n* + *offset*. Note that this only applies to accesses to the ROM area, accesses via the core's register interface are unaffected.	-	0
quadoutput	Use quad mode during the data phase for memory accesses in ESPI mode.	0 - 1	0
dualinput	Use dual mode during the address phase for memory accesses in ESPI mode.	0 - 1	0
quadinput	Use quad mode during the address phase for memory accesses in ESPI mode.	0 - 1	0
dummycycles	Number of dummy cycles to be used after address phase. If *dummybyte* is set, then 8 cycles will always be used instead.	0 - 15	0
dspi	Enables DSPI mode, all communication to the memory device is done using dual mode. If both the QSPI and DSPI bits are '1', then QSPI mode will be used.	0 - 1	0
qspi	Enables QSPI mode, all communication to the memory device is done using quad mode. If both the QSPI and DSPI bits are '1', then QSPI mode will be used.	0 - 1	0
extaddr	Use extended address mode, using 32 bit address instead of 24 for memory reads. If set to 2, the RSTADDRM input signal is used to determine which mode should be selected on reset instead.	0 - 2	0
reconf	Enables writing to configuration register. Setting this bit to '1' roughly doubles the area required for the core.	0 - 1	0
writecmd	Write instruction of memory device	0 - 16#FF#	16#02#
allow_writes	Determines the reset value of the WRT register bit.	0 - 1	0
xip_byte	Send one additional byte of data after the address phase containing either 0x00 or 0xFF, depending on the value of *xip_polarity*.	0 - 1	0

*Table 2362.*Configuration options

Generic name	Function	Allowed range	Default
xip_polarity	Determines the bit value sent to the memory device during the dummy-cycles and xip-byte phase. Either all '0's or '1's.	0 - 1	1
multiple_csn	Determines if the *DS* and *DCONF* register values are writeable. When enabled it is possible to configure the core to use all four chip select signals.	0 - 1	0

129.7 Signal descriptions

Table 2363 shows the interface signals of the core (VHDL ports).

*Table 2363.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
SPII	MISO	Input	Master-input slave-output data line	-
	MOSI	Input	Master-output slave-input data line	-
	IO2	Input	IO2 data line	-
	IO3	Input	IO3 data line	-
	CD	Input	Unused	-
	RSTADDRM	Input	If the *extaddr* VHDL generic is set to 2, RSTADRM determines the reset value of the extended address configuration bit, otherwise signal is unused.	-
SPIO	MISO	Output	Master-input slave-output data line	-
	MOSI	Output	Master-output slave-input data line	-
	IO2	Output	IO2 data line	-
	IO3	Output	IO3 data line	-
	MISOOEN	Output	Master-input slave-output output enable	-
	MOSIOEN	Output	Master-output slave-input output enable	-
	IOOEN	Output	IO 2&3 output enable	-
	SCK	Output	SPI clock	-
	CSN	Output	Chip select 0	Low
	CSN1	Output	Chip select 1	Low
	CSN2	Output	Chip select 2	Low
	CSN3	Output	Chip select 3	Low
	CDCSNOEN	Output	Chip select output enable. This signal can be left unconnected and CSN should be connected to an output pad.	-
	READY	Output	When this signal is low the core is busy performing an operation.	High
	INITIALIZED	Output	This bit goes high when the SPI memory device has been initialized and can accept read accesses. This signal has the same value as the Initialized (INIT) bit in the core's Status register.	High

* see GRLIB IP Library User's Manual

129.8 Signal definitions and reset values

The signals and their reset values are described in table 2364.

*Table 2364.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
miso	Inout	Data into the master, can also serve as bi-directional signal if the core is configured to function with dual or quad memory devices.	-	Logical 1
mosi	Inout	Data out of the master, can also serve as bi-directional signal if the core is configured to function with dual or quad memory devices.	-	Logical 1
io2	Inout	Bi-directional signal used if the core is configured to work with quad memory devices.*	-	Logical 1
io3	Inout	Bi-directional signal used if the core is configured to work with quad memory devices.*	-	Logical 1
sck	Output	SPI clock	Logical 1	Logical 0
csn	Output	Chip select 0	Logical 0	Logical 1
csn1	Output	Chip select 1	Logical 0	Logical 1
csn2	Output	Chip select 2	Logical 0	Logical 1
csn3	Output	Chip select 3	Logical 0	Logical 1
ready	Output	Core ready	Logial 1	Logical 0
initialized	Output	Memory device initialized	Logical 1	Logical 0

* IO2&3 ports on the memory device often has other functionality triggered by an active low signal when they are not used for data transfers. To not accidentally trigger any of these functions the controller always outputs a ‘1’ on these signals in those cases.

129.9 Library dependencies

Table 2365 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2365.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	SPI	Component, signals	Component and signal definitions
GRLIB	AMBA	Signals	AMBA signal definitions

129.10 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.memctrl.all;

entity spimctrl_ex is
  port (
    clk       : in std_ulogic;
    rstn      : in std_ulogic;
    -- SPIMCTRL signals
    -- For SPI Flash
    spi_c     : out   std_ulogic;
```

```
    spi_do    : inout std_ulogic;
    spi_di    : inout std_ulogic;
    spi_io2   : inout std_ulogic;
    spi_io3   : inout std_ulogic;
    spi_sn    : out   std_ulogic
    );
end;

architecture rtl of spimctrl_ex is
  -- AMBA signals
  signal ahbsi  : ahb_slv_in_type;
  signal ahbso  : ahb_slv_out_vector := (others => ahbs_none);
  ...
  -- SPIMCTRL signals
  signal spmi : spimctrl_in_type;
  signal spmo : spimctrl_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- SPMCTRL core, configured for use with generic SPI Flash memory in 1-1-1 mode
  -- with read command 0x03 without any dummy cycles following the address.
  -- Writes to ROM area not allowed.
spimctrl0 : spimctrl
      generic map (hindex => 8, hirq => 10, faddr => 16#c00#, fmask => 16#f80#,
                   ioaddr => 16#900#, iomask => 16#fff#, spliten => CFG_SPLIT,
                   sdcard => 0, readcmd => 16#03#, dummybyte => 0,
                   scaler => 3, altscaler => 2,
                   dualoutput => 0, quadoutput => 0, dualinput => 0, quadinput => 0,
                   dummycycles => 0, DSPI => 0, QSPI => 0, extaddr => 0, reconf => 1,
                   writecmd => 16#00#, allow_writes => 0, xip_byte => 0, xip_polarity => 1)
      port map (rstn, clk, ahbsi, ahbso(8), spmi, spmo);

    spi_mosi_pad0 : iopad generic map (tech => padtech)
      port map (spi_do, spmo.mosi, spmo.mosioen, spmi.mosi);
    spi_miso_pad0 : iopad generic map (tech => padtech)
      port map (spi_di, spmo.miso, spmo.misooen, spmi.miso);

    spi_io2_pad0 : iopad generic map (tech => padtech)
      port map (spi_io2, spmo.io2, spmo.iooen, spmi.io2);
    spi_io3_pad0 : iopad generic map (tech => padtech)
      port map (spi_io3, spmo.io3, spmo.iooen, spmi.io3);

    spi_slvsel0_pad0 : outpad generic map (tech => padtech)
      port map (spi_sn, spmo.csn);
    spi_clk_pad0 : outpad generic map (tech => padtech)
      port map (spi_c, spmo.sck);
end;
```

130 SPIMASTER - SPI Master Device

130.1 Overview

The core provides a link between the AMBA APB bus and the Serial Peripheral Interface (SPI) bus and function as a SPI master. Core features include configurable word length (4, 5, 6... 32 bits), bit ordering and all four SPI modes are supported. This IP is developed as part of ESA activity: Prototyping of space protocol(s) for SPI (Contract: 4000114112/15/NL/LF).

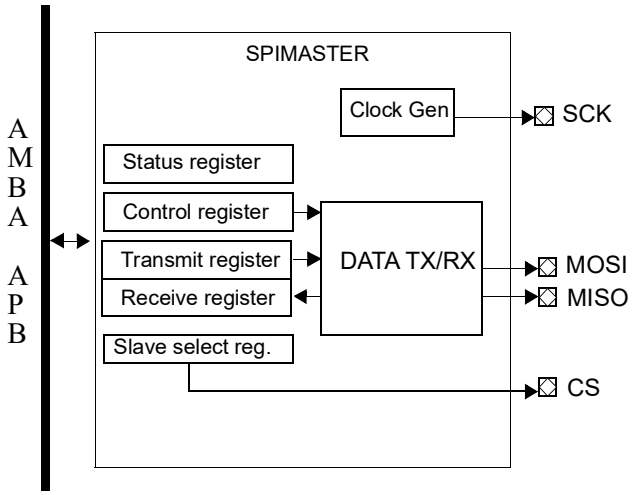


Figure 327. Block diagram

130.2 Transmission

The SPI bus is a full-duplex synchronous serial bus. Transmission starts when a master selects a slave through the slave's Slave Select (CS) signal and the clock line SCK transitions from its idle state. The clock line transition occurs when the EN bit in the Transmit Enable register is set to "high", the data need to be transferred must be written into the Transmit register before writing the transmit enable register. Data is transferred from the master through the Master-Output-Slave-Input (MOSI) signal and from the slave through the Master-Input-Slave-Output (MISO) signal. The SCK is only available during the data transfer and return to idle state after the transmission is completed.

During a transmission on the SPI bus data is either changed or read at a transition of SCK. If data has been read at edge n, data is changed at edge n+1. If data is read at the first transition of SCK the bus is said to have clock phase 0, and if data is changed at the first transition of SCK the bus has clock phase 1. The idle state of SCK may be either high or low. If the idle state of SCK is low, the bus has clock polarity 0 and if the idle state is high the clock polarity is 1. The combined values of clock polarity (CPOL) and clock phase (CPHA) determine the mode of the SPI bus. Figure below shows one byte (0x55) being transferred MSb first over the SPI bus under the four different modes. Note that the idle state of the MOSI line is '1' and that CPHA = 0 means that the devices must have data ready before the first transition of SCK. The figure does not include the MISO signal, the behavior of this line is the same as for the MOSI signal.

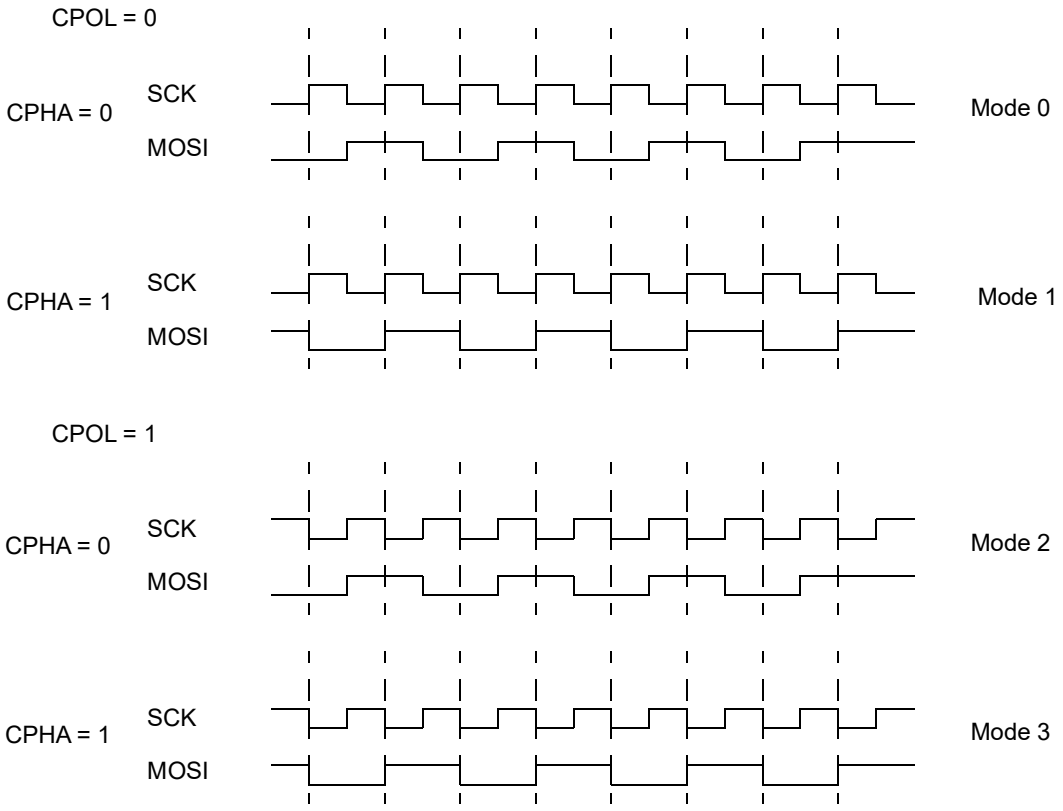


Figure 328. SPI transfer of byte 0x55 in all modes

130.3 Operation

The data transfer between this master and a slave device is performed using the registers mapped into APB address space. The data need to be transferred must be written into the Transmit register and the EN bit in the Transmit Enable register must be set “high” inorder to send and receive, the receive data is available when the transmission is completed. The EN bit will be set by the core to “low” when all the transmission is completed. It is also possible to generate an interrupt when all the transmission is completed using Transmission Done interrupt. The core’s control, polarity, chip select, clock registers must be appropriately set as per the requirements of the slave device. The SCK is only available during the data transfer and return to idle state after the transmission is completed.

The core can hold a data in addition to the one currently in transfer. The BC field in Status register specify the number of data available. If two words are written into the Transmit register the BC field show a value of 2, after the first transfer the value will be 1and finally when the transfer is completed the BC field will have value 0. The purpose of the hold register is to continuously transfer data to a slave device. In order to transfer continuously the software should perform the following, write two data to be transferred into the Transmit register and enable the transfer, after the first data is transferred (the software can learn about the data transfer either by polling the BC in Status register or using the RX Received data interrupt) the software can write the consecutive data into the Transmit register.

In order to receive a data the master should transmit a data. The Software should take care of clearing (reading) the data available in the received register before transmitting (receiving) an another data. The transmit buffer is maximum 32 bit (WLEN in Control register defines the Word length bit count), note also a transmit hold register is available which is also maximum 32 bit (also depends on the WLEN bit count). The Software controlling the master should take into account the response time of the slave (or by polling the EN bit in the Transmit Enable register, or by using the interrupt triggered

RX Received data interrupt) to determine when a transmission is completed, and also make sure the received data is read back before the next data is set to be transfered.

It is possible an overrun condition could occur if a received data is not read before the arrival of next data. If new data arrived before the Software could read the previously received data overrun condition is triggered. An Overrun interrupt is generated when overrun condition occurs for data reception.

130.4 Registers

The core is programmed through registers mapped into APB address space.

Table 2366. APB registers

APB address offset	Register
0x00	Control register
0x04	Status register
0x08	Clock divide register
0x0C	Chip select polarity register
0x10	Chip select register
0x14	Transmit register
0x18	Transmit enable register
0x1C	Receive register
0x20	Interrupt enable register
0x24	Interrupt register

130.4.1 Control Register

*Table 2367.*0x00 - CTRL - Control register

31 24	23-17	16	15-13	12 8	7	6	5	4	3	2	1	0
Key	R	OD	R	WLEN	IAMBA	CPHA	CPOL	REV	LOOP	RESET	R	R
0	0	0	0	0x0F	0	0	0	1	0	0	0	0
w	r	rw	r	rw	rw	rw	rw	rw	rw	rw	r	r

31 Safety code (KEY) - Must be 0x94 when writing, otherwise register write is ignored

23 : 17 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

16 Overrun detect (OD) - To detect overrun condition (also to trigger overrun interrupt) this bit must be enabled.

15 : 13 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

12 : 8 Word length (WLEN) - The value of this field determines the length in bits of a transfer on the SPI bus. Valid values are 0x03 to 0x1F

Word length is WLEN+1, allows words of length 4-32 bits.

7 AMBA Interrupt enable (IAMBA) - If set, AMBA interrupt generation is enabled for the events that are individually maskable by the Interrupt enable (INTE) register

6 Clock phase (CPHA) - When CPHA is '0' data will be read on the first transition of SCK. When CPHA is '1' data will be read on the second transition of SCK.

5 Clock polarity (CPOL) - Determines the polarity (idle state) of the SCK clock.

4 Reverse data (REV) - When this bit is '0' data is transmitted LSB first, when this bit is '1' data is transmitted MSB first.

3 Loop mode (LOOP) - When this bit is set, the core's transmitter and receiver are interconnected and the core will operate in loopback mode.

2 Reset (RESET) - Resets all the registers in the core.

1 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

0 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

130.4.2 Status Register

*Table 2368.*0x00 - CTRL - Control register

31 2	1 0
R	BC
	0
	r

31: 2 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

1:0 Buffer count BC - Transmit buffer count (maximum 2)

130.4.3 Clock Divide Register

*Table 2369.*0x08 - CDR - Clock divide register

31 24	23 0
Key	CD
0	0
w	rw

Table 2369. 0x08 - CDR - Clock divide register

31	Safety code (KEY) - Must be 0x94 when writing, otherwise register write is ignored
23 : 0	Clock divide (CD) - This value is used to divide the system clock and generate the SPI SCK clock. If the value is '0' then the system clock is divided by two and the SCK frequency is half the system clock. Similarly if the value is '1' the SCK frequency is 1/4 the system clock. if the value is '2' the SCK frequency is 1/6 the system clock. if the value is '3' the SCK frequency is 1/8 the system clock. if the value is '4' the SCK frequency is 1/10 the system clock.

130.4.4 Chip Select Polarity Register

Table 2370. 0x0C - CSP - Chip select polarity register

31 24	23 8	7 0
Key	R	CSP
0	0	0
w	r	rw

31 : 24	Safety code (KEY) - Must be 0x94 when writing, otherwise register write is ignored
23 : 8	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
7 : 0	Chip select polarity (CSP) - This register is provided in order to be able to communicate with devices having an active high chip select polarity. By default the polarity is active low.

130.4.5 Chip Select Register

Table 2371. 0x10 - CS - Chip select register

31 24	23 8	7 0
Key	R	CS
0	0	0xFFFFFF
w	r	rw

31 : 24	Safety code (KEY) - Must be 0x94 when writing, otherwise register write is ignored
23 : 8	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
7 : 0	Chip select (CS) - The slave device must be selected before starting communication. Software is solely responsible for activating the correct chip select signal, the core does not assert or deassert any chip select signal automatically.

130.4.6 Transmit Register

Table 2372. 0x14 - TDATA - Transmit register

31	0
TDATA	
0	
rw	

31 : 0 Transmit data (TDATA) - The written data is transferred to the slave device when appropriate conditions for CS and SCK are satisfied. The word to transmit should be written with its least significant bit at bit 0. Also note that only the number of bits need to be transferred from this register should match the word length register (WLEN).

130.4.7 Transmit Enable Register

Table 2373. 0x18 - TE - Transmit enable register

31	24	23	1	0
Key		R		EN
0		0		0
w		r		rw

31 : 24 Safety code (KEY) - Must be 0x94 when writing, otherwise register write is ignored

23 : 1 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

0 Enable (EN) - The written data in the Transmit register is transferred to the slave device when EN bit is set. This bit will go low when the transmission is completed.

130.4.8 Receive Register

Table 2374. 0x1C - RDATA - Receive register

31	0
RDATA	
0	
r	

31 : 0 Receive data (RDATA) - This register contains received data.

130.4.9 Interrupt Enable Register

Table 2375. 0x20 - INTE- Interrupt enable register

31	24	23	3	2	1	0
Key		RESERVED		OVRE	RXE	TXDE
0		0		0	0	0
w		r		rw	rw	rw

31 : 24 Safety code (KEY) - Must be 0x94 when writing, otherwise register write is ignored.

23 : 3 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

2 Overrun interrupt enable (OVRE) - If enabled an interrupt will be generated when overrun condition occurs for data reception.

1 Received data enable (RXE) - If enabled an interrupt is generated when a data is received.

0 Transmission done interrupt enable (TXDE) - If enabled an interrupt is generated when a data transmission is completed.

130.4.10Interrupt Register

*Table 2376.*0x24 - INT- Interrupt register

31 24	23 2	1	0
Key	RESERVED	OVR	RX	TXD
0	0	0	0	0
w	r	wc	wc	wc

31 : 24	Safety code (KEY) - Must be 0x94 when writing, otherwise register write is ignored.
23 : 1	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
2	Overrun interrupt (OVR) - An interrupt is generated when overrun condition occurs for data reception. (a received data must be read before the arrival of next data, if new data arrived before the Software could read the previously received data overrun condition is triggered)
1	Received data (RX) - An interrupt is generated when a data is received.
0	Transmission done (TXD) - Data Transmission completed (when the transmit and hold buffer is empty after a transmission).

130.5 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x0A6. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

130.6 Implementation

130.6.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

130.7 Configuration options

Table 2377 shows the configuration options of the core (VHDL generics).

*Table 2377.*Configuration options

Generic name	Function	Allowed range	Default
gSLVSEL	Number of slave select signals that the core will generate.	1 - 24	1
gPINDEX	APB slave index	0 - NAPBSLV-1	0
gHINDEX	Unused	0	0
gPADDR	ADDR field of the APB BAR	0 - 16#FFF#	0
gPMASK	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
gPIRQ	Interrupt line driven by APB interface.	0 - NAHBIRQ-1	1

130.8 Signal descriptions

Table 2378 shows the interface signals of the core (VHDL ports).

*Table 2378.*Signal descriptions

Signal name	Type	Function	Active
RSTN	Input	Reset	Low
CLK	Input	Clock	-
APBI	Input	APB slave input signals*	-
APBO	Output	APB slave output signals*	-
MISO	Input	SPI data line input	-
MOSI	Output	SPI data line output	-
SCK	Output	SPI clock line output	-
SLVSEL	Output	Slave select output(s). The range of the vector is (gSLVSEL-1):0	Low

* see GRLIB IP Library User's Manual

130.9 Library dependencies

Table 2379 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2379.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	SPICOMP	Component, signals	Component declaration, SPI signal definitions
GAISLER	GRSPICOMP	Component, signals	Component declaration, SPI signal definitions

130.10 Instantiation

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.devices.all;
use grlib.stdlib.all;
library gaisler;
use gaisler.spicomp.all;
use gaisler.grspicomp.all;

entity grspimaster_ex is
   generic (
      gSLVSEL    :    integer range 1 to 24 := 8);
   port (
     rstn        :    in  std_ulogic;
     clk         :    in  std_ulogic;
     apbi        :    in  APB_Slv_In_Type;
     apbo        :    out APB_Slv_Out_Type;
     -- SPI signals
     miso        :    in  std_ulogic;
     mosi        :    out std_ulogic;
     sck         :    out std_ulogic;
     slvsel      :    out std_logic_vector((gSLVSEL-1) downto 0));
end entity grspimaster_ex;

architecture rtl of grspimaster_ex is
  -- AMBA signals
```

```
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector;

  signal sck_mst        :       std_ulogic;
  signal miso_mst       :       std_ulogic;
  signal mosi_mst       :       std_ulogic;
  signal slvsel_mst     :       std_logic_vector(7 downto 0);

begin

   grspimaster0: grspimaster
      generic map (
         gSLVSEL         => gSLVSEL,
         gPINDEX         => 1,
         gPADDR          => 1,
         gPIRQ           => 1)
      port map (
        rstn             => rstn,
        clk              => clk,
        apbi             => apbi,
        apbo             => apbo(1),

        miso             => miso_mst,
        mosi             => mosi_mst,
        sck              => sck_mst,
        slvsel           => slvsel_mst);

   sck_m1pad: outpad generic map (tech => padtech)
      port map (sck, sck_mst);

   mosi_m1pad: outpad generic map (tech => padtech)
      port map (mosi, mosi_mst);

   miso_m1pad: inpad generic map (tech => padtech)
      port map (miso, miso_mst);

   cs_m1pad: outpadv generic map (width => 8, tech => padtech)
      port map (slvsel, slvsel_mst);

end architecture rtl;
```


Frontgrade Gaisler AB
Kungsgatan | SE-411 19 | Goteborg | Sweden
+46 31 7758650 | frontgrade.com/gaisler

131 SPISLAVE - Dual Port SPI Slave

131.1 Overview

This core is a Dual Port SPI Slave device that provides link between SPI and AMBA AHB and APB ports. Core features include configurable word length (4, 5, 6 ... 32 bits), bit ordering and all four SPI modes are supported. This core also has redundant SPI ports which can be interfaced using two different masters. The slave takes two sets of SPI interfaces (nominal and redundant each consists of two data signals, one clock signal and one chip select signal). This IP is developed as part of ESA activity: Prototyping of space protocol(s) for SPI (Contract: 4000114112/15/NL/LF).

131.2 Implementation of SPI protocols

This slave device is based on the protocols developed for SPI4SPACE activity. The following text explains the different protocols supported.

In order to support the SPI 0 protocol the slave provides configurable word length of 4, 5, 6 ... 32 bits transmission and reception. The Word bit ordering can be MSB first or LSB first transferred.

For SPI 1 protocol the word length of the transfer can be 8, 16 or 24 bits. The word bit ordering MSB transferred first and LSB transferred last is supported. The parity bit can be appended at the end of every word, the parity bit is not included by the SPI slave device, since the implementation supports 9, 17 and 25 bits of word transfer the parity bit can be appended by the software.

All control and data transfer for SPI protocol 0 and 1 are supported only through the APB registers.

The SPI protocol 2 uses a fixed word length of 16 bits. The word bit ordering is MSB transferred first and LSB transferred last. Also this core implements the network layer of the SPI protocol 2, the slave hardware itself can process the SPI protocol 2 commands and provide responses. The APB interface is only for control and status, all the data transfer to the AMBA is performed using the AHB Master.

:

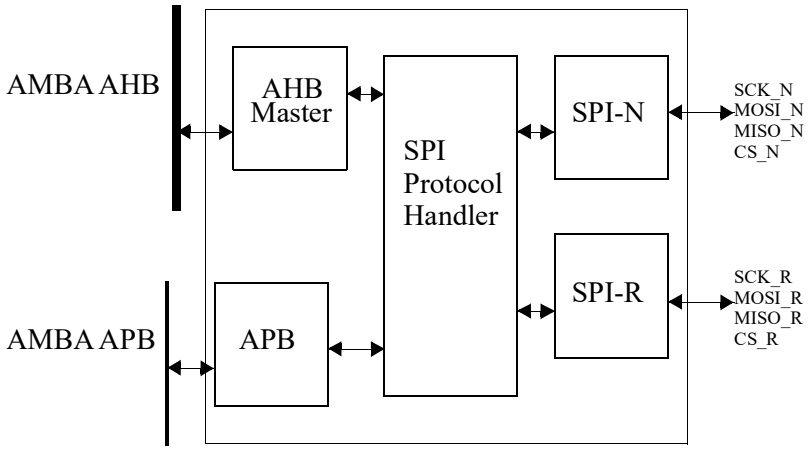


Figure 329. Block diagram

131.3 Transmission

The SPI bus is a full-duplex synchronous serial bus. Transmission starts when a master selects a slave through the slave's Slave Select (CS) signal and the clock line SCK transitions from its idle state. Data is transferred from the master through the Master-Output-Slave-Input (MOSI) signal and from the slave through the Master-Input-Slave-Output (MISO) signal. In some systems with only one master and one slave, the Slave Select input of the slave may be always active and the master does not need

to have a slave select output. This does not apply to this device, the slave select signal must be used to mark the start and end of an operation.

During a transmission on the SPI bus data is either changed or read at a transition of SCK. If data has been read at edge n, data is changed at edge n+1. If data is read at the first transition of SCK the bus is said to have clock phase 0, and if data is changed at the first transition of SCK the bus has clock phase 1. The idle state of SCK may be either high or low. If the idle state of SCK is low, the bus has clock polarity 0 and if the idle state is high the clock polarity is 1. The combined values of clock polarity (CPOL) and clock phase (CPHA) determine the mode of the SPI bus. Figure below shows one byte (0x55) being transferred MSb first over the SPI bus under the four different modes. Note that the idle state of the MOSI line is '1' and that CPHA = 0 means that the devices must have data ready before the first transition of SCK. The figure does not include the MISO signal, the behavior of this line is the same as for the MOSI signal.

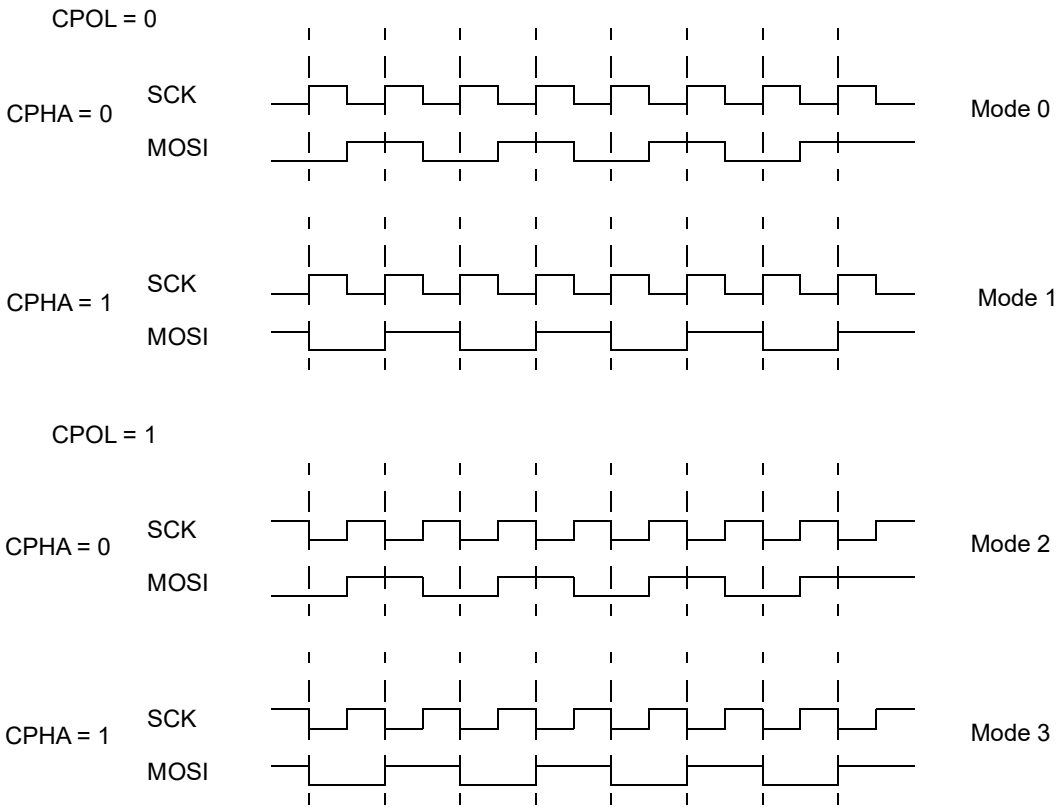


Figure 330. SPI transfer of byte 0x55 in all modes

131.4 Operation

The data transfer between the master and the slave is through APB registers or through command transfer from a master is determined by the EN bit in the SPI2 control register. When APB registers are used the data transferred by a master is available at receive registers (NRDATA or RRDATA depending on the port used) while during the same reception period the contents of the transmit registers (TDATA) are transferred to the master. When appropriate commands are transferred by a master SPI device and EN bit in the SPI2 control register is enabled then the commands are processed by the SPI 2 protocol handler available in this core. The SPI protocol 2 implementation is explained in detail in the following section.

131.5 SPI 2 Protocol Handler

The core is capable of handling the commands (based on SPI protocol 2) transferred by a SPI master and provide response. The message format transferred between a SPI master and SPI slave device is defined below.

Table 2380. Example message format (write data)

Signal	*Message Header*		*Payload*	*Payload CRC*
MOSI	*Command #1*	*Command #2*	*Data*	*CRC-16*
MISO	*Response #1*	*Response #2*	*0x0000*	*0x0000*

Table 2381. Example message format (read data)

Signal	*Message Header*		*Payload*	*Payload CRC*
MOSI	*Command #1*	*Command #2*	*0x0000*	*0x0000*
MISO	*Response #1*	*Response #2*	*Data*	*CRC-16*

The message header is composed of a Command token from the master and a Response token from the slave. The message also contains optional data words and CRC checksum appended at the end that are calculated for the data words transferred. The CRC is mandatory, if the message contains payload data then the message is always appended with one word of CRC. The received messages are processed by the SPI slave device and response and data are transferred as per the received command. Also note some of the status bits in the response token are status for the previously received command.

The SPI slave has the possibility to address incoming data with clock gaps (splitted) fashion. If the SPI master transfers the data with a clock gap the slave can accept the data and provide proper response. For example if the SPI master is software controlled and has a SPI controller with a word-width that is less than the full message then software needs to keep at least one word in the transmit queue at all times to avoid breaking the protocol. In order to provide a relaxed requirement on software, the SPI2 protocol allow gaps between clock periods (equivalent to stretching a clock period). The gaps must be at word level (16 bits) i.e. the clock gap can be between Command #1 and Command #2 not within the Command #1 16 bits.

131.5.1 Message Header - Command Token

The master transmits a message header that specifies the action need to be performed in slave. The command token sent by the master device consist of two 16 bit words. The message header content details are explained below.

Table 2382. Command word 1

MSB	*Command Token Word #1*															*LSB*
Prefix		*Command Code*						*Spare*		*Message Length*						
15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*	
'0'	*'1'*	*C5*	*C4*	*C3*	*C2*	*C1*	*C0*	*'1'*	*'1'*	*L5*	*L4*	*L3*	*L2*	*L1*	*L0*	

Table 2383. Command word 2

MSB	*Command Token Word #2*															*LSB*
Prefix		*Sub-address*								*Spare*		*CRC-4*				
31	*30*	*29*	*28*	*27*	*26*	*25*	*24*	*23*	*22*	*21*	*20*	*19*	*18*	*17*	*16*	

'0'	*'1'*	*SA7*	*SA6*	*SA5*	*SA4*	*SA3*	*SA2*	*SA1*	*SA0*	*'1'*	*'1'*	*CRC3*	*CRC 2*	*CRC1*	*CRC0*

- Prefix and spare

The prefix bits are transmitted initially. In the command word#1 and #2 the prefix and spare bits have fixed value in the current implementation, these are reserved bits. The spislave receives them and use it for validating the token. If an invalid prefix and spare bits are received then Status illegal command (SIC) status bit is enabled and also transmitted to master as part of the next response token.

- Message Length

The number of payload words that will be transmitted in the current message. The number should not include the command token and the CRC checksum appended at the end of the message.

- Sub-address

This field provide additional sub-address location for write and read commands.

- CRC-4

The final four bits of the command token consist of a checksum for all the previous command token bits transmitted in this message. The CRC-4 should be computed for the following 28 bits, Command word #1 (16bits, MSB first sent to the CRC generator) and Command word #2 (excluding this CRC-4 field) (12bits, MSB first sent to the CRC generator). The Prefix and Spare fields are included in the CRC calculation. The generator polynomial used is $X^4 + X + 1$. In this SPI slave receiving end the CRC-4 is calculated internally for the received command token, if the calculated CRC-4 does not match the expected value (this field) the corresponding command token is discarded and message error status is enabled and transmitted to master as part of the next response token.

- Payload data

The payload consist of the data need to be transferred from the master to slave. Depending on the command executed the master must include valid data or dummy information in the form of string of zeros. For example the write command have the data to be written as the payload but the read command have dummy information in the form of string of zeros.

- Payload CRC

When a valid payload is delivered in the payload data section of the message a Payload CRC (CRC-16) must be included at the end of the message. The generator polynomial used is $x^{16} + x^{15} + x^2 + 1$. When dummy information in the form of string of zeros is included then no Payload CRC need to be attached then the payload CRC field must be all zero (0X0000). In the SPI slave receiving end the CRC-16 is calculated internally for the received payload data, if the calculated CRC-16 does not match the expected value (this field) the corresponding operation with respect to the command is not performed and message error status is enabled and transmitted to master as part of the next response token.

131.5.2 Command Code

The command code specify the operating instruction for the receiving slave. The detailed explanation of each command code and its implementation are explained in the table below.

- Reset command

Code	*Command*	*Length*	*Sub-address*	*Payload*	*Description*
0x00	*RESET_SPI*	*0x00*	*0x00*	*None*	*This command will reset all the spi slave device registers to the default value except the time registers (TIME1, TIME2) and core enable registers (ENN and ENR).*

The RESET_SPI command resets the SPI slave device to a power up initialized state. The SPI slave resets the system only if it received a valid command. If the prefix and spare bits does not match or if the calculated CRC-4 does not match the expected value then the RESET_SPI command is discarded.

- Time synchronization command

Code	*Command*	*Length*	*Sub-address*	*Payload*	*Description*
0x07	*SYNCH*	*0x04*	*0x00*	*MOSI:<SYNC1><SYNC2><SYNC3><SYNC4><CRC-16>* *MISO: <all zeros>*	*The master must transmit the SYNC command token followed by payload words containing synchronization information for it. These words are copied inside dedicated registers implemented in the SPI slave device after validation.*

The time register is of 64 bit in width, the most significant time is transferred first in SYNC1 followed by SYNC2, SYNC3 and SYNC4. The time register roll over when its maximum count is reached. The time is synchronised only when all the words are received and also the command token CRC-4 and data CRC-16 must be valid. All bits are zero at reset. The RESET_SPI does not reset the time register.

- Time increment command

Code	*Command*	*Length*	*Sub-address*	*Payload*	*Description*
0x08	*TICK*	*0x00*	*Used as index for increment.*	*None*	*This command is used to advance the timing synchronization register available in the SPI slave device (same register used for the SYNC command).*

A valid command increments the implemented time register. The sub address field specify from which bit the time register must increment.

- Read back sent command

Code	*Command*	*Length*	*Sub-address*	*Payload*	*Description*
0x0A	*READBACK_CMD*	*0x02*	*0x00*	*MOSI: <all zeros>* *MISO:<CMDTOKEN><CRC-16>*	*The command can be used to verify the correct reception of the previous command. Upon reception of the command the slave respond with the previous command token.*

The SPI slave device after receiving the READBACK_CMD send the previous command token transmitted by the SPI master. This command is useful only when some other command (other than RESET_SPI) was previously transmitted. If the previous command is RESET_SPI, the SPI master only receives zeros in the payload section of this command.

- Write command

Code	*Command*	*Length*	*Sub-address*	*Payload*	*Description*
0x0D	*WRITE_SA*	*Number of words to be written, N*	*SA*	*MOSI:<DW1> <DW2> ...<DWN> <CRC-16> MISO: <all zeros>*	*The command is used to write a certain number of data words into a slave specific Sub Address. Dedicated field of the command token select the payload length and the target SA.*

When a valid WRITE_SA command is received the payload data is stored at the address specified. The address for writing the data is calculated by using the write address register (CONFIG_WRITE), and sub-address. The CRC-16 is calculated for received words and compared with the received payload CRC, if a data CRC error is detected then message error status is enabled and transmitted to master as part of the next response token.

- Read command

Code	*Command*	*Length*	*Sub-address*	*Payload*	*Description*
0x0E	*READ_SA*	*Number of words to be read, N*	*SA*	*MOSI: <all zeros> MISO:<DW1> <DW2> ...<DWN> <CRC-16>*	*The command is used to read a certain number of data words into a slave specific Sub Address. Dedicated field of the command token select the payload length and the target SA.*

When a valid READ_SA command is received the payload data is transferred from the address specified. The address for reading the data is calculated by using the read address register (CONFIG_READ), and sub-address. The CRC-16 is calculated for transmitted words and sent as payload CRC by the slave device.

- Configure address commands

Code	*Command*	*Length*	*Sub-address*	*Payload*	*Description*
0x20	*CONFIGWRITE_ADDR*	*0x02*	*0x00*	*MOSI:<CW1><CW2> <CRC-16> MISO: <all zeros>*	*The command can be used to notify the slave about the address to which the data from the master is written, used for WRITE_SA command.*
0x21	*CONFIG READ_ADDR*	*0x02*	*0x00*	*MOSI:<CR1><CR2> <CRC-16> MISO: <all zeros>*	*The command can be used to notify the slave about the address from which the data to the master is read, used for READ_SA command.*

Dedicated registers for write address and read address is implemented, these registers takes value from this command. The purpose of this register is to access upto 32 bits of address space. The most significant word CW1 (or CR1) contains the most significant bytes of the target address.

- Redundancy commands

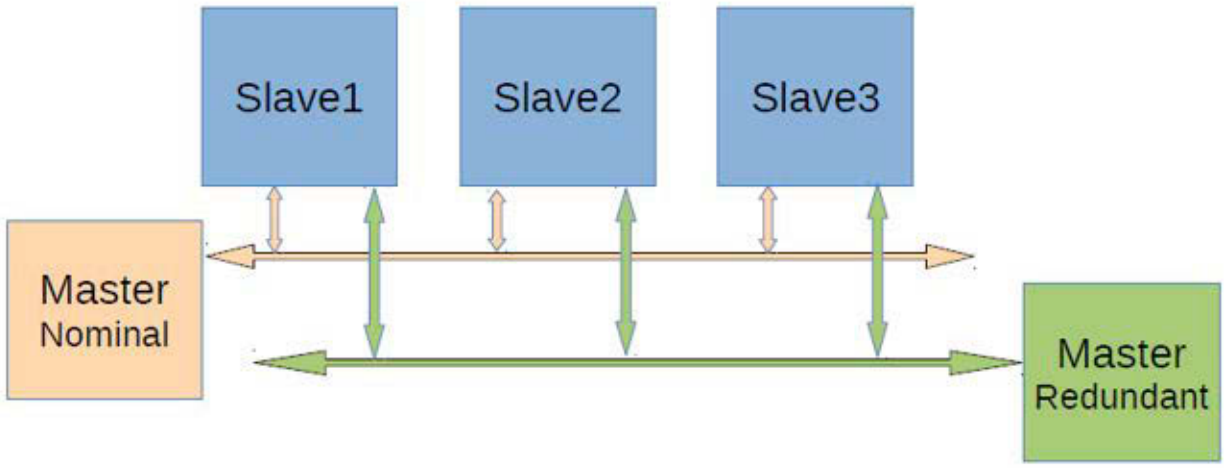


Figure 331. Redundant system

Code	*Command*	*Length*	*Sub-address*	*Payload*	*Description*
0x24	*ACTIVATE*	*0x00*	*0x00*	*None*	*The command is used to activate the other slave interface. This command cannot activate the interface in which it is receiving this command.*

Code	*Command*	*Length*	*Sub-address*	*Payload*	*Description*
0x25	*DEACTIVATE*	*0x00*	*0x00*	*None*	*The command is used to deactivate the other slave interface. This command cannot deactivate the interface in which it is receiving this command.*

The SPI Slave device has two dedicated interfaces for two masters. The masters can send to its corresponding slave interface to activate or deactivate the other SPI interface. The master device cannot activate or deactivate the same ports on which it is connected, it can only activate or deactivate the other ports.

Initially both the SPI port interfaces are enabled to receive commands, when the communication between the nominal master and slave interface fails then the redundant master can deactivate the nominal interface using its dedicated redundant interface. The redundant master can also activate the nominal interface.

An example switchover scenario from nominal to redundant interface is described in the following text.

The nominal master communicates with its dedicated interface to a slave device, a fault occurred can be detected by the master using several options,

The status received by the master have invalid values (using the response token),

The Read back command sent does not provide appropriate values in the received payload,

Error bits are enabled in the status received by the master (using the response token),

Based on any of the above mentioned fault detection methods the master can send deactivate command in the redundant interface to deactivate the nominal interface of the slave. The master can send Read back sent commands (using redundant) to check if the previous deactivate command was received by the slave and can check the status of the response token as well. After conforming a proper communication has been established the master can use the redundant interface to perform its normal operations.

- Others

Code	*Command*	*Length*	*Sub-address*	*Payload*	*Description*
All others	*N/A*	*0x00*			*These command codes are currently not implemented and their use is reserved for future. Upon reception of one of these codes the slave discard the incoming data.*

Any other commands which are not implemented is received then the command token is discarded and Status illegal command (SIC) bit is enabled and also transmitted to master as part of the next response token.

131.5.3 Message Header -Response Token

The slave transmits a message header which consist of status of module and details of error occurred. The message header sent by SPI slave device is called response token which consist of two 16 bit words. The message header content details are explained below.

MSB	*Response Token Word #1*														*LSB*
Prefix		*Status*					*Spare*					*Module State*			
15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
'1'	*'0'*	*STF*	*ME*	*AR*	*IC*	*'0'*	*'0'*	*'0'*	*'0'*	*'0'*	*'0'*	*MS3*	*MS2*	*MS1*	*MS0*

Table 2384. Response word #1

MSB	*Response Token Word #2*														*LSB*
Prefix		*Data*	*Spare*									*CRC-4*			
31	*30*	*29*	*28*	*27*	*26*	*25*	*24*	*23*	*22*	*21*	*20*	*19*	*18*	*17*	*16*
'1'	*'0'*	*DNA*	*'0'*	*'0'*	*'1'*	*'1'*	*'1 '*	*'1'*	*'0'*	*'0'*	*'0'*	*CRC 3*	*CRC 2*	*CRC1*	*CRC0*

Table 2385. Response word #2

- Status bits

Bit	*Identifier*	*Type*	*Value*	*Description*	*Clear Condition*
13	*SPI_TERMINAL_FAULT*	*Error*	*'0'=no fault* *'1'=fault*	*The bit flag a SPI terminal fault condition.*	*According to the module current state.*

In SPI slave device this bit is enabled or disabled by SPI2 control register (STF) using APB.

Bit	Identifier	Type	Value	Description	Clear Condition
12	MESSAGE_ERROR	Error	'0'=no fault '1'=fault	This bit is utilized to indicate that the previous message received from the bus master has failed to pass the validity tests.	Always related to the previous command. Reception of a valid command will clear it (with a delay of one command).

This status bit is enabled when the received message fails to pass the command token and payload data CRC checks. The next valid command clears this status bit.

Bit	Identifier	Type	Value	Description	Clear Condition
11	ADDRESS_ERROR	Error	'0'=no fault '1'=fault	This bit flag an AMBA error occurred while performing the previous command.	Always related to the previous command. Reception of a valid command will clear it (with a delay of one command)

The SPI slave device uses an AHB master to perform the memory read and write, this bit is enabled when an AHB error is reported.

Bit	Identifier	Type	Value	Description	Clear Condition
10	ILLEGAL_COMMAND	Error	'0'=no fault '1'=fault	This bit flag that the previous received command was not compatible with the SPI slave device.	Always related to the previous command. Reception of a valid command will clear it (with a delay of one command)

When the prefix and spare bits in the received command token do not match the intended value or an unimplemented command is received this status bit is enabled. The next valid command clears this status bit.

- Module state:

In the SPI slave device these bits are enabled or disabled by SPI2 control register (MODSTAT) using APB. These bits can be used by Software controlling the slave device to provide additional status to the master.

131.6 Redundancy

The SPI slave has a two SPI ports which can be interfaced using two different masters. Two SPI master capable of communicating individually to the respective port must be available inorder to achieve redundancy using this Dual-port SPI slave.The slave takes two sets of SPI interfaces (nominal and

redundant). The configuration registers available in the device is used to enable which interface to communicate and it is possible to use dedicated commands (using SPI 2 protocol) to activate and deactivate ports.

While using configuration registers to activate or deactivate ports, the complete control of activation and deactivation must be performed by the external unit, only one port must active at any time. If both enabled then both the SPI ports are open for communication which is not supported while using external configuration for redundancy. The system which initiates the communication should take responsibility for which lane to take (there must be dedicated SPI Masters available in the system to communicate with the respective slave). If both are disabled then no communication is possible. The Master (driver) must have two dedicated SPI Master to perform communication on each lane of the SPI bus.

When commands are used to control the ports, the device can receive commands from both the interfaces. By receiving from both the interfaces the slave device can deactivate a non-working interface. The intention is to keep only one bus active for normal operation but using the redundant bus to achieve switchover. The SPI protocol 2 implementation supports dedicated commands to achieve the activation and deactivation of interfaces.

In normal working case the SPI masters Nominal and redundant (using HW or SW) should make sure not to write at the same time to both lanes of dual-port SPI Slave (for example to make a transfer). The SW or HW can command the Redundant master only when it detects problem with the Nominal communication. For worst case lets say, the SPI masters Nominal (in error state babelling some command repeatedly), using the redundant port the Nominal lane can be switched off (switch over command using redundant port or external configuration), the slave takes the redundant port input, the SPI slave is designed to take the redundant port inputs when it is available rather than nominal input.

131.7 Registers

The core is programmed through registers mapped into APB address space.

Table 2386. APB registers

APB address offset	Register
0x00	Control register
0x04	Status register
0x08	Transmit register
0x0C	Nominal receive register
0x10	Redundant receive register
0x14	Interrupt enable register
0x18	Interrupt register
0x1C	Reserved
0x20	SPI2 control register
0x24	SPI2 time1 register
0x28	SPI2 time2 register
0x2C	SPI2 config address write register
0x30	SPI2 config address read register

131.7.1 Control Register

*Table 2387.*0x00 - CTRL - Control register

31 24	23-17	16	15-13	12 8	7	6	5	4	3	2	1	0
Key	R	OD	R	WLEN	IAMBA	CPHA	CPOL	REV	R	RESET	ENR	ENN
0	0	0	0	0x0F	0	0	0	1	0	0	1	1
w	r	rw	r	rw	rw	rw	rw	rw	r	rw	rw	rw

31	Safety code (KEY) - Must be 0x68 when writing, otherwise register write is ignored
23 : 17	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
16	Overrun detect (OD) - To detect overrun condition (also to trigger overrun interrupt) this bit must be enabled. Valid only for SPI protocol 0 and 1.
15 : 13	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
12 : 8	Word length (WLEN) - The value of this field determines the length in bits of a transfer on the SPI bus. Valid values are 0x03 to 0x1F Word length is WLEN+1, allows words of length 4-32 bits.
7	AMBA Interrupt enable (IAMBA) - If set, AMBA interrupt generation is enabled for the events that are individually maskable by the Interrupt enable (INTE) register
6	Clock phase (CPHA) - When CPHA is ‘0’ data will be read on the first transition of SCK. When CPHA is ‘1’ data will be read on the second transition of SCK.
5	Clock polarity (CPOL) - Determines the polarity (idle state) of the SCK clock.
4	Reverse data (REV) - When this bit is ‘0’ data is transmitted LSB first, when this bit is ‘1’ data is transmitted MSB first.
3	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
2	Reset (RESET) - Resets all the registers in the core except time registers (TIME1, TIME2) and core enable registers (ENN and ENR).
1	Enable redundant port transfer (ENR) - Enable bit for redundant port transfer. See section 5.6 for more information.
0	Enable nominal port transfer (ENN)- Enable bit for nominal port transfer. See section 5.6 for more information.

131.7.2 Status Register

Table 2388. 0x04 - STAT - Status register

31 8	7	6	5	4	3 2	1	0
RESERVED	ATR	ATN	SAR	SIC	R	RR	RN
0	0	1	0	0	0	0	0
r	r	r	r	r	r	r	r

31 : 3 RESERVED

7 Active transmission in redundant port (ATR) - This bit provides the status of the redundant transmission port. Set based on the incoming activate and deactivate commands (active '1' else '0'). Valid only for SPI protocol 2 implementation.

6 Active transmission in nominal port (ATN) - This bit provides the status of the nominal transmission port. Set based on the incoming activate and deactivate commands (active '1' else '0'). Valid only for SPI protocol 2 implementation.

5 Status address error (SAR) - This bit gets set to '1' when an AMBA write or read access resulted in a error. A valid new command clears this status bit. Valid only for SPI protocol 2 implementation.

4 Status illegal command (SIC) - This bit gets set to '1' when an illegal command is received. A valid new command clears this status bit. Valid only for SPI protocol 2 implementation.

3 : 2 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

1 Received data nominal (RR) - This bit gets set to '1' each time a data is received in the redundant port. The bit gets set to '0' when the Redundant receive register is read.

0 Received data nominal (RN) - This bit gets set to '1' each time a data is received in the nominal port. The bit gets set to '0' when the Nominal receive register is read.

131.7.3 Transmit Register

Table 2389. 0x08 - TDATA - Transmit register

31 0
TDATA
0
rw

31 : 0 Transmit data (TDATA) - The written data is transferred to the master device when appropriate conditions for CS and SCK are satisfied. The word to transmit should be written with its least significant bit at bit 0. Also note that only the number of bits need to be transferred from this register should match the word length register (WLEN). Valid only for SPI protocol 0 and 1.

131.7.4 Nominal Receive Register

Table 2390. 0x0C - NRDATA - Nominal receive register

31 0
NRDATA
0
r

31 : 0 Nominal Receive data (NRDATA) - This register contains received data from the nominal port. Valid only for SPI protocol 0 and 1.

131.7.5 Redundant Receive Register

Table 2391. 0x10 - RRDATA - Redundant receive register

31 0
RRDATA
0

*Table 2391.*0x10 - RRDATA - Redundant receive register

r

31 : 0 Redundant Receive data (RRDATA) - This register contains received data from the redundant port. Valid only for SPI protocol 0 and 1.

131.7.6 Interrupt Enable Register

*Table 2392.*0x14 - INTE- Interrupt enable register

31 24	23 9	8	7	6	5	4	3	2	1	0
Key	RESERVED	OVRE	WDE	AE	CRE	CWE	TICKE	SYNCE	RXRE	RXNE
0	0	0	0	0	0	0	0	0	0	0
w	r	rw	rw	rw	rw	rw	rw	rw	rw	rw

31 : 24 Safety code (KEY) - Must be 0x68 when writing, otherwise register write is ignored.

23 : 9 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

8 Overrun interrupt enable (OVRE) - If enabled an interrupt will be generated when overrun condition occurs for data reception. Valid only for SPI protocol 0 and 1.

7 Write data interrupt enable (WDE). Valid only for SPI protocol 2.

6 AMBA access error interrupt enable (AE). Valid only for SPI protocol 2.

5 Change in config read address interrupt enable (CRE). Valid only for SPI protocol 2.

4 Change in config write address interrupt enable (CWE). Valid only for SPI protocol 2.

3 Tick command received interrupt enable (TICKE). Valid only for SPI protocol 2.

2 Sync command received interrupt enable (SYNCE). Valid only for SPI protocol 2.

1 Data received in redundant port interrupt enable (RXRE).Valid only for SPI protocol 0 and 1.

0 Data received in nominal port interrupt enable (RXNE). Valid only for SPI protocol 0 and 1.

131.7.7 Interrupt Register

*Table 2393.*0x18- INT- Interrupt register

31 24	23 9	8	7	6	5	4	3	2	1	0
Key	RESERVED	OVR	WD	AI	CR	CW	TICK	SYNC	RXR	RXN
0	0	0	0	0	0	0	0	0	0	0
w	r	wc	wc	wc	wc	wc	wc	wc	wc	wc

31 : 24 Safety code (KEY) - Must be 0x68 when writing, otherwise register write is ignored.

23 : 9 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

8 Overrun interrupt (OVR) - An interrupt is generated when overrun condition occurs for data reception. (a received data must be read before the arrival of next data, if new data arrived before the Software could read the previously received data overrun condition is triggered). Valid only for SPI protocol 0 and 1.

7 Write data interrupt (WD). Valid only for SPI protocol 2.

6 AMBA access error interrupt (AI). Valid only for SPI protocol 2.

5 Change in config read address interrupt (CR). Valid only for SPI protocol 2.

4 Change in config write address interrupt (CW). Valid only for SPI protocol 2.

3 Tick command received interrupt (TICK). Valid only for SPI protocol 2.

2 Sync command received interrupt (SYNC). Valid only for SPI protocol 2.

1 Data received in redundant port interrupt (RXR).Valid only for SPI protocol 0 and 1.

0 Data received in nominal port interrupt (RXN). Valid only for SPI protocol 0 and 1.

131.7.8 SPI2 Control Register

*Table 2394.*0x20- SPI2C- SPI2 control register

31 24	23 8	7	6	5	4	3	2	1	0
Key	RESERVED	MODSTAT				RESERVED		STF	EN
0	0	0	0	0	0	0	0	0	1
w	r	rw	rw	rw	rw	r	r	rw	rw

31 : 24 Safety code (KEY) - Must be 0x68 when writing, otherwise register write is ignored.

23 : 8 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

7: 4 Module state (MODSTAT). The values in these bits are sent to the master via the response token. These are user configurable registers which can be set to ‘1’ or ‘0’. Valid only for SPI protocol 2.

3: 2 RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

1 SPI terminal failure (STF). This value in this bit is sent to the master via the response token. In order to intimate a terminal failure this bit can be written to ‘1’ through software. Valid only for SPI protocol 2.

0 Enable (EN). SPI protocol 2 enable bit. If set to ‘1’ the commands received from master are handled by the SPI 2 protocol handler in the core. If set to ‘0’ the data received and transfered are using the APB registers.

131.7.9 SPI2 Time1 Register

*Table 2395.*0x24 - TIME1 - SPI2 time1 register

31 0
TIME1
0x00000000
r

31 : 0 Time 1 register (TIME1) - Provides the most significant 32 bits of the time register. This is a status (read only) register, the contents of this register is a reflection of the time modified/incremented using the sync and tick command respectively.

131.7.10SPI2 Time2 Register

*Table 2396.*0x28 - TIME2 - SPI2 time2 register

31	0
TIME2	
0x00000000	
r	

31 : 0 Time 2 register (TIME2) - Provides the lower 32 bits of the time register. This is a status (read only) register, the contents of this register is a reflection of the time modified/incremented using the sync and tick command respectively.

131.7.11SPI2 Config Address Write Register

*Table 2397.*0x2C - CONFW - SPI2 config address write register

31	0
CONFW	
0x40000000	
r	

31 : 0 Configuration read address (CONFW) - Defines the base address for the memory area where the core is allowed to make accesses. This is a status (read only) register, the contents of this register can be modified by the configuration write address command.

131.7.12SPI2 Config Address Read Register

*Table 2398.*0x30 - CONFR - SPI2 config address read register

31	0
CONFR	
0x40000000	
r	

31 : 0 Configuration read address (CONFR) - Defines the base address for the memory area where the core is allowed to make accesses. This is a status (read only) register, the contents of this register can be modified by the configuration read address command.

131.8 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x0A7. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

131.9 Implementation

131.9.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

131.9.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

131.10 Configuration options

Table 2399 shows the configuration options of the core (VHDL generics).

*Table 2399.*Configuration options

Generic name	Function	Allowed range	Default
gPINDEX	APB slave index	0 - NAPBSLV-1	0
gHINDEX	AHB master index	0 - NAHBMST	0
gPADDR	ADDR field of the APB BAR	0 - 16#FFF#	0
gPMASK	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
gPIRQ	Interrupt line driven by APB interface.	0 - NAHBIRQ-1	1
gSPI2	Implement SPI protocol 2	0 - 1	1
gCONFW	Default value for config write address register	0 - 16#FFFFFF#	16#400000#
gCONFR	Default value for config read address register	0 - 16#FFFFFF#	16#400000#
gOEPOL	Output enable polarity	0 - 1	0

131.11 Signal descriptions

Table 2400 shows the interface signals of the core (VHDL ports).

*Table 2400.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBI	*	Input	AHB master input signals	-
AHBO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
SCK_N	N/A	Input	SPI clock line input Nominal	-
MOSI_N	N/A	Input	SPI data line input Nominal	-
SLVSEL_N	N/A	Input	SPI slave select output Nominal	Low
MISO_N	N/A	Output	SPI data line output Nominal	-
MISO_EN_N	N/A	Output	SPI data line output enable Nominal	**
SCK_R	N/A	Input	SPI clock line input Redundant	-
MOSI_R	N/A	Input	SPI data line input Redundant	-
SLVSEL_R	N/A	Input	SPI slave select output Redundant	Low
MISO_R	N/A	Output	SPI data line output Redundant	-
MISO_EN_R	N/A	Output	SPI data line output enable Redundant	**

* see GRLIB IP Library User's Manual
** depends on value of OEPOL VHDL generic.

131.12 Library dependencies

Table 2401 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2401.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	SPICOMP	Component, signals	Component declaration, SPI signal definitions
GAISLER	GRSPICOMP	Component, signals	Component declaration, SPI signal definitions

131.13 Instantiation

```
library  ieee;
use      ieee.std_logic_1164.all;

library  grlib, techmap;
use      grlib.amba.all;
use      grlib.devices.all;
use      grlib.stdlib.all;
use techmap.gencomp.all;
library  gaisler;
use gaisler.spicomp.all;
use gaisler.grspicomp.all;

entity grspislave_ex is
   port (
     rstn        : in  std_ulogic;
     clk         : in  std_ulogic;
     ahbmi       : in  ahb_mst_in_type;
     ahbmo       : out ahb_mst_out_type;
     apbi        : in  APB_Slv_In_Type;
     apbo        : out APB_Slv_Out_Type;
     -- SPI signals
     miso_n      : out std_ulogic;
     miso_en_n   : out std_ulogic;
     mosi_n      : in  std_ulogic;
     sck_n       : in  std_ulogic;
     slvsel_n    : in  std_ulogic;

     miso_r      : out std_ulogic;
     miso_en_r   : out std_ulogic;
     mosi_r      : in  std_ulogic;
     sck_r       : in  std_ulogic;
     slvsel_r    : in  std_ulogic);
end entity grspislave_ex;

architecture rtl of grspislave_ex is

  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector;
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector;

  signal nom_sck        :       std_ulogic;
  signal nom_miso       :       std_ulogic;
  signal nom_miso_en    :       std_ulogic;
  signal nom_mosi       :       std_ulogic;
  signal nom_slvsel     :       std_ulogic;

  signal red_sck        :       std_ulogic;
  signal red_miso       :       std_ulogic;
```

```
  signal red_miso_en    :       std_ulogic;
  signal red_mosi       :       std_ulogic;
  signal red_slvsel     :       std_ulogic;
  constant OEPOL : integer := padoen_polarity(padtech);

begin

     grspislave0: grspislave
        generic map (
           gHINDEX         => 4,
           gPINDEX         => 6,
           gPADDR          => 6,
           gPIRQ           => 6,
           gSPI2           => 1,
           gCONFW          => 16#400000#,
           gCONFR          => 16#400000#,
           gOEPOL          => OEPOL)
        port map (
          rstn             => rstn,
          clk              => clk,
          ahbmi            => ahbmi,
          ahbmo            => ahbmo(4),
          apbi             => apbi,
          apbo             => apbo(6),

          miso_n           => nom_miso,
          miso_en_n        => nom_miso_en,
          mosi_n           => nom_mosi,
          sck_n            => nom_sck,
          slvsel_n         => nom_slvsel,

          miso_r           => red_miso,
          miso_en_r        => red_miso_en,
          mosi_r           => red_mosi,
          sck_r            => red_sck,
          slvsel_r         => red_slvsel);

 --Nominal
   sck_p1pad : inpad generic map (tech => padtech)
      port map (sck_n, nom_sck);

   mosi_p1pad : inpad generic map (tech => padtech)
      port map (mosi_n, nom_mosi);

   miso_p1pad : toutpad generic map (tech => padtech, oepol => OEPOL)
      port map (miso_n, nom_miso, nom_miso_en);

   cs_p1pad : inpad generic map (tech => padtech)
      port map (slvsel_n, nom_slvsel);

--Redundant
   sck_p2pad : inpad generic map (tech => padtech)
      port map (sck_r, red_sck);

   mosi_p2pad : inpad generic map (tech => padtech)
      port map (mosi_r, red_mosi);

   miso_p2pad : toutpad generic map (tech => padtech, oepol => OEPOL)
      port map (miso_r, red_miso, nom_miso_en);

   cs_p2pad : inpad generic map (tech => padtech)
      port map (slvsel_r, red_slvsel);
end architecture rtl;
```

132 SRCTRL- 8/32-bit PROM/SRAM Controller

132.1 Overview

SRCTRL is an 8/32-bit PROM/SRAM/IO controller that interfaces external asynchronous SRAM, PROM and I/O to the AMBA AHB bus. The controller can handle 32-bit wide SRAM and I/O, and either 8- or 32-bit PROM.

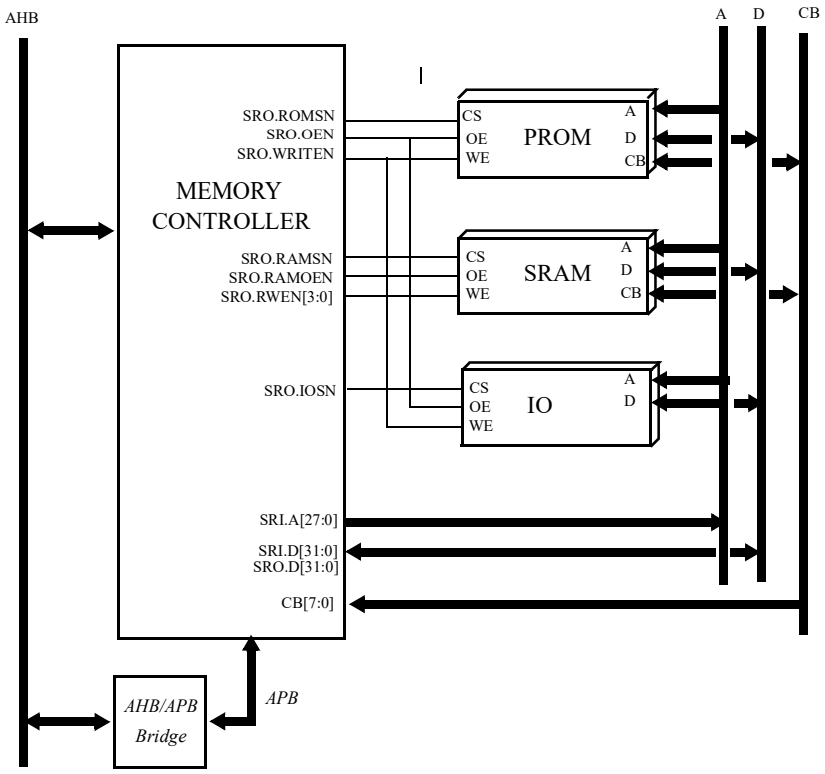


Figure 332. 8/32-bit PROM/SRAM/IO controller

The controller is configured through VHDL-generics to decode three address ranges: PROM, SRAM and I/O area. By default PROM area is mapped into address range 0x0 - 0x00FFFFFF, the SRAM area is mapped into address range 0x40000000 - 0x40FFFFFF, and the I/O area is mapped to 0x20000000 - 0x20FFFFFF.

One chip select is decoded for the I/O area, while SRAM and PROM can have up to four and two select signals respectively. The controller generates both a common write-enable signal (WRITEN) as well as four byte-write enable signals (WREN). If the SRAM uses a common write enable signal the controller can be configured to perform read-modify-write cycles for byte and half-word write accesses. Number of waitstates is separately configurable for the three address ranges.

A single write-enable signal is generated for the PROM area (WRITEN), while four byte-write enable signals (RWEN[3:0]) are provided for the SRAM area. If the external SRAM uses common write enable signal, the controller can be configured to perform read-modify-write cycles for byte and half-word write accesses.

Number of waitstates is configurable through VHDL generics for both PROM and SRAM areas.

A signal (BDRIVE) is provided for enabling the bidirectional pads to which the data signals are connected. The oepol generic is used for selecting the polarity of these enable signals. If output delay is an issue, a vectored output enable signal (VBDRIVE) can be used instead. In this case, each pad has

its own enable signal driven by a separate register. A directive is placed on these registers so that they will not be removed during synthesis (if the output they drive is used in the design).

132.1.1 Endianness

The core is designed for big-endian systems.

132.2 8-bit PROM access

The SRCTRL controller can be configured to access a 8-bit wide PROM. The data bus of external PROM should be connected to the upper byte of the 32-bit data bus, i.e. D[31:24]. The 8-bit mode is enabled with the prom8en VHDL generic. When enabled, read accesses to the PROM area will be done in four-byte bursts. The whole 32-bit word is then presented on the AHB data bus. Writes should be done one byte at a time and the byte should always be driven on bit 31-24 on the AHB data bus independent of the byte address.

It is possible to dynamically switch between 8- and 32-bit PROM mode using the BWIDTH[1:0] input signal. When BWIDTH is “00” then 8-bit mode is selected. If BWIDTH is “10” then 32-bit mode is selected. Other BWIDTH values are reserved for future use.

SRAM access is not affected by the 8-bit PROM mode.

132.3 PROM/SRAM waveform

Read accesses to 32-bit PROM and SRAM has the same timing, see figure below.

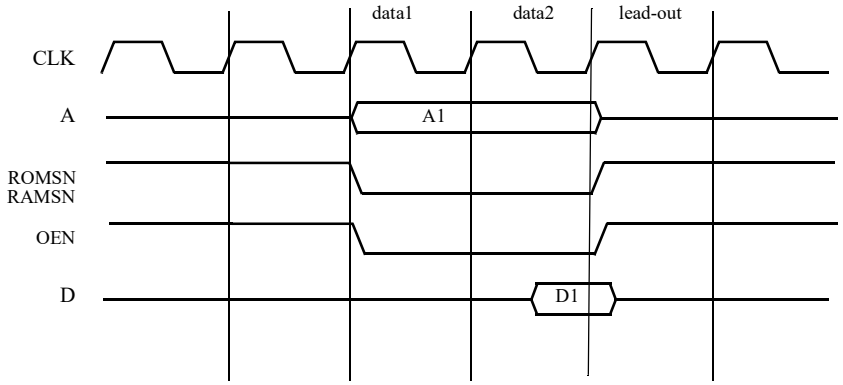


Figure 333. 32-bit PROM/SRAM/IO read cycle

The write access for 32-bit PROM and SRAM can be seen below.

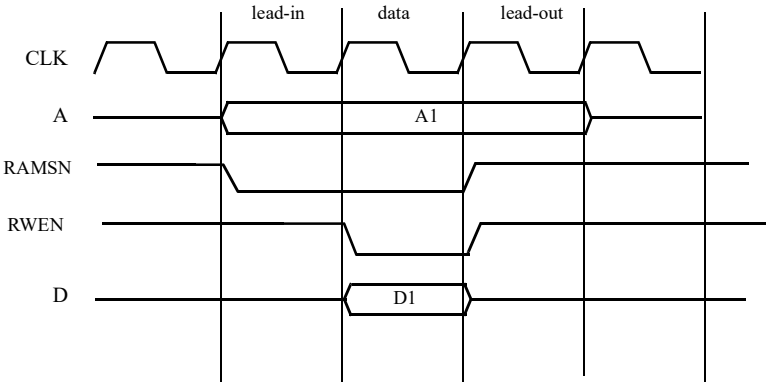


Figure 334. 32-bit PROM/SRAM/IO write cycle

If waitstates are configured through the VHDL generics, one extra data cycle will be inserted for each waitstate in both read and write cycles.

132.4 Burst cycles

To improve the bandwidth of the memory bus, accesses to consecutive addresses can be performed in burst mode. Burst transfers will be generated when the memory controller is accessed using an AHB burst request. These includes instruction cache-line fills and burst from DMA masters. The timing of a burst cycle is identical to the programmed basic cycle with the exception that during read cycles, the lead-out cycle will only occurs after the last transfer.

132.5 Registers

The core does not implement any user programmable registers.

All configuration is done through the VHDL generics.

132.6 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x008. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

132.7 Implementation

132.7.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers. The registers driving external chip select, output enable and output enables for the data bus have asynchronous reset.

132.7.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

132.8 Configuration options

Table 2403 shows the configuration options of the core (VHDL generics).

*Table 2402.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0xFFFFFF.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.	0 - 16#FFF#	16#FF0#
ramaddr	ADDR field of the AHB BAR1 defining SRAM address space. Default SRAM area is 0x40000000-0x40FFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR1 defining SRAM address space.	0 -16#FFF#	16#FF0#
ioaddr	ADDR field of the AHB BAR2 defining IO address space. Default IO area is 0x20000000-0x20FFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR2 defining IO address space.	0 -16#FFF#	16#FF0#
ramws	Number of waitstates during access to SRAM area	0 - 15	0
romws	Number of waitstates during access to PROM area	0 - 15	2
iows	Number of waitstates during access to IO area	0 - 15	2
rmw	Enable read-modify-write cycles.	0 - 1	0
prom8en	Enable 8 - bit PROM accesses	0 - 1	0
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
srbanks	Set the number of SRAM banks	1 - 5	1
banksz	Set the size of bank 1 - 4. 0 = 8 Kbyte, 1 = 16 Kbyte, ... , 13 = 64Mbyte.	0 - 13	13
romasel	address bit used for PROM chip select.	0 - 27	19

132.9 Signal description

Table 2402 shows the interface signals of the core (VHDL ports).

*Table 2403.*Signal descriptions

Signal name	Field	Type	Function	Polarity
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
SRI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Not used	-
	BEXCN	Input	Not used	-
	WRN[3:0]	Input	Not used	-
	BWIDTH[1:0]	Input	BWIDTH=”00” => 8-bit PROM mode BWIDTH=”10” => 32-bit PROM mode	-
	SD[31:0]	Input	Not used	-

Table 2403. Signal descriptions

Signal name	Field	Type	Function	Polarity
SRO	ADDRESS[27:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data	High
	RAMSN[4:0]	Output	SRAM chip-select	Low
	RAMOEN[4:0]	Output	SRAM output enable	Low
	IOSN	Output	Not used. Driven to '1' (inactive)	Low
	ROMSN[1:0]	Output	PROM chip-select	Low
	RAMN	Output	Common SRAM chip-select. Asserted when one of the RAMSN[4:0] signals is asserted.	Low
	ROMN	Output	Common PROM chip-select. Asserted when one of the ROMSN[1:0] signals is asserted.	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0].	Low
	MBEN[3:0]	Output	Byte enable: MBEN[0] corresponds to DATA[31:24], MBEN[1] corresponds to DATA[23:16], MBEN[2] corresponds to DATA[15:8], MBEN[3] corresponds to DATA[7:0].	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus. Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0].	Low/High[2]
	VBDRIVE[31:0]	Output	Identical to BDRIVE but has one signal for each data bit. Every index is driven by its own register. This can be used to reduce the output delay.	Low/High[2]
	READ	Output	Read strobe	High
	SA[14:0]	Output	Not used	High
AHBSI	1)	Input	AHB slave input signals	-
AHBSO	1)	Output	AHB slave output signals	-
SDO	SDCASN	Output	Not used. All signals are driven to inactive state.	Low

1) See GRLIB IP Library User's Manual

2) Polarity is selected with the oepol generic

132.10 Library dependencies

Table 2404 shows libraries used when instantiating the core (VHDL libraries).

*Table 2404.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

132.11 Component declaration

The core has the following component declaration.

```
component srctrl
  generic (
    hindex  : integer := 0;
    romaddr : integer := 0;
    rommask : integer := 16#ff0#;
    ramaddr : integer := 16#400#;
    rammask : integer := 16#ff0#;;
    ioaddr  : integer := 16#200#;
    iomask  : integer := 16#ff0#;
    ramws   : integer := 0;
    romws   : integer := 2;
    iows    : integer := 2;
    rmw     : integer := 0;-- read-modify-write enable
    prom8en : integer := 0;
    oepol   : integer := 0;
    srbanks : integer range 1 to 5 := 1;
    banksz  : integer range 0 to 13:= 13;
    romasel : integer range 0 to 27:= 19
  );
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;
    ahbso   : out ahb_slv_out_type;
    sri     : in  memory_in_type;
    sro     : out memory_out_type;
    sdo     : out sdctrl_out_type
  );
end component;
```

132.12 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined on the example designs port map and connected to the memory controller. System clock and reset are generated by GR Clock Generator and Reset Generator.

Memory controller decodes default memory areas: PROM area is 0x0 - 0xFFFFFF and SRAM area is 0x40000000 - 0x40FFFFF. The 8-bit PROM mode is disabled. Two SRAM banks of size 64 Mbyte are used and the fifth chip select is disabled.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
```

```
use gaisler.pads.all;   -- used for I/O pads
use gaisler.misc.all;
library esa;
use esa.memoryctrl.all;

entity srctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;

    -- memory bus
    address  : out   std_logic_vector(27 downto 0); -- memory bus
    data     : inout std_logic_vector(31 downto 0);
    ramsn    : out   std_logic_vector(4 downto 0);
    ramoen   : out   std_logic_vector(4 downto 0);
    rwen     : inout std_logic_vector(3 downto 0);
    romsn    : out   std_logic_vector(1 downto 0);
    iosn     : out   std_logic;
    oen      : out   std_logic;
    read     : out   std_logic;
    writen   : inout std_logic;
    brdyn    : in    std_logic;
    bexcn    : in    std_logic;
    modesel  : in    std_logic; --PROM width select
-- sdram i/f
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0) -- optional sdram data
       );
end;

architecture rtl of srctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo : sdctrl_out_type;

  signal wprot : wprot_out_type;  -- dummy signal, not used
  signal clkm, rstn : std_ulogic; -- system clock and reset

-- signals used by clock and reset generators
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;

  signal gnd : std_ulogic;

begin

  -- AMBA Components are defined here ...


  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                                tech => virtex2, sdinvclk => 0)
```

```
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  rst0 : rstgen
  port map (resetn, clkm, cgo.clklock, rstn);


  -- Memory controller
srctrl0 : srctrl generic map (rmw => 1, prom8en => 0, srbanks => 2,
  banksz => 13, ramsel5 => 0)
    port map (rstn, clkm, ahbsi, ahbso(0), memi, memo, sdo);


  -- I/O pads driving data memory bus data signals
  datapads : for i in 0 to 3 generate
      data_pad : iopadv generic map (width => 8)
      port map (pad => data(31-i*8 downto 24-i*8),
                o => memi.data(31-i*8 downto 24-i*8),
                en => memo.bdrive(i),
                i => memo.data(31-i*8 downto 24-i*8));
  end generate;

  -- Alternative I/O pad instantiation with vectored enable instead
  datapads : for i in 0 to 3 generate
      data_pad : iopadvv generic map (width => 8)
      port map (pad => data(31-i*8 downto 24-i*8),
                o => memi.data(31-i*8 downto 24-i*8),
                en => memo.bdrive(31-i*8 downto 24-i*8),
                i => memo.data(31-i*8 downto 24-i*8));
  end generate;

  -- connect memory controller outputs to entity output signals
  address <= memo.address; ramsn <= memo.ramsn; romsn <= memo.romsn;
  oen <= memo.oen; rwen <= memo.wrn; ramoen <= memo.ramoen;
  writen <= memo.writen; read <= memo.read; iosn <= memo.iosn;
  sdcke <= sdo.sdcke; sdwen <= sdo.sdwen; sdcsn <= sdo.sdcsn;
  sdrasn <= sdo.rasn; sdcasn <= sdo.casn; sddqm <= sdo.dqm;

end;
```

133 SSRCTRL- 32-bit SSRAM/PROM Controller

133.1 Overview

The memory controller (SSRCTRL) is an 32-bit SSRAM/PROM/IO controller that interfaces external Synchronous pipelined SRAM, PROM, and I/O to the AMBA AHB bus. The controller acts as a slave on the AHB bus and has a configuration register accessible through an APB slave interface. Figure 335 illustrates the connection between the different devices.

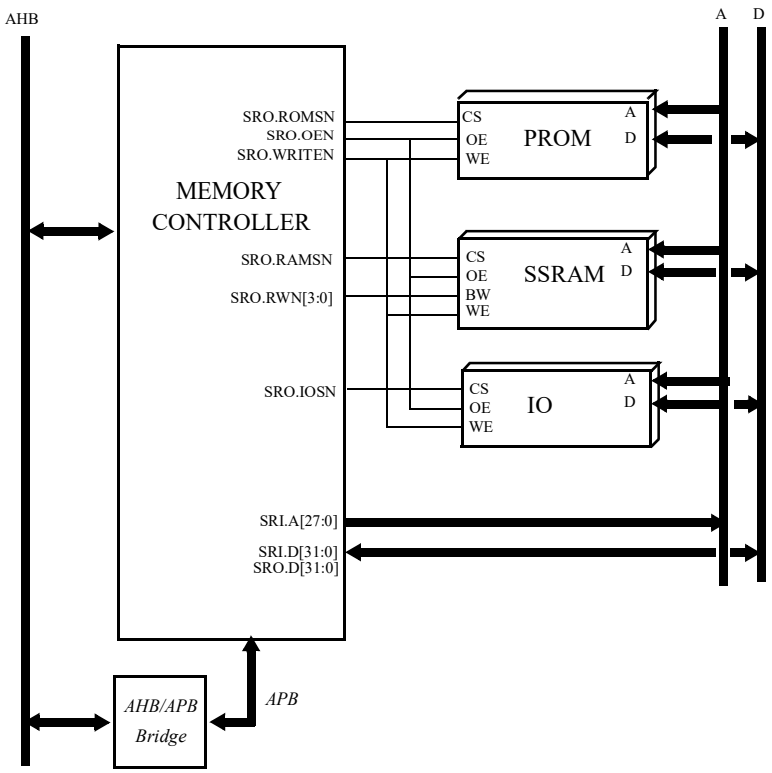


Figure 335. 32-bit SSRAM/PROM/IO controller

The controller is configured by VHDL-generics to decode three address ranges: PROM, SSRAM and I/O area. By default PROM area is mapped into address range 0x0 - 0x00FFFFFF; the SSRAM area is mapped into address range 0x40000000 - 0x40FFFFFF; and the I/O area is mapped to 0x20000000 - 0x20FFFFFF.

One chip select is generated for each of the address areas. The controller generates both a common write-enable signal (WRITEN) as well as four byte-write enable signals (WRN). The byte-write enable signal enables byte and half-word write access to the SSRAM.

A signal (BDRIVE) is provided for enabling the bidirectional pads to which the data signals are connected. The oepol generic is used to select the polarity of these enable signals. If output delay is an issue, a vectored output enable signal (VBDRIVE) can be used instead. In this case, each pad has its own enable signal driven by a separate register. A directive is placed on these registers so that they will not be removed during synthesis (in case the output they drive is used in the design).

The SSRCTRL conteoller can optionally support 16-bit PROM/IO devices. This is enabled through the BUS16 generic. A 32-bit access to the PROM or IO area will be translated into two 16-bit accesses with incrementing address.

133.1.1 Endianness

The core is designed for big-endian systems.

133.2 SSRAM/PROM waveform

Because the SSRAM (Synchronous pipelined SRAM) has a pipelined structure, the data output has a latency of three clock cycles. The pipelined structure enables a new memory operation to be issued each clock cycle. Figure 335 and figure 336 show timing diagrams for the SSRAM read and write accesses.

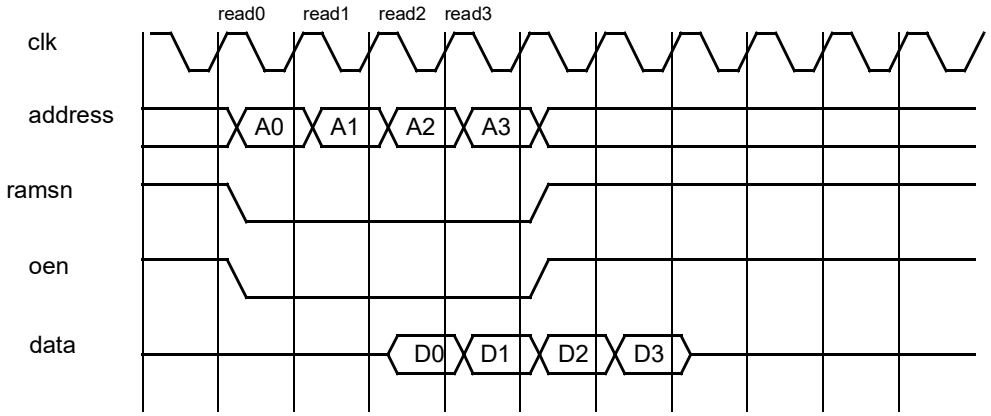


Figure 336. 32-bit SSRAM read cycle

As shown in the figure above, the controller always perform a burst read access to the memory. This eliminates all data output latency except for the first word when a burst read operation is executed.

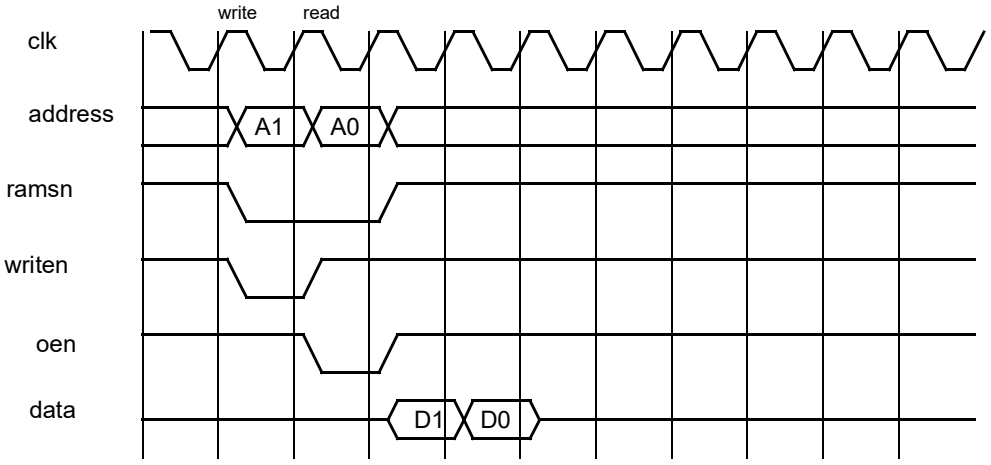


Figure 337. 32-bit SSRAM write cycle

A write operation takes three clock cycles. On the rising edge of the first clock cycle, the address and control signals are latched into the memory. On the next rising edge, the memory puts the data bus in high-impedance mode. On the third rising edge the data on the bus is latched into the memory and the write is complete. The controller can start a new memory (read or write) operation in the second clock cycle. In figure 337 this is illustrated by a read operation following the write operation.

Due to the memory automatically putting the data bus in high-impedance mode when a write operation is performed, the output-enable signal (OEN) is held active low during all SSRAM accesses (including write operations).

133.2.1 PROM and IO access

For the PROM and I/O operations, a number of waitstates can be inserted to increase the read and write cycle. The number of waitstates can be configured separately for the I/O and PROM address ranges, through a programmable register mapped into the APB address space. After a reset the wait-states for PROM area is set to its maximum (15). Figure 338 and figure 339 show timing diagrams for the PROM read and write accesses.

Read accesses to 32-bit PROM and I/O has the same timing, see figure 338

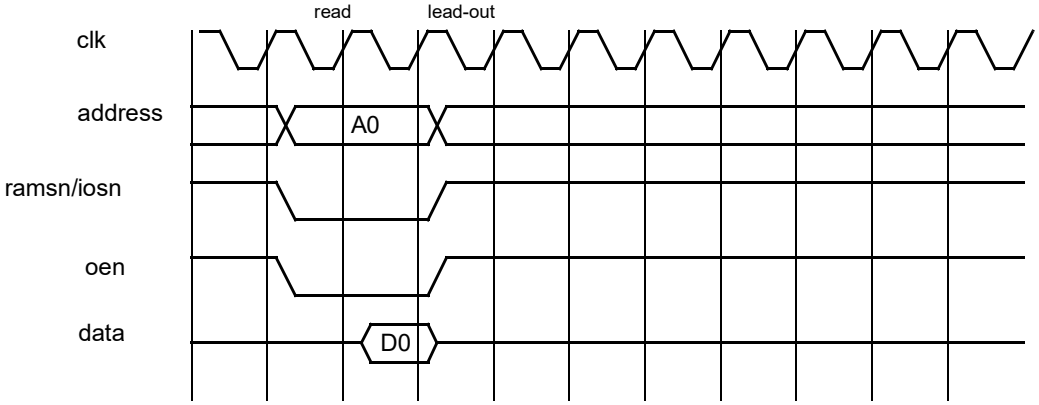


Figure 338. 32-bit PROM/IO read cycle

The write access for 32-bit PROM and I/O can be seen in figure 339

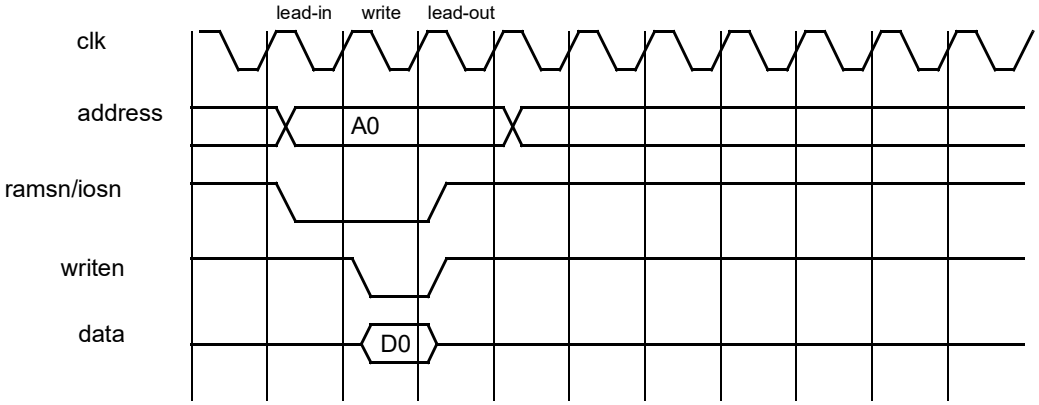


Figure 339. 32-bit PROM/IO write cycle

The SSRCTRL conteoller can optionally support 16-bit PROM/IO devices. This is enabled through the BUS16 generic. A 32-bit access to the PROM or IO area will be translated into two 16-bit accesses with incrementing address. A 16-bit access will result in one bus access only. 8-bit accesses are not allowed.

16-bit PROM/IO operation is enabled by writing “01” to the romwidth field in SSRAM control register. At reset, the romwidth field is set by the MEMI.BWIDTH input signal.

Read accesses to 16-bit PROM and I/O has the same timing, see figure 340

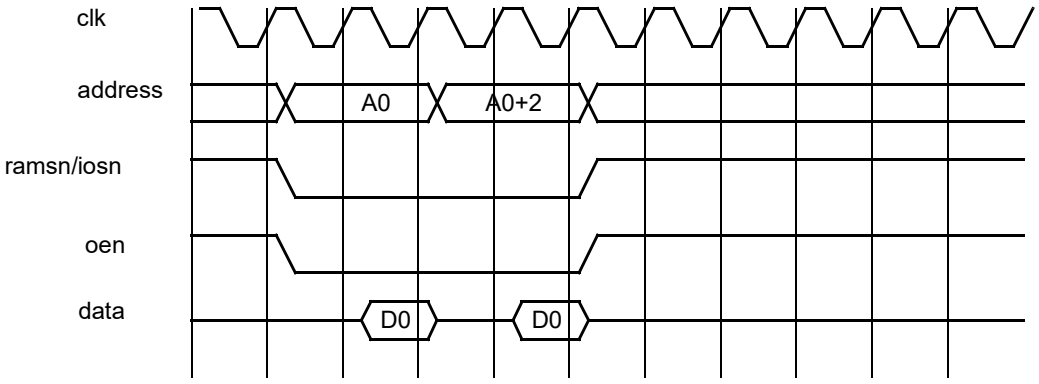


Figure 340. 32-bit PROM/IO read cycle in 16-bit mode

The write access for 32-bit PROM and I/O can be seen in figure 341

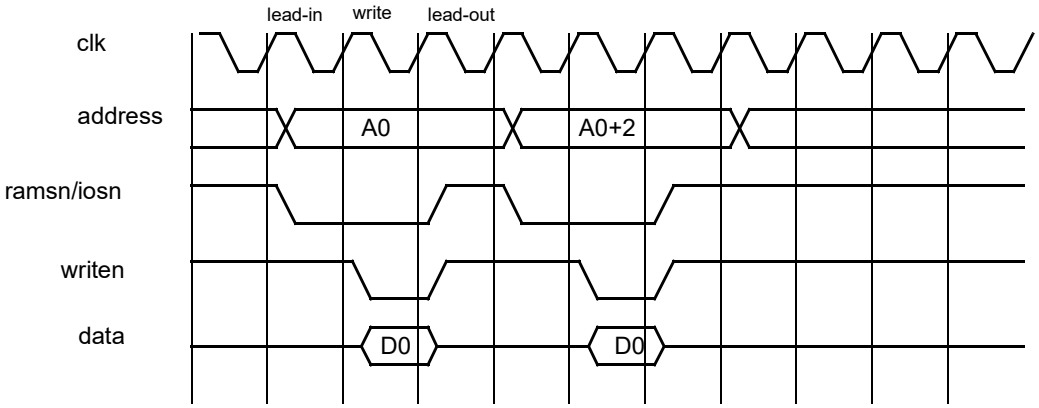


Figure 341. 32-bit PROM/IO write cycle in 16-bit mode

133.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 2405.*SSRAM controller registers

APB address offset	Register
0x00	Memory configuration register

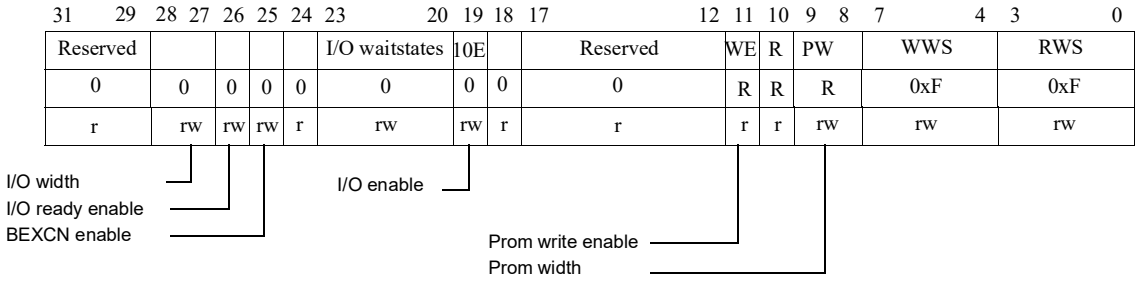


Figure 342. Memory configuration register

[3:0]: Prom read waitstates. Defines the number of waitstates during prom read cycles (“0000”=0, “0001”=1,... “1111”=15).
[7:4]: Prom write waitstates. Defines the number of waitstates during prom write cycles (“0000”=0, “0001”=1,... “1111”=15).
[9:8]: Prom width. Defines the data with of the prom area (“01”=16, “10”=32).
[10]: Reserved
[11]: Prom write enable. If set, enables write cycles to the prom area. NOT USED.
[17:12]: Reserved
[19]: I/O enable. If set, the access to the memory bus I/O area are enabled. NOT USED.
[23:20]: I/O waitstates. Defines the number of waitstates during I/O accesses (“0000”=0, “0001”=1, “0010”=2,..., “1111”=15).
[25]: Bus error (BEXCN) enable. NOT USED.
[26]: Bus ready (BRDYN) enable. NOT USED.
[28:27]: I/O bus width. Defines the data with of the I/O area (“01”=16, “10”=32).

During power-up (reset), the PROM waitstates fields are set to 15 (maximum) and the PROM bus width is set to the value of MEMI.BWIDTH. All other fields are initialized to zero.

133.4 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x00A. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

133.5 Implementation

133.5.1 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User’s Manual). The core is designed for big-endian systems.

133.6 Configuration options

Table 2406 shows the configuration options of the core (VHDL generics).

*Table 2406.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0xFFFFFF.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.	0 - 16#FFF#	16#FF0#
ramaddr	ADDR field of the AHB BAR1 defining RAM address space. Default RAM area is 0x40000000-0x40FFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR1 defining RAM address space.	0 -16#FFF#	16#FF0#
ioaddr	ADDR field of the AHB BAR2 defining IO address space. Default IO area is 0x20000000-0x20FFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR2 defining IO address space.	0 -16#FFF#	16#FF0#
paddr	ADDR field of the APB BAR configuration registers address space.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR configuration registers address space.	0 - 16#FFF#	16#FFF#
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
bus16	Enable support for 16-bit PROM/IO accesses	0 - 1	0

133.7 Signal descriptions

Table 2407 shows the interface signals of the core (VHDL ports).

*Table 2407.*Signal descriptions

Signal name	Field	Type	Function	Polarity
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
SRI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Not used	-
	BEXCN	Input	Not used	-
	WRN[3:0]	Input	Not used	-
	BWIDTH[1:0]	Input	PROM bus width at reset	-
	SD[63:0]	Input	Not used	-
	CB[7:0]	Input	Not used	-
	SCB[7:0]	Input	Not used	-
	EDAC	Input	Not used	-

Table 2407. Signal descriptions

Signal name	Field	Type	Function	Polarity
SRO	ADDRESS[27:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data	High
	SDDATA[63:0]	Output	Not used	-
	RAMSN[7:0]	Output	SSRAM chip-select, only bit 0 is used	Low
	RAMOEN[7:0]	Output	Same as OEN	Low
	IOSN	Output	I/O chip-select	Low
	ROMSN[7:0]	Output	PROM chip-select, only bit 0 is used	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SSRAM byte write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0].	Low
	MBEN[3:0]	Output	Not used	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus. Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0]. Any BDRIVE[ ] signal can be used for CB[ ].	Low/High[2]
	VBDRIVE[31:0]	Output	Identical to BDRIVE but has one signal for each data bit. Every index is driven by its own register. This can be used to reduce the output delay.	Low/High[2]
	SVBDRIVE	Output	Not used	-
	READ	Output	Not used	-
	SA[14:0]	Output	Not used	-
	CB[7:0]	Output	Not used	-
	SCB[7:0]	Output	Not used	-
	VCDRIVE[7:0]	Output	Not used	-
	SVCDRIVE[7:0]	Output	Not used	-
	CE	Output	Not used	-
AHBSI	1)	Input	AHB slave input signals	-
AHBSO	1)	Output	AHB slave output signals	-
APBI	1)	Input	APB slave input signals	-
APBO	1)	Output	APB slave output signals	-

1) See GRLIB IP Library User's Manual

2) Polarity is selected with the oepol generic

133.8 Library dependencies

Table 2408 shows libraries used when instantiating the core (VHDL libraries).

*Table 2408.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

133.9 Component declaration

The core has the following component declaration.

```
component ssrctrl
  generic (
    hindex  : integer := 0;
    pindex  : integer := 0;
    romaddr : integer := 0;
    rommask : integer := 16#ff0#;
    ramaddr : integer := 16#400#;
    rammask : integer := 16#ff0#;
    ioaddr  : integer := 16#200#;
    iomask  : integer := 16#ff0#;
    paddr   : integer := 0;
    pmask   : integer := 16#fff#;
     oepol   : integer := 0;
     bus16 : integer := 0
  );
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;
    ahbso   : out ahb_slv_out_type;
    apbi    : in  apb_slv_in_type;
    apbo    : out apb_slv_out_type;
    sri     : in  memory_in_type;
    sro     : out memory_out_type

  );
end component;
```

133.10 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it, including the memory controller. The external memory bus is defined in the example designs port map and connected to the memory controller. System clock and reset are generated by the Clkgen_ml401 Clock Generator and GR Reset Generator.

The memory controller decodes default memory areas: PROM area is 0x0 - 0x00FFFFFF, I/O-area is 0x20000000-0x20FFFFFF and RAM area is 0x40000000 - 0x40FFFFFF.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib, techmap;
use grlib.amba.all;
use grlib.stdlib.all;
use techmap.gencomp.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.misc.all;

entity ssrctrl_ex is
```

```
 port (
    sys_rst_in: in  std_ulogic;
    sys_clk: in  std_ulogic; -- 100 MHz main clock
    sram_flash_addr : out std_logic_vector(22 downto 0);
    sram_flash_data : inout std_logic_vector(31 downto 0);
    sram_cen  : out std_logic;
    sram_bw   : out std_logic_vector (0 to 3);
    sram_flash_oe_n : out std_ulogic;
    sram_flash_we_n : out std_ulogic;
    flash_ce  : out std_logic;
    sram_clk  : out std_ulogic;
    sram_clk_fb: in  std_ulogic;
    sram_mode : out std_ulogic;
    sram_adv_ld_n : out std_ulogic;
    sram_zz : out std_ulogic;
    iosn    : out std_ulogic;
);
end;

architecture rtl of ssrctrl_ex is

-- Clock generator component
component clkgen_ml401
  generic (
    clk_mul  : integer := 1;
    clk_div  : integer := 1;
    freq     : integer := 100000);-- clock frequency in KHz
  port (
    clkin   : in  std_logic;
    clk     : out std_logic;-- main clock
    ddrclk  : out std_logic;-- DDR clock
    ddrclkfb: in  std_logic;-- DDR clock feedback
    ddrclk90  : out std_logic;-- DDR 90 clock
    ddrclk180 : out std_logic;-- 180 clock
    ddrclk270 : out std_logic;-- DDR clock
    ssrclk  : out std_logic;-- SSRAM clock
    ssrclkfb: in  std_logic;-- SSRAM clock feedback
    cgi     : in clkgen_in_type;
    cgo     : out clkgen_out_type);
end component;


-- signals used to connect memory controller and memory bus
signal memi  : memory_in_type;
signal memo  : memory_out_type;

-- AMBA bus (AHB and APB)
signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector := (others => apb_none);
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
signal ahbmi : ahb_mst_in_type;
signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

-- Signals used by clock and reset generators
signal clkm, rstn, rstraw, srclkl : std_ulogic;
signal cgi   : clkgen_in_type;
signal cgo   : clkgen_out_type;
signal ddrclkfb, ssrclkfb, ddr_clkl, ddr_clknl : std_ulogic;


begin

  clkgen0 : clkgen_ml401  -- clock generator
  port map (sys_clk, clkm, ddr_clkl, ddrclkfb, open, ddr_clknl, open, sram_clk,
      sram_clk_fb, cgi, cgo);

  rst0 : rstgen-- reset generator
  port map (sys_rst_in, clkm, cgo.clklock, rstn, rstraw);

  -- AMBA Components are defined here ...
```

```
  -- Memory controller
  mctrl0 : ssrctrl generic map (hindex => 0, pindex => 0)
  port map (rstn, clkm, ahbsi, ahbso(0), apbi, apbo(0), memi, memo);

  -- connect memory controller outputs to entity output signals
  sram_adv_ld_n <=  '0'; sram_mode <=  '0'; sram_zz <= '0';
  sram_flash_addr <= memo.address(24 downto 2); sram_cen <= memo.ramsn(0);
  flash_ce <= memo.romsn(0); sram_flash_oe_n <= memo.oen; iosn <= memo.iosn;
  sram_bw <= memo.wrn; sram_flash_we_n <= memo.writen;

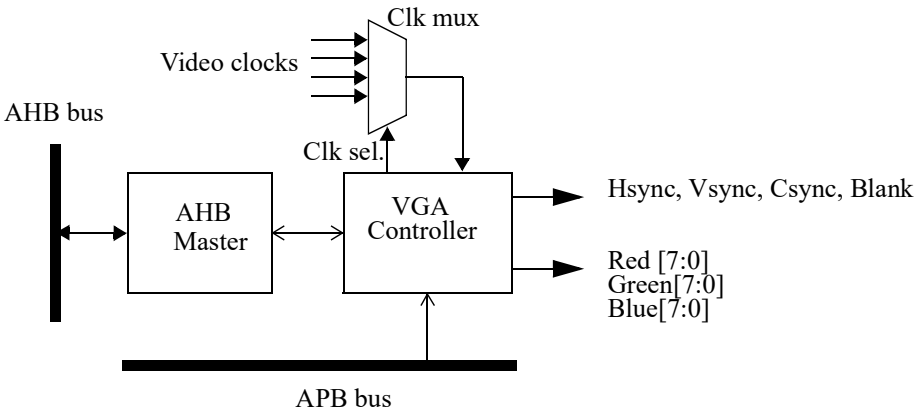
  -- I/O pad instantiation with vectored enable instead
  bdr : for i in 0 to 31 generate
      data_pad : iopad generic map (tech => padtech)
      port map (sram_flash_data(i), memo.data(i),
             memo.vbdrive(i), memi.data(i));
  end generate;

end;
```

134 SVGACTRL - VGA Controller Core

134.1 Overview

The core is a pixel based video controller (frame buffer), capable of displaying standard and custom resolutions with variable bit depth and refresh rates. The video controller consists of a synchronization unit, main control unit, FIFO unit and an AHB master as shown in the figure below.



134.2 Operation

The core uses external frame buffer memory located in the AHB address space. A frame on the display is created by fetching the pixel data from memory and sending it to the screen through an external DAC using three 8-bit color vectors. To hide the AHB bus latency, the pixel data is buffered in a FIFO inside the core. The start address of the frame buffer is specified in the Frame buffer Memory Position register, and can be anywhere in the AHB address space. In addition to the color vectors the video controller also generates HSYNC, VSYNC, CSYNC and BLANK signals control signals.

The video timing is programmable through the Video Length, Front Porch, Sync Length and Line Length registers. The bit depth selection and enabling of the controller is done through the status register. These values make it possible to display a wide range of resolutions and refresh rates.

The pixel clock can be either static or dynamic multiplexed. The frequency of the pixel clock is calculated as *Horizontal Line Length * Vertical Line Length * refresh rate*. When using a dynamically multiplexed clock, bits [5:4] in the status register are used to control the clock selector. The dynamic pixel clocks should be defined in the core's VHDL generics to allow software to read out the available pixel clock frequencies.

The core can use bit depths of 8, 16 and 32 bits. When using 32 bits, bits[23:0] are used, when 16 bits a [5,6,5] color scheme is used and when using 8 bits a color lookup table "CLUT" is used. The CLUT has 256 positions, each 24 bits wide, and the 8 bit values read from memory are used to index the CLUT to obtain the actual color.

134.3 DVI support

In order to initialize a DVI transmitter, an additional core such as the I^2C master is normally required. Additional glue logic may also be required since the interfaces of DVI transmitters differ between manufacturers and product lines. Examples on how to interface the core to a DVI transmitter are available in the GRLIB IP Library's template designs.

134.4 Registers

The core is programmed through registers mapped into APB address space.

Table 2409. VGA controller registers

APB address offset	Register
0x00	Status register
0x04	Video length register
0x08	Front Porch register
0x0C	Sync Length register
0x10	Line Length register
0x14	Framebuffer Memory Position register
0x18	Dynamic Clock 0 register
0x1C	Dynamic Clock 1 register
0x20	Dynamic Clock 2 register
0x24	Dynamic Clock 3 register
0x28	CLUT Access register

134.4.1 Status Register

*Table 2410.*0x00 -STAT - Status register

31 10	9	8	7 6	5 4	3	2	1	0
RESERVED	VPOL	HPOL	CLKSEL	BDSEL	VR	R	RST	EN
0	NR	NR	0	NR	0	0	0	0
r	rw	rw	rw	rw	r	r	rw	rw

31:10 RESERVED
9 V polarity (VPOL)- Sets the polarity for the vertical sync pulse.
8 H polarity (HPOL) - Sets the polarity for the horizontal sync pulse.
7:6 Clock Select (CLKSEL) Clock selector when using dynamic pixelclock
5:4 Bit depth selector (BDSEL) - “01” = 8-bit mode; “10” = 16-bit mode; “11” = 32-bit mode
3 Vertical refresh (VR) - High during vertical refresh
2 RESERVED
1 Reset (RST) - Resets the core
0 Enable (EN) - Enables the core

134.4.2 Video Length Register

*Table 2411.*0x04 - VLEN - Video Length register

31 16	15 0
VRES	HRES
0	0
rw	rw

31:16 Vertical screen resolution (VRES) - Vertical screen resolution in pixels -1
15:0 Horisontal screen resolution (HRES) - Horizontal screen resolution in pixels -1.

134.4.3 Front Porch Register

*Table 2412.*0x08 - FPORCH - Front porch register

31 16	15 0
VPORCH	HPORCH
0	0
rw	rw

31:16 Vertical front porch (VPORCH) - Vertical front porch in pixels.
15:0 Horisontal front porch (HPORCH) - Horizontal front porch in pixels.

134.4.4 Sync Length Register

*Table 2413.*0x0C - SYNLEN - Sync length register

31 16	15 0
VPLEN	HPLEN
0	0
rw	rw

31:16 Vertical sync pulse length (VPLEN) - Vertical sync pulse length in pixels.
15:0 Horisontal sync pulse length (HPLEN) - Horizontal sync pulse length in pixels.

134.4.5 Line Length Register

*Table 2414.*0x10 - LINLEN - Line Length register

31 16	15 0
VLLEN	HLLEN
0	0
rw	rw

31:16 Vertical line length (VLLEN) - The length of the total line with front and back porch, sync pulse length and vertical screen resolution.

15:0 Horisontal line length (HLLEN) - The length of the total line with front and back porch, sync pulse length and horizontal screen resolution,

134.4.6 Framebuffer Memory Position Register

*Table 2415.*0x14 - FBUF - Framebuffer Memory Position register

31 0
FMEM
0
rw

31:0 Framebuffer memory position (FMEM) - Holds the memory position of the framebuffer, must be aligned on a 1 Kbyte boundary.

134.4.7 Dynamic Clock 0 Register

*Table 2416.*0x18 - DCLK0 - Dynamic clock 0 register

31 0
CLK0
*
r

31:0 Dynamic pixel clock 0 (CLK0) - Dynamic pixel clock defined in ps.

134.4.8 Dynamic Clock 1 Register

*Table 2417.*0x1C - DCLK1 - Dynamic clock 1 register

31 0
CLK1
*
r

31:0 Dynamic pixel clock 1 (CLK1) - Dynamic pixel clock defined in ps.

134.4.9 Dynamic Clock 2 Register

*Table 2418.*0x20 - DCLK2 - Dynamic clock 2 register

31 0
CLK2
*
r

31:0 Dynamic pixel clock 2 (CLK2) - Dynamic pixel clock defined in ps.

134.4.10Dynamic Clock 3 Register

*Table 2419.*0x24 - DCLK3 - Dynamic clock 3 register

31	0
CLK3	
*	
e	

31:0 Dynamic pixel clock 3 (CLK3) - Dynamic pixel clock defined in ps.

134.4.11CLUTA Access Register

*Table 2420.*0x28 - CLUT - CLUT Access register

31 24	23 16	15 8	7 0
CREG	RED	GREEN	BLUE
NR	NR	NR	NR
w	w	w	w

31:24 Color lookup table register (CREG) - Color lookup table register to set.

23:16 Red color data (RED) - Red color data to set in the specified register.

15:8 Green color data (GREEN) - Green color data to set in the specified register.

7:0 Blue color data (BLUE) - Blue color data to set in the specified register.

134.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x063. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

134.6 Implementation

134.6.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

See also the documentation for the *asyncrst* VHDL generic.

134.6.2 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). All accesses match the width of the AHB bus, defined by the *ahbaccsz* generic.

134.7 Configuration options

Table 2421 shows the configuration options of the core (VHDL generics).

*Table 2421.*Configuration options

Generic name	Function	Allowed range	Default
length	Size of the pixel FIFO	3 - 1008	384
part	Pixel FIFO part length	1 - 336	128
memtech	Memory technology	0 - NTECH	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	12-bit MSB APB address	0 - 16#FFF#	0
pmask	APB address mask	0 - 16#FFF#	16#FFF#
hindex	AHB master index	0 - NAHBMST-1	0
hirq	Interrupt line	0 - NAHBIRQ-1	0
clk0	Period of dynamic clock 0 in ps	0- 16#FFFFFFFF#	40000
clk1	Period of dynamic clock 1 in ps	0- 16#FFFFFFFF#	20000
clk2	Period of dynamic clock 2 in ps	0- 16#FFFFFFFF#	15385
clk3	Period of dynamic clock 3 in ps	0- 16#FFFFFFFF#	0
burstlen	AHB burst length. The core will burst $2^{burstlen}$ words.	2 - 8	8
ahbaccsz	Determines the size of the AMBA accesses that the core will use when fetching data from memory.	32 - AHBDW	32
asyncrst	Use asynchronous reset for the VGA clock domain. If this generic is set to 1 the core will use the *arst* input to reset part of the registers in the VGA domain. Asynchronous reset should be used if the VGA clock is not available during system reset. If this generic is 0 the *arst* input is not used.	0 - 1	0

134.8 Signal descriptions

Table 2422 shows the interface signals of the core (VHDL ports).

*Table 2422.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	System clock	-
VGACLK	N/A	Input	Pixel clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
VGAO	HSYNC	Output	Horizontal sync	-
	VSYNC	Output	Vertical sync	-
	COMP_SYNC	Output	Composite sync	-
	BLANK	Output	Blanking	-
	VIDEO_OUT_R[7:0]	Output	Video out, red.	-
	VIDEO_OUT_G[7:0]	Output	Video out, green.	-
	VIDEO_OUT_B[7:0]	Output	Video out, blue.	-
	BITDEPTH[1:0]	Output	Value of Status register's BDSEL field	-
AHBI	*	Input	AHB master input signals	-
AHBO	*	Output	AHB master output signals	-
CLK_SEL[1:0]	N/A	Output	2-bit clock selector	-
ARST	N/A	Input	Asynchronous reset input	Low

* see GRLIB IP Library User's Manual

134.9 Library dependencies

Table 2423 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2423.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component, signals	Component and signal definitions.

134.10 Instantiation

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
library Gaisler;
use gaiser.misc.all;
.
architecture rtl of test is
signal apbi : apb_slv_in_type;
signal apbo : apb_slv_out;
signal vgao : apbvga_out_type;
signal ahbi : ahb_mst_in_type;
signal ahbo : ahb_mst_out_type;
signal clk_sel :std_logic_vector(1 downto 0));
signal clkmvga : std_logic;
begin
.
.
```

```
-- VGA Controller
  vga0 : svgactrl
  generic map(memtech => memtech, pindex => 6, paddr => 6, hindex => 6,
    clk0 => 40000, clk1 => 20000, clk2 => 15385, clk3 => 0)
  port map(rstn,clkm,clkmvga, apbi, apbo(6), vgao,ahbmi,ahbmo(6),clk_sel);
end;
```

134.11 Linux 2.6 driver

A video driver for the core is provided Snapgear Linux (-p27 and later). The proper kernel command line options must be used for the driver to detect the core. Please see the SnapGear Linux for LEON manual for further information.

135 SYNCIOTEST - Test block for synchronous I/O interfaces

135.1 Overview

This IP core is a helper block for instantiation into other IP cores, in order to simplify clock-to-out and setup/hold production testing on ASICs. It needs to be embedded into the IP in order to use the same registers as is used in the functional mode in order to get relevant timing measurement.

It currently has a few conditions on its use for implementation reasons:

- Number of inputs+bidirs needs to be 8 or more.
- Number of outputs needs to be at least 1

135.2 Operation

The modes are:

- Idle
- Output pseudo-random sequence on outputs and bidirs (pattern repeats after 255 cycles)
- Output toggle values between all-ones and all-zeros
- Output toggle between all-one, tristate, all-zero, tristate, ...
- Output toggle each bit in sequence "00101010" with other values first as all-one and then again with other outputs as all-zero.
- Input same pseudo-random sequence as in output mode above. If the wrong data is received on a byte lane, one of the outputs is set to flag an error. The PRNG is re-seeded with the input data every cycle to make the test mode self synchronizing with the input stream.

The tmodeact output controls when the test mode is activated, when high the surrounding IP core should mux in the dataout vector to the output registers and the tmodeoe signal to bidir output-enable registers.

A simple safety scheme is implemented. An inverted copy of the mode vector must be supplied on the top bits of the tmode input, if this is incorrect then no mode is activated. If the tmode signal is mapped to a register in the IP core, it can scrub the field for accidental bit-flips by clearing it when the tmode-act signal is low.

135.3 Implementation

135.3.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

135.4 Configuration options

Table 2424 shows the configuration options of the core (VHDL generics).

*Table 2424.*Configuration options

Generic name	Function	Allowed range	Default
ninputs	Number of inputs		1
noutputs	Number of outputs		1
nbidir	Number of bidirectional signals		1
dirmode	Direction mode. 0=both, 1=in-only, 2=out-only		1

135.5 Signal descriptions

Table 2425 shows the interface signals of the core (VHDL ports).

*Table 2425.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RSTN	N/A	Input	Reset	Low
DATAIN	N/A	Input	Data input	-
DATAOUT	N/A	Output	Data output	-
TMODE	N/A	Input	Test mode	High
TMODEACT	N/A	Output	Test mode activated	High
TMODEOE	N/A	Output	Test mode output enable	High

* see GRLIB IP Library User's Manual

135.6 Library dependencies

Table 2426 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2426.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	DFTLIB	Component, signals	Component declaration

136 SYNCRAM - Single-port RAM generator

136.1 Overview

SYNCRAM is a single port RAM that maps on technology-specific RAM blocks. The core has a common address bus, and separate data-in and data-out buses. All inputs are latched on the on the rising edge of clk. The read data appears on dataout directly after the clk rising edge.

136.2 Configuration options

Table 2427 shows the configuration options of the core (VHDL generics).

*Table 2427.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	see table below	-
dbits	Data width	see table below	-
testen	Enable bypass logic for scan testing. Scan test support is further described in the next section and in grlib.pdf, section 5.9.	0 - 1	0
custombits	Determines the width of the signals *customin* and *customout* used for MBIST. Memory BIST is further described in grlib.pdf, section 5.10.		1
pipeline	Adds pipeline registers on data outputs. For technologies supporting pipeline registers natively in the memories (e.g. PolarFire or RTG4) it determines the number of output stages. For other cases, a pipeline register is added to the output if this generic is set to a value other than 0, which adds one clock cycle of latency.	0 - 15	0
rdhold	Read-hold support. If enabled, the data previously read from the memory is retained even if *enable* is no longer being asserted. If disabled, the data read from the memory is undefined when not asserting *enable*.	0 - 1	0
gatedwr	Gated writes. If enabled, a memory write is only done if both *write* and *enable* are asserted simultaneously.	0 - 1	0

136.3 Scan test support

Scan test support will be enabled if the TESTEN generic is set to 1. This option will generate a register (flip-flops) connected between the DATAIN and DATAOUT of the syncram module. In test mode, DATAOUT is driven from the register rather then from the RAM outputs. This will allow both input and output paths around the syncram to be testable by scan. The address bus and control signals are xored with the DATAIN signal to also increase test coverage of those. Test mode is enabled by driving the TESTIN(3) signals to 1. This signal should typically be connected to the global test enable signals of the design.

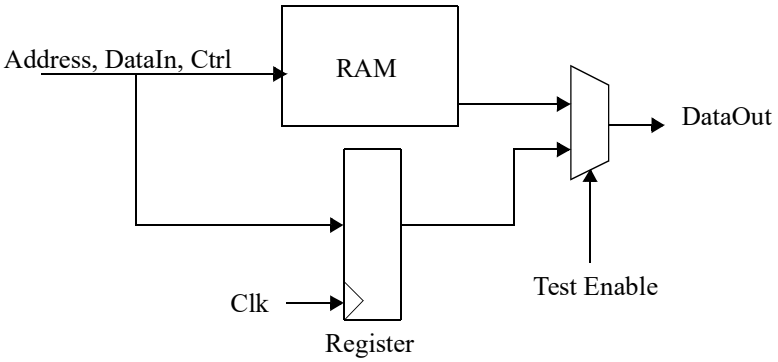


Figure 343. Scan test support

136.4 Supported technologies

Table 2428 shows the supported technologies for the core.

*Table 2428.*Supported technologies

Tech name	Technology	RAM cell	abit range	dbit range
altera	All Altera devices	altsyncram	unlimited	unlimited
ihp15	IHP 0.25	sram2k (512x32)	2 - 9	unlimited
inferred	Behavioral description	Tool dependent	unlimited	unlimited
virtex	Xilinx Virtex, VirtexE, Spartan2	RAMB4_Sn	unlimited	unlimited
virtex2, virtex4, virtex5, spartan3, spartan6, virtex7, kintex7, artix7, zynq7000, kintexu, versal	Xilinx Virtex2/4/5/6, Spartan3/3a/3e/6, 7-Series, Ultrascale, Ultrascale+, Versal	RAMB16_Sn	unlimited	unlimited
axcel / axdsp	Actel AX, RTAX and RTAX-DSP	RAM64K36	2 - 12	unlimited
proasic	Actel Proasic	RAM256x9SST	2 - 14	unlimited
proasic3	Actel Proasic3	ram4k9, ram512x18	2 - 12	unlimited
lattice	Lattice XP/EC/ECP	sp8ka	2 - 13	unlimited
memvirage	Virage ASIC RAM	hdss1_128x32cm4sw0 hdss1_256x32cm4sw0 hdss1_512x32cm4sw0 hdss1_1024x32cm8sw0	7 - 11	32
memartisan	Artisan ASIC RAM	sp_256x32m32 sp_512x32m32 sp_1kx32m32 sp_2kx32m32 sp_4kx32m32 sp_8kx32m32 sp_16kx32m32	8 - 14	32
memvirage90	Virage 90 nm ASIC RAM	SPRAM_HS_32x30 SPRAM_HS_128x32 SPRAM_HS_256x32 SPRAM_HS_1024x32	2 - 10	128

*Table 2428.*Supported technologies

Tech name	Technology	RAM cell	abit range	dbit range
eclipse	Aeroflex/Quicklogic FPGA	RAM128x18_25um RAM256X9_25um RAM512X4_25um RAM1024X2_25um	2 - 10	unlimited
easic90	eASIC 90 nm Nextreme	eram, bram	2 - 15	unlimited
easic45	eASIC 45 nm Nextreme2	bRAM, rFile	unlimited	unlimited
igloo2 / smartfusion2	Microchip IGLOO2 / SmartFusion2	RAM1K18	2 - 14	unlimited
rtg4	Microchip RTG4	RAM1K18_RT	2 - 16	unlimited
polarfire	Microchip PolarFire	RAM1K20 RAM64x12	unlimited	unlimited
nx	NanoXplore BRAVE FPGAs	NX_RAM	2 - 12	unlimited
nexus	Lattice Nexus family: Crosslink / Certus / CertusPro	SP16K	2 - 17	unlimited

136.5 Signal descriptions

Table 2429 shows the interface signals of the core (VHDL ports).

*Table 2429.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock. All input signals are latched on the rising edge of the clock.	-
ADDRESS	N/A	Input	Address bus. Used for both read and write access.	-
DATAIN	N/A	Input	Data inputs for write data	-
DATAOUT	N/A	Output	Data outputs for read data	-
ENABLE	N/A	Input	Chip select	High
WRITE	N/A	Input	Write enable	High
TESTIN	N/A	Input	Test inputs (see text)	High
CUSTOMCLK	N/A	Input	Memory BIST input clock	-
CUSTOMIN	N/A	Input	Memory BIST input data	-
CUSTOMOUT	N/A	Output	Memory BIST output data	-

136.6 Library dependencies

Table 2430 shows libraries used when instantiating the core (VHDL libraries).

*Table 2430.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Constants	Technology constants

136.7 Component declaration

The core has the following component declaration.

```
library techmap;
use techmap.gencomp.all;

  component syncram
```

```
  generic (tech : integer := 0; abits : integer := 6; dbits : integer := 8);
  port (
    clk      : in std_ulogic;
    address  : in std_logic_vector((abits -1) downto 0);
    datain   : in std_logic_vector((dbits -1) downto 0);
    dataout  : out std_logic_vector((dbits -1) downto 0);
    enable   : in std_ulogic;
    write    : in std_ulogic;
    testin   : in std_logic_vector(3 downto 0) := "0000");
  end component;
```

136.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;
.

clk      : std_ulogic;
address  : std_logic_vector((abits -1) downto 0);
datain   : std_logic_vector((dbits -1) downto 0);
dataout  : std_logic_vector((dbits -1) downto 0);
enable   : std_ulogic;
write    : std_ulogic);

ram0 : syncram generic map ( tech => tech, abits => addrbits, dbits => dbits)
      port map ( clk, addr, datain, dataout, enable, write);
```

137 SYNCRAMBW - Single-port RAM generator with byte enables

137.1 Overview

SYNCRAMBW implements a single port RAM with byte enables, using the GRLIB technology wrapping for different target technologies. The core operates identically to SYNCRAM, with the addition that each byte has a separate chip select (ENABLE) and write select (WRITE). The core is provided in a generic configuration and also in configurations of 128, 156 and 256 bits, and the corresponding entities are named SYNCRAMBW, SYNCRAM128BW, SYNCRAM156BW and SYNCRAM256BW. In the simplest case, the IP cores just instantiate several eight bit wide SYNCRAM components. SYNCRAM128BW, SYNCRAM156BW and SYNCRAM256BW, used in GRLIB's Level-2 cache core, contain specialized maps for several technologies to more efficiently utilize device resources.

Note that some SYNCRAM components may be missing from the library depending on the type of GRLIB distribution.

137.2 Configuration options

Table 2431 shows the configuration options of the core (VHDL generics).

*Table 2431.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	see table below	-
testen	Enable bypass logic for scan testing. Scan test support is further described in the next section and in grlib.pdf, section 5.9.	0 - 1	0
custombits	Determines the width of the signals *customin* and *customout* used for MBIST. Memory BIST is further described in grlib.pdf, section 5.10.		1
pipeline	Adds pipeline registers on data outputs. For technologies supporting pipeline registers natively in the memories (e.g. PolarFire or RTG4) it determines the number of output stages. For other cases, a pipeline register is added to the output if this generic is set to a value other than 0, which adds one clock cycle of latency.	0 - 15	0

137.3 Scan test support

Scan test support will be enabled if the TESTEN generic is set to 1. This option will generate a register (flip-flops) connected between the DATAIN and DATAOUT of the syncram module. In test mode, DATAOUT is driven from the register rather then from the RAM outputs. This will allow both input and output paths around the syncram to be testable by scan. The address bus and control signals are xored with the DATAIN signal to also increase test coverage of those. Test mode is enaled by driving the TESTIN(3) signals to 1. This signal should typically be connected to the global test enable signals of the design.

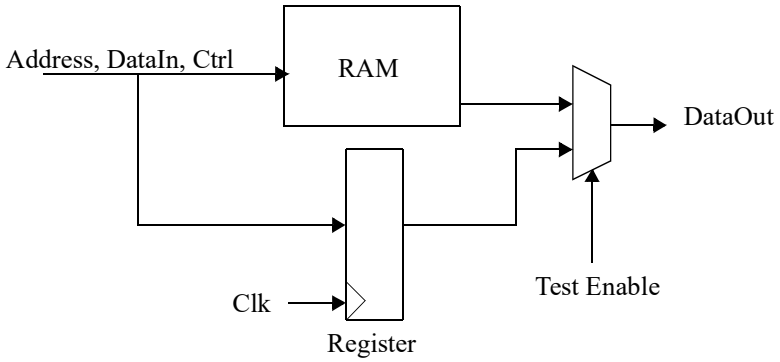


Figure 344. Scan test support

137.4 Technology support

Table 2432 shows the supported technologies for the core.

*Table 2432.*Supported technologies

Tech name	Technology	RAM cell	abit range	dbit range
altera	All Altera devices	altsyncram	unlimited	unlimited
inferred	Behavioral description	Tool dependent	unlimited	unlimited
virtex2, virtex4, virtex5, spartan3, spartan6, virtex7, kintex7, artix7, zynq7000, kintexu, versal	Xilinx Virtex2/4/5/6, Spartan3/3a/3e/6, 7-Series, Ultrascale	RAMB16_Sn	unlimited	unlimited
all others	-	syncram core with dwidth=8	tech depend.	tech depend.

To add support for a new technology, the following steps should be taken:

- Add technology-specific version for the RAM core in lib/techmap/TECH
- Instantiate the technology-specific RAM core in lib/techmap/maps/syncram256bw.vhd, and set the has_sram256bw() constant to 1 for the specific technology:

```
constant has_sram256bw : tech_ability_type := (
virtex2 => 1, virtex4 => 1, virtex5 => 1, spartan3 => 1,
spartan3e => 1, spartan6 => 1, virtex6 => 1,
altera => 1, cyclone3 => 1, stratix2 => 1, stratix3 => 1,
tm65gpl => 0, others => 0);
```

See also syncrambw.vhd, syncram128bw.vhd and syncram156bw.vhd under lib/techmap/maps/ for the corresponding SYNCRAM BW IP cores.

137.5 Signal descriptions

Table 2433 shows the interface signals of the core (VHDL ports).

*Table 2433.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock. All input signals are latched on the rising edge of the clock.	-
ADDRESS	N/A	Input	Address bus. Used for both read and write access.	-
DATAIN	N/A	Input	Data inputs for write data	-
DATAOUT	N/A	Output	Data outputs for read data	-
ENABLE	N/A	Input	Byte Chip select	High
WRITE	N/A	Input	Byte Write enable	High
TESTIN		Input	Test inputs (see text)	High
CUSTOMCLK	N/A	Input	Memory BIST input clock	-
CUSTOMIN	N/A	Input	Memory BIST input data	-
CUSTOMOUT	N/A	Output	Memory BIST output data	-

137.6 Library dependencies

Table 2434 shows libraries used when instantiating the core (VHDL libraries).

*Table 2434.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Constants	Technology contants

137.7 Component declaration

The core has the following component declaration.

```
library techmap;
use techmap.gencomp.all;

  component syncram_bw128
  generic (tech : integer := 0; abits : integer := 6);
  port (
    clk      : in std_ulogic;
    address  : in std_logic_vector((abits -1) downto 0);
    datain   : in std_logic_vector(127 downto 0);
    dataout  : out std_logic_vector(127 downto 0);
    enable   : in std_logic_vector(15 downto 0);
    write    : in std_logic_vector(15 downto 0);
    testin   : in std_logic_vector(3 downto 0) := "0000");
  end component;

  component syncram_bw256
  generic (tech : integer := 0; abits : integer := 6);
  port (
    clk      : in std_ulogic;
    address  : in std_logic_vector((abits -1) downto 0);
    datain   : in std_logic_vector(255 downto 0);
    dataout  : out std_logic_vector(255 downto 0);
    enable   : in std_logic_vector(31 downto 0);
    write    : in std_logic_vector(31 downto 0);
    testin   : in std_logic_vector(3 downto 0) := "0000");
  end component;
```

137.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;
.

clk      : std_ulogic;
address  : std_logic_vector(9 downto 0);
datain   : std_logic_vector(255 downto 0);
dataout  : std_logic_vector(255 downto 0);
enable   : std_logic_vector(31 downto 0);
write    : std_logic_vector(31 downto 0);

ram0 : syncram generic map ( tech => tech, abits => 10)
      port map ( clk, addr, datain, dataout, enable, write);
```

138 SYNCRAM_2P - Two-port RAM generator

138.1 Overview

The two-port RAM generator has a one read port and one write port. Each port has a separate address and data bus. All inputs are registered on the rising edge of clk. The read data appears on dataout directly after the clk rising edge. Address width, data width and target technology is parametrizable through generics.

138.2 Write-through operation

Write-through is supported if the function *syncram_2p_write_through(tech)* returns 1 for the target technology, or if the *wrfst* generic is set to 1. If *wrfst* = 1, additional logic will be generated to detect simultaneous read/write to the same memory location, and in that case bypass the written data to the data outputs.

138.3 Conflicts

Some technologies will produce unpredictable results when a read and write operation occurs simultaneously to the same memory location. The function *syncram_2p_dest_rw_collision(tech)* returns 1 for technologies that has this characteristic. If SYNCRAM_2P is implemented with *sepclk* = 0 then logic will be included that disables the read enable signal, if needed, when a collision is detected. If the core is implemented with *sepclk* = 1 (and *syncram_2p_dest_rw_collision(tech)* returns 1) then collision avoidance must be handled by external logic.

138.4 Scan test support

Scan test support will be enabled if the TESTEN generic is set to 1. This option will generate a register (flip-flops) connected between the DATAIN and DATAOUT of the syncram module. In test mode, DATAOUT is driven from the register rather then from the RAM outputs. This will allow both input and output paths around the syncram to be testable by scan. The address bus and control signals are xored with the DATAIN signal to also increase test coverage of those. Test mode is enaled by driving the TESTIN(3) signals to 1. This signal should typically be connected to the global test enable signals of the design.

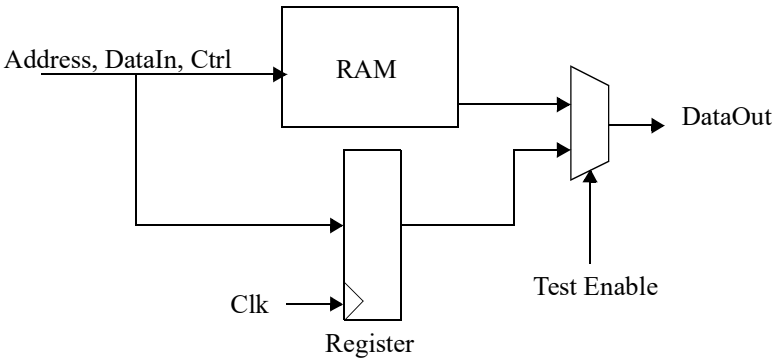


Figure 345. Scan test support

138.5 Configuration options

Table 2435 shows the configuration options of the core (VHDL generics).

*Table 2435.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	see table below	-
dbits	Data width	see table below	-
sepclk	If 1, separate clocks (rclk/wclk) are used for the two ports. If 0, wclk is used for both ports.	0 - 1	0
wrfst	Enable bypass logic for write-through operation. Can only be enabled for sepclk = 0.	0 - 1	0
testen	Enable bypass logic for scan testing. Scan test support is further described in the next section and in grlib.pdf, section 5.9.	0 - 1	0
words	Deprecated - do not use	0	0
custombits	Determines the width of the signals *customin* and *customout* used for MBIST. Memory BIST is further described in grlib.pdf, section 5.10.		1
pipeline	Adds pipeline registers on data outputs. For technologies supporting pipeline registers natively in the memories (e.g. PolarFire) it determines the number of output stages. For other cases, a pipeline register is added to the output if this generic is set to a value other than 0, which adds one clock cycle of latency.	0 - 15	0
rdhold	Read-hold support. If enabled, the data previously read from the memory is retained even if *enable* is no longer being asserted. If disabled, the data read from the memory is undefined when not asserting *enable*.	0 - 1	0

Table 2436 shows the supported technologies for the core.

*Table 2436.*Supported technologies

Tech name	Technology	RAM cell	abit range	dbit range
Inferred	Behavioural description	Tool dependent	unlimited	unlimited
altera	All Altera devices	altsyncram	unlimited	unlimited
virtex	Xilinx Virtex, Virtex-E, Spartan-2	RAMB4_Sn	2 - 10	unlimited
virtex2, virtex4, virtex5, spartan3, spartan6, virtex7, kintex7, artix7, zynq7000, kintexu, versal	Xilinx Virtex2/4/5/6, Spartan3/3a/3e/6, 7-Series, Ultrascale, Ultrascale+, Versal	RAMB16_Sn	2 - 14	unlimited
axcel / axdsp	Actel AX, RTAX and RTAX-DSP	RAM64K36	2 - 12	unlimited
proasic	Actel Proasic	RAM256x9SST	2 - 14	unlimited
proasic3	Actel Proasic3	ram4k9, ram512x18	2 - 12	unlimited
lattice	Lattice XP/EC/ECP	dp8ka	2 - 13	unlimited
memvirage	Virage ASIC RAM	hdss2_64x32cm4sw0 hdss2_128x32cm4sw0 hdss2_256x32cm4sw0 hdss2_512x32cm4sw0	6 - 9	32
memartisan	Artisan ASIC RAM	rf2_256x32m4 rf2_512x32m4	8 - 9	32
eclipse	Aeroflex/Quicklogic FPGA	RAM128x18_25um RAM256X9_25um RAM512X4_25um RAM1024X2_25um	2 - 10	unlimited
easic90	eASIC 90 nm Nextreme	eram	2 - 12	unlimited
easic45	eASIC 45 nm Nextreme2	bRAM, rFile	unlimited	unlimited
igloo2 / smartfusion2	Microsemi IGLOO2 / SmartFusion2	RAM1K18	2 - 14	unlimited
rtg4	Microsemi RTG4	RAM1K18_RT	2 - 16	unlimited
polarfire	Microchip PolarFire	RAM1K20 RAM64x12		
nx	NanoXplore BRAVE FPGAs	NX_RAM	2 - 12	unlimited
nexus	Lattice Nexus family: Crosslink / Certus / CertusPro	DP16K	2 - 17	unlimited

138.6 Signal descriptions

Table 2437 shows the interface signals of the core (VHDL ports).

*Table 2437.*Signal descriptions

Signal name	Type	Function	Active
RCLK	Input	Read port clock	-
RENABLE	Input	Read enable	High
RADDRESS	Input	Read address bus	-
DATAOUT	Output	Data outputs for read data	-
WCLK	Input	Write port clock	-
WRITE	Input	Write enable	High
WADDRESS	Input	Write address	-
DATAIN	Input	Write data	-
TESTIN	Input	Test inputs (see text)	High
CUSTOMCLK	Input	Memory BIST input clock	-
CUSTOMIN	Input	Memory BIST input data	-
CUSTOMOUT	Output	Memory BIST output data	-

138.7 Library dependencies

Table 2438 shows libraries used when instantiating the core (VHDL libraries).

*Table 2438.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Constants	Technology contants

138.8 Component declaration

The core has the following component declaration.

```
library techmap;
use techmap.gencomp.all;

component syncram_2p
  generic (tech : integer := 0; abits : integer := 6; dbits : integer := 8; sepclk : integer
:= 0);
  port (
    rclk     : in std_ulogic;
    renable  : in std_ulogic;
    raddress : in std_logic_vector((abits -1) downto 0);
    dataout  : out std_logic_vector((dbits -1) downto 0);
    wclk     : in std_ulogic;
    write    : in std_ulogic;
    waddress : in std_logic_vector((abits -1) downto 0);
    datain   : in std_logic_vector((dbits -1) downto 0);
    testin   : in std_logic_vector(3 downto 0) := "0000");
  end component;
```

138.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
```

```
use techmap.gencomp.all;

rclk     : in std_ulogic;
renable  : in std_ulogic;
raddress : in std_logic_vector((abits -1) downto 0);
dataout  : out std_logic_vector((dbits -1) downto 0);
wclk     : in std_ulogic;
write    : in std_ulogic;
waddress : in std_logic_vector((abits -1) downto 0);
datain   : in std_logic_vector((dbits -1) downto 0));

ram0 : syncram_2p generic map ( tech => tech, abits => addrbits, dbits => dbits)
      port map ( rclk, renable, raddress, dataout, wclk, write, waddress, datain, enable,
write);
```

139 SYNCRAM_DP - Dual-port RAM generator

139.1 Overview

The dual-port RAM generator has two independent read/write ports. Each port has a separate address and data bus. All inputs are latched on the on the rising edge of clk. The read data appears on dataout directly after the clk rising edge. Address width, data width and target technology is parametrizable through generics. Simultaneous write to the same address is technology dependent, and generally not allowed.

139.2 Configuration options

Table 2439 shows the configuration options of the core (VHDL generics).

*Table 2439.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	see table below	-
dbits	Data width	see table below	-
testen	Enable bypass logic for scan testing. Scan test support is further described in the next section and in grlib.pdf, section 5.9.	0 - 1	0
custombits	Determines the width of the signals *customin* and *customout* used for MBIST. Memory BIST is further described in grlib.pdf, section 5.10.		1
sepclk	If 1, separate clocks are used for the two ports.	0 - 1	0
wrfst	Enable bypass logic for write-through operation. Can only be enabled for sepclk = 0.	0 - 1	0
pipeline	Adds pipeline registers on data outputs. For technologies supporting pipeline registers natively in the memories (e.g. PolarFire or RTG4) it determines the number of output stages. For other cases, a pipeline register is added to the output if this generic is set to a value other than 0, which adds one clock cycle of latency.	0 - 15	0
rdhold	Read-hold support. If enabled, the data previously read from the memory is retained even if *enable* is no longer being asserted. If disabled, the data read from the memory is undefined when not asserting *enable*.	0 - 1	0
gatedwr	Gated writes. If enabled, a memory write is only done if both *write* and *enable* are asserted simultaneously.	0 - 1	0

139.3 Supported technologies

Table 2440 shows the supported technologies for the core.

*Table 2440.*Supported technologies

Tech name	Technology	RAM cell	abit range	dbit range
altera	All altera devices	altsyncram	unlimited	unlimited
virtex	Xilinx Virtex, Virtex-E, Spartan-2	RAMB4_Sn	2 - 10	unlimited

Table 2440. Supported technologies

Tech name	Technology	RAM cell	abit range	dbit range
virtex2, virtex4, virtex5, spartan3, spartan6, virtex7, kintex7, artix7, zynq7000, kintexu, versal	Xilinx Virtex2/4/5/6, Spartan3/3a/3e/6, 7-Series, Ultrascale, Ultrascale+, Versal	RAMB16_Sn	2 - 14	unlimited
proasic3	Actel Proasic3	ram4k9	2 - 12	unlimited
lattice	Lattice XP/EC/ECP	dp8ka	2 - 13	unlimited
memvirage	Virage ASIC RAM	hdss2_64x32cm4sw0 hdss2_128x32cm4sw0 hdss2_256x32cm4sw0 hdss2_512x32cm4sw0	6 - 9	32
memartisan	Artisan ASIC RAM	dp_256x32m4 dp_512x32m4 dp_1kx32m4	8 - 10	32
memvirage90	Virage 90 nm ASIC RAM	DPRAM_HS_256x20 DPRAM_HS_256x32	2 - 8	128
easic45	eASIC 45 nm Nextreme2	bRAM	unlimited	unlimited
igloo2 / smartfusion2	Microsemi IGLOO2 / SmartFusion2	RAM1K18	2 - 14	unlimited
rtg4	Microsemi RTG4	RAM1K18_RT	2 - 16	unlimited
polarfire	Microchip PolarFire	RAM1K20	unlimited	unlimited
nx	NanoXplore BRAVE FPGAs	NX_RAM	2 - 12	unlimited
nexus	Lattice Nexus family: Crosslink / Certus / CertusPro	DP16K	2 - 17	unlimited

139.4 Signal descriptions

Table 2441 shows the interface signals of the core (VHDL ports).

Table 2441. Signal descriptions

Signal name	Field	Type	Function	Active
CLK1	N/A	Input	Port1 clock	-
ADDRESS1	N/A	Input	Port1 address	-
DATAIN1	N/A	Input	Port1 write data	-
DATAOUT1	N/A	Output	Port1 read data	-
ENABLE1	N/A	Input	Port1 chip select	High
WRITE1	N/A	Input	Port 1 write enable	High
CLK2	N/A	Input	Port2 clock	-
ADDRESS2	N/A	Input	Port2 address	-
DATAIN2	N/A	Input	Port2 write data	-
DATAOUT2	N/A	Output	Port2 read data	-
ENABLE2	N/A	Input	Port2 chip select	High
WRITE2	N/A	Input	Port 2 write enable	High
TESTIN	N/A	Input	Test inputs (see text)	High
CUSTOMCLK	N/A	Input	Memory BIST input clock	-
CUSTOMIN	N/A	Input	Memory BIST input data	-
CUSTOMOUT	N/A	Output	Memory BIST output data	-

139.5 Library dependencies

Table 2442 shows libraries used when instantiating the core (VHDL libraries).

*Table 2442.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Constants	Technology contants

139.6 Component declaration

The core has the following component declaration.

```
library techmap;
use techmap.gencomp.all;

component syncram_dp
  generic (tech : integer := 0; abits : integer := 6; dbits : integer := 8);
  port (
    clk1     : in std_ulogic;
    address1 : in std_logic_vector((abits -1) downto 0);
    datain1  : in std_logic_vector((dbits -1) downto 0);
    dataout1 : out std_logic_vector((dbits -1) downto 0);
    enable1  : in std_ulogic;
    write1   : in std_ulogic;
    clk2     : in std_ulogic;
    address2 : in std_logic_vector((abits -1) downto 0);
    datain2  : in std_logic_vector((dbits -1) downto 0);
    dataout2 : out std_logic_vector((dbits -1) downto 0);
    enable2  : in std_ulogic;
    write2   : in std_ulogic);
  end component;
```

139.7 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;

clk1     : in std_ulogic;
address1 : in std_logic_vector((abits -1) downto 0);
datain1  : in std_logic_vector((dbits -1) downto 0);
dataout1 : out std_logic_vector((dbits -1) downto 0);
enable1  : in std_ulogic;
write1   : in std_ulogic;
clk2     : in std_ulogic;
address2 : in std_logic_vector((abits -1) downto 0);
datain2  : in std_logic_vector((dbits -1) downto 0);
dataout2 : out std_logic_vector((dbits -1) downto 0);
enable2  : in std_ulogic;
write2   : in std_ulogic);

ram0 : syncram_dp generic map ( tech => tech, abits => addrbits, dbits => dbits)
      port map ( clk1, address1, datain1, dataout1, enable1, write1, clk2, address2, datain2,
dataout2, enable2, write2);
```

140 SYNCRAMFT - Single-port RAM generator with EDAC

140.1 Overview

SYNCRAMFT is a single port RAM that maps on technology-specific RAM blocks. The core has a common address bus, and separate data-in and data-out buses. All inputs are latched on the on the rising edge of clk. The read data appears on dataout directly after the clk rising edge. SYNCRAMFT can be configured to include logic for error detection or error detection and correction. This protection is added either as logic connected to an internal SYNCRAM instantiation or by mapping directly to technology specific memory resources with SECDED EDAC.

140.2 Configuration options

Table 2443 shows the configuration options of the core (VHDL generics).

*Table 2443.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	-	-
dbits	Data width	-	-
ft	Fault-tolerance 0: Standard SYNCRAM, no FT 1: byte parity DMR 2: true TMR 3: byte-parity, no DMR (only error detection) 4: SECDED - BCH 5: SECDED - target technology specific Technology specific protection is further documented in the GRLIB-FT User's Manual (grlib-ft.pdf).	0 - 5	0
testen	Enable bypass logic for scan testing. Scan test support is further described in the next section and in grlib.pdf, section 5.9.	0 - 1	0
custombits	Determines the width of the signals *customin* and *customout* used for MBIST. Memory BIST is further described in grlib.pdf, section 5.10.		1
pipeline	Adds pipeline registers on data outputs. Bits 3:0 of is a field that adds registers to the data outputs of the RAM cell. This adds one additional clock cycle before the read data is available on the SYNCRAM data outputs. Bits 7:4 is a field that adds a register after the error correction logic. This adds one additional clock cycle before the read data is available on the data outputs. The error signals are delayed in the same way as the data. This has the following effect on data read latency Pipeline value: added latency 0 : 0 1 : 1 16: 1 17: 2	0 - 255	0

*Table 2443.*Configuration options

Name	Function	Range	Default
rdhold	Read-hold support. If enabled, the data previously read from the memory is retained even if *enable* is no longer being asserted. If disabled, the data read from the memory is undefined when not asserting *enable*.	0 - 1	0
gatedwr	Gated writes. If enabled, a memory write is only done if both *write* and *enable* are asserted simultaneously.	0 - 1	0

140.3 Scan test support

See SYNCRAM documentation.

140.4 Signal descriptions

Table 2444 shows the interface signals of the core (VHDL ports).

*Table 2444.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock. All input signals are latched on the rising edge of the clock.	-
ADDRESS	N/A	Input	Address bus. Used for both read and write access.	-
DATAIN	N/A	Input	Data inputs for write data	-
DATAOUT	N/A	Output	Data outputs for read data	-
ENABLE	N/A	Input	Chip select	High
WRITE	N/A	Input	Write enable	High
ERROR	N/A	Output	Different behaviour depending on setting of VHDL generic ft: 0 .. 3: Error output has one position per byte 4 .. 5: Error output has two positions: 0:CERR, 1: UCERR. CERR is asserted for correctable error. UCERR is asserted for uncorrectable errors and overrides CERR.	High
TESTIN	N/A	Input	Test inputs (see text)	High
ERRINJ	N/A	Input	Different behaviour depending on setting of VHDL generic ft: errinj bits (((dbits + 7)/8)*2-1 downto 0) are used for byte parity DMR (FT = 1) errinj cannot currently be used with FT = 2 (TMR) errinj bits (((dbits + 7)/8)-1 downto 0) are used for byte parity, no DMR (FT = 3) errinj bits 6 downto 0 are used for SECDED - BCH (FT = 4) errinj bits used for technology specific EDAC (FT = 5) depends on the target technology. For Xilinx, bits 1:0 are used. 0: inject correctable error. 1: inject uncorrectable error. For RTG4 only bit 0 is used and is used to disabled the EDAC logic.	High
CUSTOMCLK	N/A	Input	Memory BIST input clock	-
CUSTOMIN	N/A	Input	Memory BIST input data	-
CUSTOMOUT	N/A	Output	Memory BIST output data	-

140.5 Library dependencies

Table 2445 shows libraries used when instantiating the core (VHDL libraries).

*Table 2445.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Constants	Technology constants

141 TAP - JTAG TAP Controller

141.1 Overview

JTAG TAP Controller provides an Test Access Port according to IEEE-1149 (JTAG) Standard. The core implements the Test Access Port signals, the synchronous TAP state-machine, a number of JTAG data registers (depending on the target technology) and an interface to user-defined JTAG data registers.

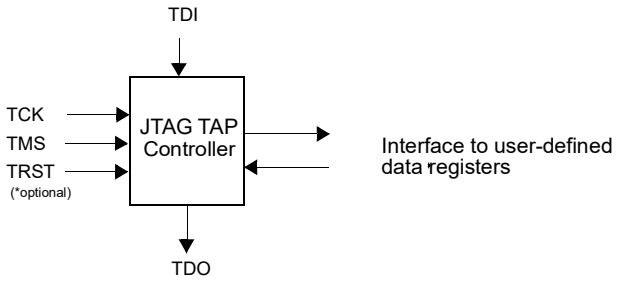


Figure 346. TAP controller block diagram

141.2 Operation

141.2.1 Generic TAP Controller

The generic TAP Controller implements JTAG Test Access Point interface with signals TCK, TMS, TDI and TDO, a synchronous state-machine compliant to the IEEE-1149 standard, JTAG instruction register and two JTAG data registers: bypass and device identification code register. The core is capable of shifting and updating the JTAG instruction register, putting the device into bypass mode (BYPASS instruction) and shifting out the devices identification number (IDCODE instruction). User-defined JTAG test registers are accessed through user-defined data register interface.

The access to the user-define test data registers is provided through the user-defined data register interface. The instruction in the TAP controller instruction register appears on the interface as well as shift-in data and signals indicating that the TAP controller is in Capture-Data-Register, Shift-Data-Register or Update-Data-Register state. Logic controlling user-defined data registers should observe value in the instruction register and TAP controller state signals in order to capture data, shift data or update data-registers.

JTAG test registers such as boundary-scan register can be interfaced to the TAP controller through the user data register interface.

141.3 Technology specific TAP controllers

The core instantiates technology specific TAP controller for Altera and Xilinx devices.

141.4 Registers

The core implements three JTAG registers: instruction, bypass and device identification code register.

141.5 Vendor and device identifiers

The core does not have vendor and device identifiers since it does not have AMBA interfaces.

141.6 Implementation

141.6.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). If the VHDL generic trsten is non-zero then the corer makes use of asynchronous reset. Otherwise synchronous reset is used.

141.7 Configuration options

Table 2446 shows the configuration options of the core (VHDL generics).

*Table 2446.*Configuration options

Generic	Function	Allowed range	Default
tech	Target technology	0 - NTECH	0
irlen	Instruction register length (generic tech only)	2 - 8	4
idcode	JTAG IDCODE instruction code(generic tech only)	0 - 255	9
manf	Manufacturer id. Appears as bits 11-1 in TAP controllers device identification register. Used only for generic technology. Default is Frontgrade Gaisler manufacturer id.	0 - 2047	804
part	Part number (generic tech only). Bits 27-12 in device id. reg.	0 - 65535	0
ver	Version number (generic tech only). Bits 31-28 in device id. reg.	0-15	0
trsten	Support optional TRST signal (generic tech only)	0 - 1	1
scantest	Enable scan test support	0 - 1	0
oepol	Polarity for TDOEN signal	0 - 1	1
tcknen	Support externally inverted TCK signal (generic tech only)	0 - 1	0

141.8 Signal descriptions

Table 2447 shows the interface signals of the core (VHDL ports).

*Table 2447.*Signal declarations

Signal name	Field	Type	Function	Active
TRST	N/A	Input	JTAG TRST signal*	Low
TCK	N/A	Input	JTAG clock*	-
TMS	N/A	Input	JTAG TMS signal*	High
TDI	N/A	Input	JTAG TDI signal*	High
TDO	N/A	Output	JTAG TDO signal*	High
User-defined data register interface				
TAPO_TCK	N/A	Output	TCK signal	High
TAPO_TDI	N/A	Output	TDI signal	High
TAPO_INST[7:0]	N/A	Output	Instruction in the TAP Ctrl instruction register	High
TAPO_RST	N/A	Output	TAP Controller in Test-Logic_Reset state	High
TAPO_CAPT	N/A	Output	TAP Controller in Capture-DR state	High
TAPO_SHFT	N/A	Output	TAP Controller in Shift-DR state	High
TAPO_UPD	N/A	Output	TAP Controller in Update-DR state	High
TAPO_XSEL1	N/A	Output	Xilinx User-defined Data Register 1 selected (Xilinx tech only)	High
TAPO_XSEL2	N/A	Output	Xilinx User-defined Data Register 2 selected (Xilinx tech only)	High
TAPI_EN1	N/A	Input	Enable shift-out data port 1 (TAPI_TDO1), when disabled data on port 2 is used	High
TAPI_TDO1	N/A	Input	Shift-out data from user-defined register port 1	High
TAPI_TDO2	N/A	Input	Shift-out data from user-defined register port 2	High
TAPO_NINST	N/A	Output	Instruction to be written into TAP Ctrl instruction register, only valid when TAPO_IUPD is high. (Generic tech only)	High
TAPO_IUPD	N/A	Output	TAP Controller in Update-IR state (Generic tech only)	High
TAPO_TCKN	N/A	Outpu	Inverted TCK signal	High
Additional signals				
TESTEN	N/A	Input	Test mode enable signal	High
TESTRST	N/A	Input	Test mode reset signal	Low
TESTOEN	N/A	Input	Test mode output-enable control	see oepol
TDOEN	N/A	Output	JTAG TDO enable signal*	see oepol
TCKN	N/A	Input	Inverted clock in (if tcknen generic is set)	

*) If the target technology is Xilinx or Altera the cores JTAG signals TCK, TCKN, TMS, TDI and TDO are not used. Instead the dedicated FPGA JTAG pins are used. These pins are implicitly made visible to the core through technology-specific TAP macro instantiation.

141.9 Library dependencies

Table 2448 shows libraries used when instantiating the core (VHDL libraries).

*Table 2448.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Component	TAP Controller component declaration

141.10 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library techmap;
use gaisler.gencomp.all;

entity tap_ex is
  port (
    clk : in std_ulogic;
    rst : in std_ulogic;

    -- JTAG signals
    tck  : in std_ulogic;
    tms  : in std_ulogic;
    tdi  : in std_ulogic;
    tdo  : out std_ulogic
);
end;

architecture rtl of tap_ex is

signal gnd : std_ulogic;

signal tapo_tck, tapo_tdi, tapo_rst, tapo_capt : std_ulogic;
signal tapo_shft, tapo_upd : std_ulogic;
signal tapi_en1, tapi_tdo : std_ulogic;
signal tapo_inst : std_logic_vector(7 downto 0);


begin

 gnd <= '0';
 tckn <= not tck;


-- TAP Controller

  tap0 : tap (tech => 0)
    port map (rst, tck, tckn, tms, tdi, tdo, open, tapo_tck, tapo_tdi, tapo_inst,
      tapo_rst, tapo_capt, tapo_shft, tapo_upd, open, open,
       tapi_en1, tapi_tdo, gnd);


-- User-defined JTAG data registers

  ...


end;
```

142 GRTM - CCSDS/ECSS Telemetry Encoder

142.1 Overview

The CCSDS/ECSS/PSS Telemetry Encoder implements part of the Data Link Layer, covering the Protocol Sub-layer and the Frame Synchronization and Coding Sub-layer and part of the Physical Layer of the packet telemetry encoder protocol.

The operation of the Telemetry Encoder is highly programmable by means of control registers. The design of the Telemetry Encoder is highly configurable by means of VHDL generics.

The Telemetry Encoder comprises several encoders and modulators implementing the Consultative Committee for Space Data Systems (CCSDS) recommendations, European Cooperation on Space Standardization (ECSS) and the European Space Agency (ESA) Procedures, Standards and Specifications (PSS) for telemetry and channel coding. The Telemetry Encoder comprises the following:

- Packet Telemetry and/or Advanced Orbiting Systems (AOS) Encoder
- Reed-Solomon Encoder
- Turbo Encoder (future option)
- Pseudo-Randomiser (PSR)
- Non-Return-to-Zero Mark encoder (NRZ)
- Convolutional Encoder (CE)
- Split-Phase Level modulator (SP)
- Sub-Carrier modulator (SC)
- Clock Divider (CD)

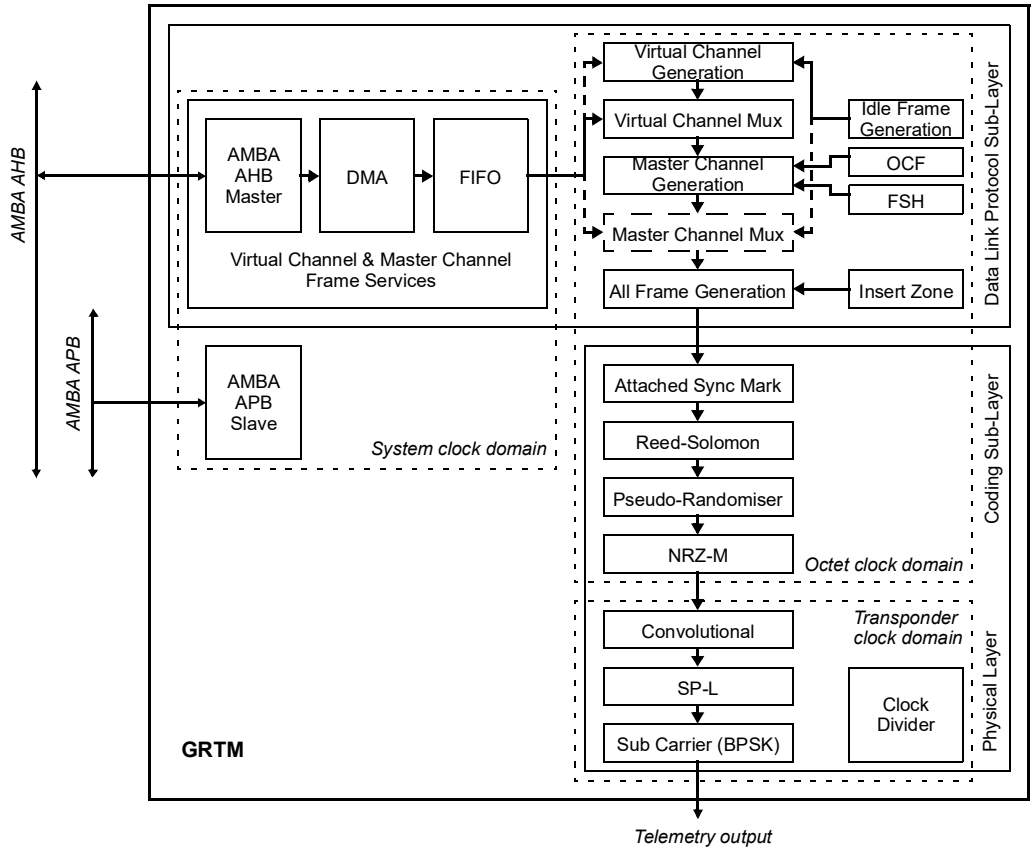


Figure 347. Block diagram

142.2 References

142.2.1 Documents

[CCSDS-131.0-B-2] TM Synchronization and Channel Coding

[CCSDS-132.0-B-1] TM Space Data Link Protocol

[CCSDS-133.0-B-1] Space Packet Protocol

[CCCSDS-732.0-B-2] AOS Space Data Link Protocol

[ECSS-E-ST-50-01C] Space engineering - Space data links - Telemetry synchronization and channel coding

[ECSS-E-ST-50-03C] Space engineering - Space data links - Telemetry transfer frame protocol

[ECSS-E-ST-50-05C] Space engineering - Radio frequency and modulation

[PSS-04-103] Telemetry channel coding standard

[PSS-04-105] Radio frequency and modulation standard

[PSS-04-106] Packet telemetry standard

142.2.2 Acronyms and abbreviations

AOS Advanced Orbiting Systems

ASM Attached Synchronization Marker

CCSDS Consultative Committee for Space Data Systems

CLCW Command Link Control Word

CRC Cyclic Redundancy Code

DMA Direct Memory Access

ECSS European Cooperation for Space Standardization

ESA European Space Agency

FECF Frame Error Control Field

FHEC Frame Header Error Control

FHP First Header Pointer

GF Galois Field

LFSR Linear Feedback Shift Register

MC Master Channel

NRZ Non Return to Zero

OCF Operational Control Field

PSR Pseudo Randomiser

PSS Procedures, Standards and Specifications

RS Reed-Solomon

SP Split-Phase

TE Turbo Encoder

TM Telemetry

VC Virtual Channel

142.3 Layers

142.3.1 Introduction

The Packet Telemetry (or simply Telemetry or TM) and Advanced Orbiting System (AOS) standards are similar in their format, with only some minor variations. The AOS part covered here is the down-link or transmitter, not the uplink or receiver.

The relationship between these standards and the Open Systems Interconnection (OSI) reference model is such that the OSI Data Link Layer corresponds to two separate layer, namely the Data Link Protocol Sub-layer and Synchronization and Channel Coding Sub-Layer. The OSI Data Link Layer is covered here.

The OSI Physical Layer is also covered here to some extended, as specified in [ECSS-E-ST-50-05C] and [PSS-04-105].

The OSI Network Layer or higher layers are not covered here.

142.3.2 Data Link Protocol Sub-layer

The Data Link Protocol Sub-layer differs somewhat between TM and AOS. Differences are pointed out where needed in the subsequent descriptions.

The following functionality is not implemented in the core:

- Packet Processing
- Bitstream Processing (applies to AOS only)

The following functionality is implemented in the core:

- Virtual Channel Generation (for Idle Frame generation only)
- Virtual Channel Multiplexing (for Idle Frame generation only)
- Master Channel Generation (applies to Packet Telemetry only)
- Master Channel Multiplexing (including Idle Frame generation)
- All Frame Generation

142.3.3 Synchronization and Channel Coding Sub-Layer

The Synchronization and Channel Coding Sub-Layer does not differ between TM and AOS.

The following functionality is implemented in the core:

- Attached Synchronization Marker
- Reed-Solomon coding
- Turbo coding (future option)
- Pseudo-Randomiser
- Convolutional coding

142.3.4 Physical Layer

The Physical Layer does not differ between TM and AOS.

The following functionality is implemented in the core:

- Non-Return-to-Zero modulation
- Split-Phase modulation
- Sub-Carrier modulation

142.4 Data Link Protocol Sub-Layer

142.4.1 Physical Channel

The configuration of a Physical Channel covers the following parameters:

- Transfer Frame Length (in number of octets)
- Transfer Frame Version Number

Note that there are other parameters that need to be configured for a Physical Channel, as listed in section 142.4.8, covering the All Frame Generation functionality.

The Transfer Frame Length can be programmed by means of the DMA length register.

The Transfer Frame Version Number can be programmed by means of a register, and can take one of two legal values: 00b for Telemetry and 01b for AOS.

142.4.2 Virtual Channel Frame Service

The Virtual Channel Frame Service is implemented by means of a DMA interface, providing the user with a means for inserting Transfer Frames into the Telemetry Encoder. Transfer Frames are automatically fetched from memory, for which the user configures a descriptor table with descriptors that point to each individual Transfer Frame. For each individual Transfer Frame the descriptor also provides means for bypassing functions in the Telemetry Encoder. This includes the following:

- Virtual Channel Counter generation can be enabled in the Virtual Channel Generation function (this function is normally only used for Idle Frame generation but can be used for the Virtual Channel Frame Service when sharing a Virtual Channel)
- Master Channel Counter generation can be bypassed in the Master Channel Generation function (TM only)
- Frame Secondary Header (FSH) generation can be bypassed in the Master Channel Generation function (TM only)
- Operational Control Field (OCF) generation can be bypassed in the Master Channel Generation function (TM only)
- Frame Error Header Control (FECH) generation can be bypassed in the All Frame Generation function (AOS only)
- Insert Zone (IZ) generation can be bypassed in the All Frame Generation function (AOS only)
- Frame Error Control Field (FECF) generation can be bypassed in the All Frame Generation function
- A Time Strobe can be generated for the Transfer Frame.

Note that the above features can only be bypassed for each Transfer Frame, the overall enabling of the features is done for the corresponding functions in the Telemetry Encoder, as described in the subsequent sections.

The detailed operation of the DMA interface is described in section 142.8.

142.4.3 Virtual Channel Generation

The Virtual Channel Generation function is used to generate the Virtual Channel Counter for Idle Frames as described hereafter. The function can however also be enabled for any Transfer Frame inserted via the Virtual Channel Frame Service described above, allowing a Virtual Channel to be shared between the two services. In this case the Virtual Channel Counter, the Extended Virtual Channel Counter (only for TM, as defined for ECSS and PSS, including the complete Transfer Frame Secondary Header) and the Virtual Channel Counter Cycle (only for AOS) fields will be inserted and incremented automatically when enabled as described hereafter.

142.4.4 Virtual Channel Multiplexing

The Virtual Channel Multiplexing Function is used to multiplex Transfer Frames of different Virtual Channels of a Master Channel. Virtual Channel Multiplexing in the core is performed between two sources: Transfer Frames provided through the Virtual Channel Frame Service and Idle Frames. Note that multiplexing between different Virtual Channels is assumed to be done as part of the Virtual Channel Frame Service outside the core.

The Virtual Channel Frame Service user interface is described above. The Idle Frame generation is described hereafter.

Idle Frame generation can be enabled and disabled by means of a register. The Spacecraft ID to be used for Idle Frames is programmable by means of a register. The Virtual Channel ID to be used for Idle Frames is programmable by means of a register.

Master Channel Counter generation for Idle Frames can be enabled and disabled by means of a register (only for TM). Note that it is also possible to generate the Master Channel Counter field as part of the Master Channel Generation function described in the next section. When Master Channel Counter generation is enabled for Idle Frames, then the generation in the Master Channel Generation function is bypassed.

The Virtual Channel Counter generation for Idle Frames is always enabled (both for TM and AOS) and generated in the Virtual Channel Generation function described above.

Extended Virtual Channel Counter generation for Idle Frames can be enabled and disabled by means of a register (only for TM, as defined for ECSS and PSS). This includes the complete Transfer Frame Secondary Header.

Virtual Channel Counter Cycle generation for Idle Frames can be enabled and disabled by means of a register (only for AOS).

If Frame Secondary Header generation is enabled in the Master Channel Generation function described in the next section, it can be bypassed for Idle Frames, programmable by means of a register. This allows VC_FSH or MC_FSH realization.

If Operation Control Field generation is enabled in the Master Channel Generation function described in the next section, it can be bypassed for Idle Frames, programmable by means of a register. This allows VC_OCF or MC_OCF realization.

142.4.5 Master Channel Generation

The Master Channel Counter can be generated for all frames on a master channel (only for TM). It can be can be enabled and disabled by means of a register. The generation can also be bypassed for Idle Frames or Transfer Frames provided via the DMA interface.

The Frame Secondary Header (FSH) can be generated from a 128-bit register (only for TM). This can be done for all frames on an master channel (MC_FSH) or be bypassed for Idle Frames or Transfer Frames provided via the DMA interface, effectively implementing FSH on a per virtual channel basis (VC_FSH). The FSH length is programmable by means of a register.

The Operational Control Field (OCF) can be generated from a 32-bit register. This can be done for all frames on an master channel (MC_OCF) or be bypassed for Idle Frames or Transfer Frames provided via the DMA interface, effectively implementing OCF on a per virtual channel basis (VC_OCF).

142.4.6 Master Channel Frame Service

The Master Channel Frame Service user interface is equivalent to the previously described Virtual Channel Frame Service user interface, using the same DMA interface. The interface can thus be used for inserting both Master Channel Transfer Frame and Virtual Channel Transfer Frames.

142.4.7 Master Channel Multiplexing

The Master Channel Multiplexing Function is used to multiplex Transfer Frames of different Master Channels of a Physical Channel. Master Channel Multiplexing is performed between three sources: Master Channel Generation Service, Master Channel Frame Service and Idle Frames.

Bypassing all the functionality of the Master Channel Generation functionality described above effectively establishes the master channel frame service. The same holds for the Idle Frame generation described above, allowing the core to generate Idle Frame on the level of the Physical Channel.

142.4.8 All Frame Generation

The All Frame Generation functionality operates on all transfer frames of a Physical Channel. Each of the individual functions can be bypassed for each frame coming from the DMA interface or idle frame generation functionality.

The Frame Header Error Control (FHEC) generation can be enabled and disabled by means of a register (AOS only).

The Insert Zone can be generated from a 128-bit register (only for AOS). This can be done for all frames on an physical channel (MC_FSH) or be bypassed for Idle Frames or Transfer Frames provided via the DMA interface. The Insert Zone length is programmable by means of a register. Note that the Insert Zone and Frame Secondary Header functionality share the same resources, since they cannot be used simultaneously for a Physical Channel.

Frame Error Control Field (FECF) generation can be enabled and disabled by means of a register.

142.5 Synchronization and Channel Coding Sub-Layer

142.5.1 Attached Synchronization Marker

The 32-bit Attached Synchronization Marker is placed in front of each Transfer Frame as per [CCSDS-131.0-B-2] and [ECSS-E-ST-50-03C].

An alternative Attached Synchronization Marker for embedded data streams can also be used, its enabling and bit pattern being programmable via a configuration register.

142.5.2 Reed-Solomon Encoder

The CCSDS recommendation [CCSDS-131.0-B-2] and ECSS standard [ECSS-E-ST-50-03C] specify Reed-Solomon codes, one (255, 223) code and one (255, 239) code. The ESA PSS standard [PSS013] only specifies the former code. Although the definition style differs between the documents, the (255, 223) code is the same in all three documents. The definition used in this document is based on the PSS standard [PSS013].

The Reed-Solomon Encoder implements both codes.

The Reed-Solomon encoder is compliant with the coding algorithms in [CCSDS-131.0-B-2] and [ECSS-E-ST-50-03C]:

- there are 8 bits per symbol;
- there are 255 symbols per codeword;
- the encoding is systematic:
- for E=8 or (255, 239), the first 239 symbols transmitted are information symbols, and the last 16 symbols transmitted are check symbols;
- for E=16 or (255, 223), the first 223 symbols transmitted are information symbols, and the last 32 symbols transmitted are check symbols;
- the E=8 code can correct up to 8 symbol errors per codeword;

- the E=16 code can correct up to 16 symbol errors per codeword;
- the field polynomial is

$$f_{esa}(x) = x^8 + x^6 + x^4 + x^3 + x^2 + x + 1$$

- the code generator polynomial for E=8 is

$$g_{esa}(x) = \prod_{i=120}^{135} (x + \alpha^i) = \sum_{j=0}^{16} g_j \cdot x^j$$

  for which the highest power of x is transmitted first;
- the code generator polynomial for E=16 is

$$g_{esa}(x) = \prod_{i=112}^{143} (x + \alpha^i) = \sum_{j=0}^{32} g_j \cdot x^j$$

  for which the highest power of x is transmitted first;
- interleaving is supported for depth I = {1 to 8}, where information symbols are encoded as I codewords with symbol numbers $i + j*I$ belonging to codeword i {where $0 \leq i < I$ and $0 \leq j < 255$};
- shortened codeword lengths are supported;
- the input and output data from the encoder are in the representation specified by the following transformation matrix T_{esa}, where i_0 is transferred first

$$\begin{bmatrix} \iota_0 & \iota_1 & \iota_2 & \iota_3 & \iota_4 & \iota_5 & \iota_6 & \iota_7 \end{bmatrix} = \begin{bmatrix} \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

- the following matrix T^{-1}_{esa} specifying the reverse transformation

$$\begin{bmatrix} \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 \end{bmatrix} = \begin{bmatrix} \iota_0 & \iota_1 & \iota_2 & \iota_3 & \iota_4 & \iota_5 & \iota_6 & \iota_7 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

- the Reed-Solomon output is non-return-to-zero level encoded.

The Reed-Solomon Encoder encodes a bit stream from preceding encoders and the resulting symbol stream is output to subsequent encoder and modulators. The encoder generates codeblocks by receiving information symbols from the preceding encoders which are transmitted unmodified while calculating the corresponding check symbols which in turn are transmitted after the information symbols. The check symbol calculation is disabled during reception and transmission of unmodified data not

related to the encoding. The calculation is independent of any previous codeblock and is perform correctly on the reception of the first information symbol after a reset.

Each information symbol corresponds to an 8 bit symbol. The symbol is fed to a binary network in which parallel multiplication with the coefficients of a generator polynomial is performed. The products are added to the values contained in the check symbol memory and the sum is then fed back to the check symbol memory while shifted one step. This addition is performed octet wise per symbol. This cycle is repeated until all information symbols have been received. The contents of the check symbol memory are then output from the encoder. The encoder is based on a parallel architecture, including parallel multiplier and adder.

The encoder can be configured at compile time to support only the E=16 (255, 223) code, only the E=8 (255, 239) code, or both. This is done with the *reed* VHDL generic. Only the selected coding schemes are implemented. The choice between the E=16 and E=8 coding can be performed during operation by means of a configuration register.

The maximum number of supported interleave depths I_{max} is selected at compile time with the *reed-depth* VHDL generic, the range being 1 to 8. For a specific instantiation of the encoder, the choice of any interleave depth ranging from 1 to the chosen I_{max} is supported during operation. The area of the encoder is minimized, i.e. logic required for a greater interleave depth than I_{max} is not unnecessarily included.

The interleave depth is chosen during operation by means of a configuration register.

142.5.3 Pseudo-Randomiser

The Pseudo-Randomiser (PSR) generates a bit sequence according to [CCSDS-131.0-B-2] and [ECSS-E-ST-50-03C] which is xor-ed with the data output of preceding encoders. This function allows the required bit transition density to be obtained on a channel in order to permit the receiver on ground to maintain bit synchronization.

The polynomial for the Pseudo-Randomiser is $h(x) = x^8+x^7+x^5+x^3+1$ and is implemented as a Fibonacci version (many-to-one implementation) of a Linear Feedback Shift Register (LFSR). The registers of the LFSR are initialized to all ones between Transfer Frames. The Attached Synchronization Marker (ASM) is not effected by the encoding.

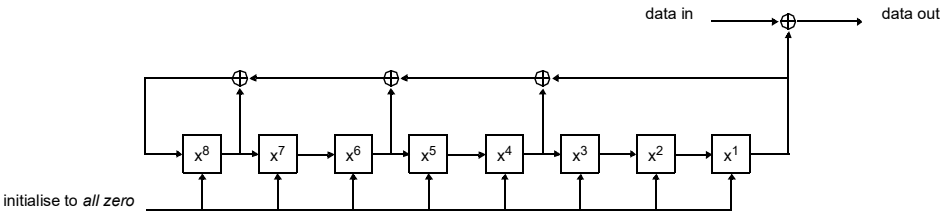


Figure 348. Pseudo-randomiser

142.5.4 Convolutional Encoder

The Convolutional Encoder (CE) implements two convolutional encoding schemes. The ESA PSS standard [PSS-04-103] specifies a basic convolutional code without puncturing. This basic convolutional code is also specified in the CCSDS recommendation [CCSDS-131.0-B-2] and ECSS standard [ECSS-E-ST-50-03C], which in addition specifies a punctured convolutional code.

The basic convolutional code has a code rate of 1/2, a constraint length of 7, and the connection vectors G1 = 1111001_b (171 octal) and G2 = 1011011_b (133 octal) with symbol inversion on output path, where G1 is associated with the first symbol output.

The punctured convolutional code has a code rate of 1/2 which is punctured to 2/3, 3/4, 5/6 or 7/8, a constraint length of 7, and the connection vectors G1 = 1111001_b (171 octal) and G2 = 1011011_b (133 octal) without any symbol inversion. The puncturing and output sequences are defined in [CCSDS-

131.0-B-2]. The encoder also supports rate 1/2 unpunctured coding with aforementioned connection vectors and no symbol inversion.

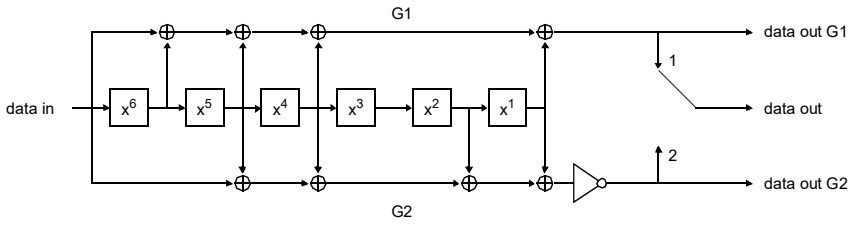


Figure 349. Unpuctured convolutional encoder

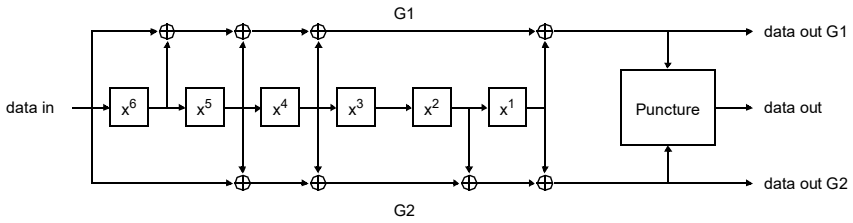


Figure 350. Punctured convolutional encoder

142.6 Physical Layer

142.6.1 Non-Return-to-Zero Mark encoder

The Non-Return-to-Zero Mark encoder (NRZ) encodes differentially a bit stream from preceding encoders according to [ECSS-E-ST-50-05C]. The waveform is shown in figure 351

Both data and the Attached Synchronization Marker (ASM) are affected by the coding. When the encoder is not enabled, the bit stream is by default non-return-to-zero level encoded.

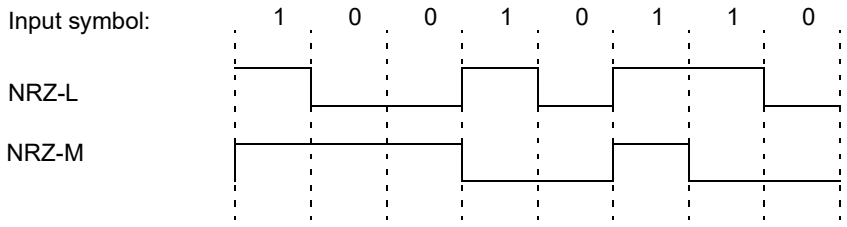


Figure 351. NRZ-L and NRZ-M waveform

142.6.2 Split-Phase Level modulator

The Split-Phase Level modulator (SP) modulates a bit stream (output symbols) from preceding encoders according (input symbols) to [ECSS-E-ST-50-05C]. The waveform is shown in figure 352.

Both data and the Attached Synchronization Marker (ASM) are effected by the modulator. The modulator will increase the output bit rate with a factor of two.

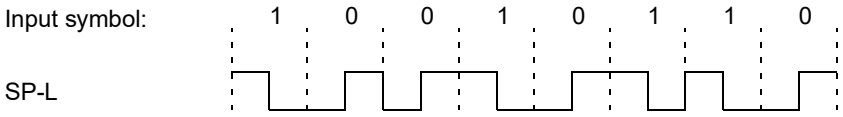


Figure 352. SP-L waveform

142.6.3 Sub-Carrier modulator

The Sub-Carrier modulator (SC) modulates a bit stream (output symbols) from preceding encoders according (input symbols) to [ECSS-E-ST-50-05C], which is Binary Phase Shift Key modulation (BPSK) or Phase Shift Key Square.

The sub-carrier modulation frequency is programmable. The symbol rate clock be divided to a degree 2^{15}. The divider can be configured during operation to divide the symbol rate clock frequency from 1/2 to $1/2^{15}$. The phase of the sub-carrier is programmable, selecting which phase 0^{o} or 180^{o} should correspond to a logical one on the input.

142.6.4 Clock Divider

The Clock Divider (CD) provides clock enable signals for the telemetry and channel encoding chain. The clock enable signals are used for controlling the bit rates of the different encoder and modulators.

The source for the bit rate frequency is the dedicated bit rate clock input. The bit rate clock input can be divided to a degree 2^{15}. The divider can be configured during operation to divide the bit rate clock frequency from 1/1 to $1/2^{15}$. In addition, the Sub-Carrier modulator can divide the above resulting clock frequency from 1/2 to $1/2^{15}$. The divider in the sub-carrier modulator can be used without enabling actual sub-carrier modulation, allowing division up to $1/2^{30}$.

The bit rate frequency is based on the output frequency of the last encoder in a coding chain, except for the sub-carrier modulator. No actual clock division is performed, since clock enable signals are used. No clock multiplexing is performed in the core.

The Clock Divider (CD) supports clock rate increases for the following encoders and rates:

- Convolutional Encoder (CE), 1/2, 2/3, 3/4, 5/6 and 7/8;
- Split-Phase Level modulator (SP-L), rate 1/2;
- Sub-Carrier modulator (SC), rate 1/2 to $1/2^{15}$.

The resulting symbol rate and telemetry rate are depended on what encoders and modulators are enabled. The following variables are used in the tables hereafter: f = input bit frequency, n = SYMBOLRATE+1 (GRTM physical layer register field +1), and m = SUBRATE+1 (physical layer register field +1), c = convolutional coding rate {1/2, 2/3, 3/4, 5/6, 7/8) (see CERATE field in GRTM coding sub-layer register).

*Table 2449.*Data rates without sub-carrier modulation (SUB=0)

Coding & Modulation	Telemetry rate	Convolutional rate	Split-Phase rate	Sub-carrier frequency	Output symbol rate	Output clock frequency
-	f / n / m	-	-	-	f / n / m	f / n / m
CE	(f / n / m) * c	f / n / m	-	-	f / n / m	f / n / m
SP-L	f / n / m / 2	-	f / n / m	-	f / n / m	f / n / m
CE + SP-L	(f / n / m / 2) * c	f / n / m / 2	f / n / m	-	f / n / m	f / n / m
For n = 1, no output symbol clock is generated, i.e. SYMBOLRATE register field equals 0. m should be an even number, i.e. SUBRATE register field should be uneven and > 0 to generate an output symbol clock with 50% duty cycle. If m > 1 then also n must be > 1, i.e. if SUBRATE register field is > 0 then SYMBOLRATE register field must be > 0.						

*Table 2450.*Data rates with sub-carrier modulation (SUB=1)

Coding & Modulation	Telemetry rate	Convolutional rate	Split-Phase rate	Sub-carrier frequency	Output symbol rate [1]	Output clock frequency
SC	f / n / m	-	-	f / n / 2	f / n	f / n
CE + SC	(f / n / m) * c	f / n / m	-	f / n / 2	f / n	f / n
SP-L+ SC	f / n / m / 2	-	f / n / m	f / n / 2	f / n	f / n
CE + SP-L + SC	(f / n / m / 2) * c	f / n / m / 2	f / n / m	f / n / 2	f / n	f / n

n = 1 or *m* = 1 are invalid settings for sub-carrier modulation, i.e SYMBOLRATE and SUBRATE register fields must be > 0.
m must be an even number, i.e. SUBRATE register field must be uneven and > 0.
m defines number of sub-carrier phases per input bit from preceding encoder or modulator.
Note 1: The output symbol rate for sub-carrier modulation corresponds to the rate of phases, not the frequency. Sub-carrier frequency is half the symbol rate.

142.7 Connectivity

The output from the Packet Telemetry and AOS encoder can be connected to:

- Reed-Solomon encoder
- Pseudo-Randomiser
- Non-Return-to-Zero Mark encoder
- Convolutional encoder
- Split-Phase Level modulator
- Sub-Carrier modulator

The input to the Reed-Solomon encoder can be connected to:

- Packet Telemetry and AOS encoder

The output from the Reed-Solomon encoder can be connected to:

- Pseudo-Randomiser
- Non-Return-to-Zero Mark modulator
- Convolutional encoder
- Split-Phase Level modulator
- Sub-Carrier modulator

The input to the Pseudo-Randomiser (PSR) can be connected to:

- Packet Telemetry and AOS encoder
- Reed-Solomon encoder

The output from the Pseudo-Randomiser (PSR) can be connected to:

- Non-Return-to-Zero Mark modulator
- Convolutional encoder
- Split-Phase Level modulator
- Sub-Carrier modulator

The input to the Non-Return-to-Zero Mark encoder (NRZ) can be connected to:

- Packet Telemetry and AOS encoder
- Reed-Solomon encoder
- Pseudo-Randomiser

The output from the Non-Return-to-Zero Mark encoder (NRZ) can be connected to:

- Convolutional encoder
- Split-Phase Level modulator
- Sub-Carrier modulator

The input to the Convolutional Encoder (CE) can be connected to:

- Packet Telemetry and AOS encoder
- Reed-Solomon encoder
- Pseudo-Randomiser
- Non-Return-to-Zero Mark encoder

The output from the Convolutional Encoder (CE) can be connected to:

- Split-Phase Level modulator
- Sub-Carrier modulator

The input to the Split-Phase Level modulator (SP) can be connected to:

- Packet Telemetry and AOS encoder
- Reed-Solomon encoder
- Pseudo-Randomiser
- Non-Return-to-Zero Mark encoder
- Convolutional encoder

The output from the Split-Phase Level modulator (SP) can be connected to:

- Sub-Carrier modulator

The input to the Sub-Carrier modulator (SC) can be connected to:

- Packet Telemetry and AOS encoder
- Reed-Solomon encoder
- Pseudo-Randomiser
- Non-Return-to-Zero Mark encode
- Convolutional encoder
- Split-Phase Level modulator

142.8 Operation

142.8.1 Introduction

The DMA interface provides a means for the user to insert Transfer Frames in the Packet Telemetry and AOS Encoder. Depending on which functions are enabled in the encoder, the various fields of the Transfer Frame are overwritten by the encoder. It is also possible to bypass some of these functions for each Transfer Frame by means of the control bits in the descriptor associated to each Transfer Frame. The DMA interface allows the implementation of Virtual Channel Frame Service and Master Channel Frame Service, or a mixture of both, depending on what functions are enabled or bypassed.

142.8.2 Descriptor setup

The transmitter DMA interface is used for transmitting transfer frames on the downlink. The transmission is done using descriptors located in memory.

A single descriptor is shown in table 2451 and 2452. The number of bytes to be sent is set globally for all transfer frames in the length field in register DMA length register. The address field of the descriptor should point to the start of the transfer frame. The address must be word-aligned. If the interrupt enable (IE) bit is set, an interrupt will be generated when the transfer frame has been sent (this requires that the transmitter interrupt enable bit in the control register is also set). The interrupt will be generated regardless of whether the transfer frame was transmitted successfully or not. The wrap (WR) bit is also a control bit that should be set before transmission and it will be explained later in this section.

Table 2451. GRTM transmit descriptor word 0 (address offset 0x0)

31 16	15	14	13 10	9	8	7	6	5	4	3	2	1	0
RESERVED	UE	TS	0000	VCE	MCB	FSHB	OCFB	FHECB	IZB	FECFB	IE	WR	EN

31: 16	RESERVED
15	Underrun Error (UE) - underrun occurred while transmitting frame (status bit only)
14	Time Strobe (TS) - generate a time strobe for this frame
13: 10	RESERVED
9	Virtual Channel Counter Enable (VCE) - enable virtual channel counter generation (using the Idle Frame virtual channel counter)
8	Master Channel Counter Bypass (MCB) - bypass master channel counter generation (TM only)
7	Frame Secondary Header Bypass (FSHB) - bypass frame secondary header generation (TM only)
6	Operational Control Field Bypass (OCFB) - bypass operational control field generation
5	Frame Error Header Control Bypass (FECHB) - bypass frame error header control generation (AOS)
4	Insert Zone Bypass (IZB) - bypass insert zone generation (AOS)
3	Frame Error Control Field Bypass (FECFB) - bypass frame error control field generation
2	Interrupt Enable (IE) - an interrupt will be generated when the frame from this descriptor has been sent provided that the transmitter interrupt enable bit in the control register is set. The interrupt is generated regardless if the frame was transmitted successfully or if it terminated with an error.
1	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 8. The pointer automatically wraps to zero when the 1 kB boundary of the descriptor table is reached.
0	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.

Table 2452. GRTM transmit descriptor word 1 (address offset 0x4)

31 2	1 0
ADDRESS	RES

31: 2	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.
1: 0	RESERVED

To enable a descriptor the enable (EN) bit should be set and after this is done, the descriptor should not be touched until the enable bit has been cleared by the core.

142.8.3 Starting transmissions

Enabling a descriptor is not enough to start a transmission. A pointer to the memory area holding the descriptors must first be set in the core. This is done in the transmitter descriptor pointer register. The address must be aligned to a 1 kByte boundary. Bits 31 to 10 hold the base address of descriptor area while bits 9 to 3 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the core, the pointer field is incremented by 8 to point at

the next descriptor. The pointer will automatically wrap back to zero when the next 1 kByte boundary has been reached (the descriptor at address offset 0x3F8 has been used). The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 1 kByte boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when a transmission is active.

The final step to activate the transmission is to set the transmit enable bit in the DMA control register. This tells the core that there are more active descriptors in the descriptor table. This bit should always be set when new descriptors are enabled, even if transmissions are already active. The descriptors must always be enabled before the transmit enable bit is set.

If external virtual channel generation is enabled the descriptors for this function are read from the area starting at the address defined by the externaladdr VHDL generic.

142.8.4 Descriptor handling after transmission

When a transmission of a frame has finished, status is written to the first word in the corresponding descriptor. The Underrun Error bit is set if the FIFO became empty before the frame was completely transmitted. The other bits in the first descriptor word are set to zero after transmission while the second word is left untouched. The enable bit should be used as the indicator when a descriptor can be used again, which is when it has been cleared by the core.

There are multiple bits in the DMA status register that hold transmission status.

The Transmitter Interrupt (TI) bit is set each time a DMA transmission of a transfer frame ended successfully. The Transmitter Error (TE) bit is set each time an DMA transmission of a transfer frame ended with an underrun error. For either event, an interrupt is generated for transfer frames for which the Interrupt Enable (IE) was set in the descriptor. The interrupt is maskable with the Interrupt Enable (IE) bit in the control register.

The Transmitter AMBA error (TA) bit is set when an AMBA AHB error was encountered either when reading a descriptor or when reading transfer frame data. Any active transmissions were aborted and the DMA channel was disabled. It is recommended that the Telemetry Encoder is reset after an AMBA AHB error. The interrupt is maskable with the Interrupt Enable (IE) bit in the control register.

The Transfer Frame Sent (TFS) bit is set whenever a transfer frame has been sent, independently if it was sent via the DMA interface or generated by the core. The interrupt is maskable with the Transfer Frame Interrupt Enable (TFIE) bit in the control register.

The Transfer Frame Failure (TFF) bit is set whenever a transfer frame has failed for other reasons, such as when Idle Frame generation is not enabled and no user Transfer Frame is ready for transmission, independently if it was sent via the DMA interface or generated by the core. The interrupt is maskable with the Transfer Frame Interrupt Enable (TFIE) bit in the control register.

The Transfer Frame Ongoing (TFO) bit is set when DMA transfers are enabled, and is not cleared until all DMA induced transfer frames have been transmitted after DMA transfers are disabled.

142.8.5 Interrupts

The Transfer Frame Sent (TFS) and Transfer Frame Failure (TFF) interrupts are maskable with the Transfer Frame Interrupt Enable (TFIE) bit in the DMA control register, and can be observed via the DMA status register.

The Transmitter Interrupt (TI), Transmitter Error (TE) and Transmitter AMBA Error (TA) interrupts are maskable with the Interrupt Enable (IE) bit in the DMA control register, and can be observed via the DMA status register.

The Time Strobe Interrupt (TSI) is maskable with the Transfer Frame Interrupt Enable (TFIE) bit in the DMA control register.

All interrupts except Time Strobe Interrupt (TSI) are output on interrupt number defined by *pirq* VHDL generic.

The Time Strobe Interrupt (TSI) is output on interrupt number defined by *pirq+1* VHDL generic.

142.9 Registers

The core is programmed through registers mapped into APB address space.

*Table 2453.*GRTM registers

APB address offset	Register
0x00	GRTM DMA Control register
0x04	GRTM DMA Status register
0x08	GRTM DMA Length register
0x0C	GRTM DMA Descriptor Pointer register
0x10	GRTM DMA Configuration register
0x14	GRTM DMA Revision register
0x20	GRTM DMA External VC Control & Status register
0x2C	GRTM DMA External VC Descriptor Pointer register
0x80	GRTM Control register
0x84	GRTM Status register (unused)
0x88	GRTM Configuration register
0x90	GRTM Physical Layer register
0x94	GRTM Coding Sub-Layer register
0x98	GRTM Attached Synchronization Marker
0xA0	GRTM All Frames Generation register
0xA4	GRTM Master Frame Generation register
0xA8	GRTM Idle Frame Generation register
0xC0	GRTM FSH/Insert Zone register 0
0xC4	GRTM FSH/Insert Zone register 1
0xC8	GRTM FSH/Insert Zone register 2
0xCC	GRTM FSH/Insert Zone register 3
0xD0	GRTM Operational Control Field register

142.9.1 GRTM DMA Control Register

Table 2454. 0x00 - DCR - DMA control register

31 5	4	3	2	1	0
RESERVED	TFIE	RST	TXRST	IE	EN
0	0	0	*	0	*
r	rw	rw	rw	rw	rw*

31: 5 RESERVED

4 Transfer Frame Interrupt Enable (TFIE) - enable telemetry frame sent (TFS) and failure (TFF) interrupt, and time strobe interrupt

3 Reset (RST) - reset DMA and telemetry transmitter

2 Reset Transmitter (TXRST) - reset telemetry transmitter

1 Interrupt Enable (IE) - enable DMA interrupt (TI), (TE) and (TA)

0 Enable (EN) - enable DMA transfers

142.9.2 GRTM DMA Status Register

Table 2455. 0x04 - DSR - DMA status register

31 8	7	6	5	4	3	2	1	0
RESERVED	TXSTAT	TXRDY	TFO	TFS	TFF	TA	TI	TE
0	0	0	*	0	0	0	*	*
r	r	r	r	wc	wc	wc	wc	wc

31: 8 RESERVED

7 Transmitter Reset Status (TXSTAT) - telemetry transmitter is in reset mode when set (read-only)

6 Transmitter Ready (TXRDY) - telemetry transmitter ready for operation after setting the TE bit in GRTM control register (read-only)

5 Transfer Frame Ongoing (TFO) - telemetry frames via DMA transfer are on-going (read-only)

4 Transfer Frame Sent (TFS) - telemetry frame interrupt, cleared by writing a logical 1

3 Transfer Frame Failure (TFF) - telemetry transmitter failure, cleared by writing a logical 1

2 Transmitter AMBA Error (TA) - DMA AMBA AHB error, cleared by writing a logical 1

1 Transmitter Interrupt (TI) - DMA interrupt, cleared by writing a logical 1

0 Transmitter Error (TE) - DMA transmitter underrun, cleared by writing a logical 1

142.9.3 GRTM DMA Length Register

Table 2456. 0x08 - DLR - DMA length register

31 27	26 16	15 11	10 0
RESERVED	LIMIT-1	RESERVED	LENGTH-1
0	*	0	*
r	rw	r	rw

31: 27 RESERVED

26: 16 Transfer Limit (LIMIT)- length-1 of data to be fetched by DMA before transfer starts.

Note: LIMIT must be equal to or less than LENGTH.
LIMIT must be equal to or less than FIFOSZ.
LIMIT must be equal to or larger than BLOCKSZ*2 for LENGTH > BLOCKSZ*2.

15: 11 RESERVED

10: 0 Transfer Length (LENGTH) - length-1 of data to be transferred by DMA

142.9.4 GRTM DMA Descriptor Pointer Register

Table 2457. 0x0C - DPR - DMA descriptor pointer register

31 10	9 3	2 0
BASE	INDEX	RESERVED
*	*	0
rw	rw	r

31: 10 Descriptor base (BASE) - base address of descriptor table

9: 3 Descriptor index (INDEX) - index of active descriptor in descriptor table

2: 0 Reserved - fixed to “00”

142.9.5 GRTM DMA Configuration Register (read-only)

Table 2458. 0x10 - DCF - DMA configuration register (read-only)

31 16	15 0
FIFOSZ	BLOCKSZ
*	*
r	r

31: 16 FIFO size (FIFOSZ) - size of FIFO memory in number of bytes (read-only)

15: 0 Block size (BLOCKSZ) - size of block in number of bytes (read-only)

142.9.6 GRTM DMA Revision Register (read-only)

Table 2459. 0x14 - DRR - DMA revision register (read-only)

31 22	21	20	19	18	17	16	15 8	7 0
RESERVED	AS	ASX	FIX	EX	IN	TIRQ	REVISION	SUB REVISION
0	-	-	*	*	*	*	*	0x04
r	r	r	r	r	r	r	r	r

31: 22 RESERVED

21 Autostart (AS) - Automatic start for telemetry

20 Autostart external (ASX) - Automatic start for external Virtual Channels

19 Fixed Frame Length (FIX) - Frame length fixed

18 External Virtual Channels (EX) - External Virtual Channels supported

17 Internal Virtual Channels (IN) - Internal Virtual Channels supported

16 Time Strobe Interrupt (TIRQ) - Separate time strobe interrupt supported

15: 8 REVISION - Main revision number

0x00: Initial release

7: 0 SUB REVISION - Sub revision number

0x00: Initial release

0x01: Added time interrupt, moved TXRDY bit, added TXSTAT bit, added this revision register

0x02: Added support for internal and external virtual channels

0x03: Added Master Channel Frame Counter value to master and idle frame generation registers

0x04: Added indicators for autostart of Telemetry and autostart of external Virtual Channels

142.9.7 GRTM DMA External VC Control & Status Register

*Table 2460.*0x20 - EUCS - DMA external VC control & status register

31 6	5	4 3	2	1	0
RESERVED	XTFO	RESERVED	XTI	XTE	XEN
0	*	0	*	*	*
r	r	r	wc	wc	rw

31: 6 RESERVED

5 External Transfer Frame Ongoing (XTFO) - telemetry frames via DMA transfer for external VC are on-going (read-only)

4: 3 RESERVED

2 External Transmitter Interrupt (XTI) - DMA interrupt for external VC, cleared by writing a logical 1

1 External Transmitter Error (XTE) - DMA transmitter underrun for external VC, cleared by writing a logical 1

0 External Enable (XEN) - enable DMA transfers for external VC (note that descriptor table is checked continuously till this bit is cleared).

142.9.8 GRTM DMA External VC Descriptor Pointer Register

Table 2461. 0x2C - EVDP - DMA external VC descriptor pointer register

31 10	9 3	2 0
BASE	INDEX	RESERVED
*	*	0
rw	rw	r

31: 10 Descriptor base (BASE) - base address of descriptor table

9: 3 Descriptor index (INDEX) - index of active descriptor in descriptor table

2: 0 Reserved - fixed to “00”

142.9.9 GRTM Control Register

Table 2462. 0x80 - CTRL - control register

31 1	0
RESERVED	TE
0	*
r	rw

31: 1 RESERVED

0: Transmitter Enable (TE) - enables telemetry transmitter (should be done after the complete configuration of the telemetry transmitter, including the LENGTH field in the GRTM DMA length register)

142.9.10GRTM Configuration Register (read-only)

Table 2463. 0x88 - CONF - configuration register (read-only)

31 23	22	21	20	19	18	17	16	15	14	13	12	11	10 9	8 6	5	4	3	2	1	0
RESERVED	OCFB	CIF	AOS	FHEC	IZ	MCG	FSH	IDLE	EVC	OCF	FECF	AASM	RS	RS DEPTH	TE	PSR	NRZ	CE	SP	SC
0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

31: 23	RESERVED
22	Operational Control Field Bypass (OCFB) - CLCW implemented externally, no OCF_SDU register
21	Encryption/Cipher Interface (CIF) - interface between protocol and channel coding sub-layers
20	Advanced Orbiting Systems (AOS) - AOS transfer frame generation implemented
19	Frame Header Error Control (FHEC) - frame header error control implemented, only if AOS also set
18	Insert Zone (IZ) - insert zone implemented, only if AOS also set
17	Master Channel Generation (MCG) - master channel counter generation implemented
16	Frame Secondary Header (FSH) - frame secondary header implemented
15	Idle Frame Generation (IDLE) - idle frame generation implemented
14	Extended VC Cntr (EVC) - extended virtual channel counter implemented (ECSS)
13	Operational Control Field (OCF) - CLCW implemented
12	Frame Error Control Field (FECF) - transfer frame CRC implemented
11	Alternative ASM (AASM) - alternative attached synchronization marker implemented
10: 9	Reed-Solomon (RS) - reed-solomon encoder implemented, “01” E=16, “10” E=8, “11” E=16 & 8
8: 6	Reed-Solomon Depth (RSDEPTH) - reed-solomon interleave depth -1 implemented
5	Turbo Encoder (TE) - turbo encoder implemented (reserved)
4	Pseudo-Randomiser (PSR) - pseudo-Randomiser implemented
3	Non-Return-to-Zero (NRZ) - non-return-to-zero - mark encoding implemented
2	Convolutional Encoding (CE) - convolutional encoding implemented
1	Split-Phase Level (SP) - split-phase level modulation implemented
0	Sub Carrier (SC) - sub carrier modulation implemented

142.9.11GRTM Physical Layer Register

Table 2464. 0x90 - PLR - physical layer register

31	30 16	15	14 0
SF	SYMBOLRATE	SCF	SUBRATE
‘*	*	*	*
rw	rw	rw	rw

31	Symbol Fall (SF) - symbol clock has a falling edge at start of symbol bit
30: 16	Symbol Rate (SYMBOLRATE) - symbol rate division factor - 1
15	Sub Carrier Fall (SCF) -sub carrier output start with a falling edge for logical 1
14: 0	Sub Carrier Rate (SUBRATE) - sub carrier division factor - 1

142.9.12GRTM Coding Sub-Layer Register

Table 2465. 0x94 - CSL - coding sub-layer register

31 20	19	18 17	16	15	14 12	11	10 8	7	6	5	4 2	1	0
RESERVED	CIF	CSEL	AASM	RS	RSDEPTH	RS8	RESERVED	PSR	NRZ	CE	CE RATE	SP	SC
0	*	*	*	*	*	*	0	*	*	*	*	*	*
r	rw	rw	rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw

31: 20	RESERVED
19	Encryption/Cipher Interface (CIF) - enable external encryption/cipher interface between sub-layers
18: 17	Clock Selection (CSEL) - selection of external telemetry clock source (application specific)
16	Alternative ASM (AASM) - alternative attached synchronization marker enable. When enabled the value from the GRTM Attached Synchronization Marker register is used, else the standardized ASM value 0x1ACFFC1D is used
15	Reed-Solomon (RS) - reed-solomon encoder enable
14: 12	Reed-Solomon Depth (RSDEPTH) - reed-solomon interleave depth -1
11	Reed-Solomon Rate (RS8) - ‘0’ E=16, ‘1’ E=8
10: 8	RESERVED
7	Pseudo-Randomiser (PSR) - pseudo-Randomiser enable
6	Non-Return-to-Zero (NRZ) - non-return-to-zero - mark encoding enable
5	Convolutional Encoding (CE) - convolutional encoding enable
4: 2	Convolutional Encoding Rate (CERATE): “00-” rate 1/2, no puncturing “01-” rate 1/2, punctured “100” rate 2/3, punctured “101” rate 3/4, punctured “110” rate 5/6, punctured “111” rate 7/8, punctured
1	Split-Phase Level (SP) - split-phase level modulation enable
0	Sub Carrier (SC) - sub carrier modulation enable

142.9.13GRTM Attached Synchronization Marker Register

Table 2466. 0x98 - ASM - attached synchronization marker register

31 0
ASM
*
rw

31: 0	Attached Synchronization Marker (ASM) - pattern for alternative ASM, (bit 31 MSB sent first, bit 0 LSB sent last) (The reset value is the standardized alternative ASM value 0x352EF853.)

142.9.14GRTM All Frames Generation Register

Table 2467. 0xA0 - AFGR - all frames generation register

31 22	21 17	16	15	14	13 12	11 0
RESERVED	FSH / IZ LENGTH	IZ	FECF	FHEC	VER	RESERVED
0	*	*	*	*	*	0
r	rw	rw	rw	rw	rw	r

31: 22	RESERVED
21: 17	Frame Secondary Header (TM) / Insert Zone (AOS) (FSH / IZ LENGTH) - length in bytes
16	Insert Zone (IZ) - insert zone enabled, only with AOS
15	Frame Error Control Field (FECF) - transfer frame CRC enabled
14	Frame Header Error Control (FHEC) - frame header error control enabled, only with AOS
13: 12	Version (VER) - Transfer Frame Version - “00” Packet Telemetry, “01” AOS
11: 0	RESERVED

142.9.15GRTM Master Frame Generation Register

Table 2468. 0xA4 - MFGR - master frame generation register

31 24	23 4	3	2	1	0
MCFC	RESERVED	MC	FSH	OCF	OW
*	0	*	*	*	*
r	r	rw	rw	rw	rw

31: 24	Master Channel Frame Counter (MCFC) - diagnostic read out (read only, TM only)
23: 4	RESERVED
3	Master Channel (MC) - enable master channel counter generation (TM only)
2	Frame Secondary Header (FSH) - enable MC_FSH for master channel (TM only)
1	Operation Control Field (OCF) - enable MC_OCF for master channel
0	Over Write OCF (OW) - overwrite OCF bits 16 and 17 when set

142.9.16GRTM Idle Frame Generation Register

Table 2469. 0xA8 - IFGR - idle frame generation register

31 24	23 22	21	20	19	18	17	16	15 10	9 0
IDLEMCFC	RESERVED	IDLE	OCF	EVC	FSH	VCC	MC	VCID	SCID
*	0	*	*	*	*	*	*	*	*
r	r	rw	rw	rw	rw	rw	rw	rw	rw

31: 24 Idle Master Channel Frame Counter (IDLEMCFC) - diagnostic read out (read only, TM only)

23: 22 RESERVED

21 Idle Frames (IDLE) - enable idle frame generation

20 Operation Control Field (OCF) - enable OCF for idle frames

19 Extended Virtual Channel Counter (EVC) - enable extended virtual channel counter generation for idle frames (TM only, ECSS)

18 Frame Secondary Header (FSH) - enable FSH for idle frames (TM only)

17 Virtual Channel Counter Cycle (VCC) - enable virtual channel counter cycle generation for idle frames (AOS only)

16 Master Channel (MC) - enable separate master channel counter generation for idle frames (TM only)

15: 10 Virtual Channel Identifier (VCID) - virtual channel identifier for idle frames

9: 0 Spacecraft Identifier (SCID) - spacecraft identifier for idle frames

142.9.17GRTM FSH / IZ Register 0, MSB

Table 2470. 0xC0 - FSH0 - FSH / IZ register 0, MSB

31 0
DATA
*
rw

31: 0 FSH / Insert Zone Data (DATA) - data (bit 31 MSB sent first)

Note: Writing to this register prevents the new FSH/Insert Zone data value to be transferred.

142.9.18GRTM FSH / IZ Register 1

Table 2471. 0xC4 - FSH1 - FSH / IZ register 1

31 0
DATA
*
rw

31: 0 FSH / Insert Zone Data (DATA) - data

142.9.19GRTM FSH / IZ Register 2

Table 2472. 0xC8 - FSH2 - FSH / IZ register 2

31 0
DATA
*
rw

31: 0 FSH / Insert Zone Data (DATA) - data

142.9.20GRTM FSH / IZ Register 3, LSB

Table 2473. 0xCC - FSH3 - FSH / IZ register 3, LSB

31	0
DATA	
*	
rw	

31: 0 FSH / Insert Zone Data (DATA) - data (bit 0 LSB sent last)

Note: Writing to this registers enables the new FSH/Insert Zone data value to be transferred.

142.9.21GRTM OCF Register

Table 2474. 0x00 - OCF - OCF register

31	0
CLCW	
*	
rw	

31: 0 Operational Control Field (OCF) - CLCW data (bit 31 MSB, bit 0 LSB)

142.10 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x030. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

142.11 Implementation

142.11.1Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

142.12 Configuration options

Table 2475 shows the configuration options of the core (VHDL generics).

*Table 2475.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR	0 - 16#FFF#	0
pmask	MASK field of the APB BAR	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by core	0 - NAHBIRQ-1	0
memtech	Memory technology	0 to NTECH	0
ft	Enable fault-tolerance against SEU errors	0 - 1	0
blocksize	Block size (in number of bytes)	16 to 512	512
fifosize	FIFO size (in number of bytes)	32 to 4096	4096
fixedsize	Fixed Transfer Frame Length	0 - 1	0
framelength	Default Transfer Frame Length	0 - 4095	0
txlimit	Default Limit size	0 - 4095	0
nsync	Level of synchronization	1 - 2	2

*Table 2475.*Configuration options

Generic name	Function	Allowed range	Default
exconf	External reset configuration	0 - 1	0
autostart	Automatic start of Telemetry	0 - 1	0
internal	Support for internal Virtual Channels	0 - 1	1
external	Support for external Virtual Channels 1 = support external virtual channels 2 = support external virtual channels, autostart external virtual channels 3 = support external virtual channels, stand alone operation (autostart external virtual channels, never stop)	0 - 3	0
externaladdr	Default base address for external Virtual Channels	-	16#000#
resync	Resynchronization of internal constants	0 - 1	0
altasm	Alternative Attached Synchronization Marker	0 - 1	1
aos	Advanced Orbiting System (AOS)	0 - 1	1
fhec	Frame Header Error Control, AOS only	0 - 1	1
insertzone	Insert Zone, AOS only	0 - 1	1
mcgf	Master Channel Generation Function	0 - 1	1
fsh	Frame Secondary Header	0 - 1	1
idle	Idle Frame Generation	0 - 1	1
idleextvccntr	Extended Virtual Channel Counter, Idle Frames	0 - 1	1
ocf	Operation Control Field (2 equals external bypass)	0 - 2	1
fecf	Frame Error Control Field	0 - 1	1
cipher	Encryption / Cipher Interface	0-1	0
reed	Reed-Solomon, 1- E=16, 2 - E=8, 3 - E=8 & 16	0 - 3	3
reeddepth	Reed-Solomon Interleave Depth, maximum	1 - 8	8
turbo	Reserved	0	0
pseudo	Pseudo-Randomiser encoding	0 - 1	1
mark	Non-Return-to-Zero Mark modulation	0 - 1	1
conv	Convolutional coding	0 - 1	1
split	Split-Phase Level modulation	0 - 1	1
sub	Sub-Carrier modulation	0 - 1	1
timeirq	Separate time strobe interrupt	0 - 1	0
synchronous	0 = octet clock derived from transponder clock, separate clock domain 1 = octet clock equals the transponder clock, same clock domain	0 - 1	0
syncassert	0 = combinatorial (asynchronous) assertion of internal reset signal 1= sequential (synchronous) assertion of internal reset signal	0 - 1	0

142.13 Signal descriptions

Table 2476 shows the interface signals of the core (VHDL ports).

Table 2476. Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
TMI	BITLOCK	Input	Bit Lock	High
	RFAVAIL		RF Available	High
	OCF_SDU[0:31]		OCF_SDU Bypass	-
	CIPHER		Encryption/ Cipher Interface	-
	EXREQUEST		External VC Request	High
	CONF		External configuration	-
TMO	TIME	Output	Time strobe	High
	SYNC		ASM indicator	High
	FRAME		Frame indicator	High
	SERIAL		Serial bit data	High
	CLOCK		Serial bit data clock	High
	DATA [0:7]		Parallel data, octet	High
	STROBE		Parallel data strobe	High
	CLKSEL[0:1]		External clock selection	-
	FRAMELSB[0:1]		MC Counter least significant bits	-
	CIPHER		Encryption/ Cipher Interface	-
	SCID[0:9]		SCID setting	-
	OCF		OCF setting	High
	FECF		FECF setting	High
	EXENABLE		External VC enable	High
	EXGRANT		External VC grant	High
	EXREADY		External VC ready	High
TCLK	N/A	Input	Transponder clock	-
OCLKO	N/A	Output	Octet clock output	-
OCLKI	N/A	Input	Octet clock input	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-

* see GRLIB IP Library User's Manual

142.14 Signal definitions and reset values

The signals and their reset values are described in table 2477.

*Table 2477.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
bitlock	Input	Bit Lock	High	-
rfavail	Input	RF Available	High	-
time	Output	Time strobe	High	Logical 0
sync	Output	ASM indicator	High	Logical 0
frame	Output	Frame indicator	High	Logical 0
serial	Output	Serial bit data	-	-
clock	Output	Serial bit data clock	High	Logical 0
data[0:7]	Output	Parallel data, octet	-	-
strobe	Output	Parallel data strobe	High	Logical 0

142.15 Timing

The timing waveforms and timing parameters are shown in figure 353 and are defined in table 2478.

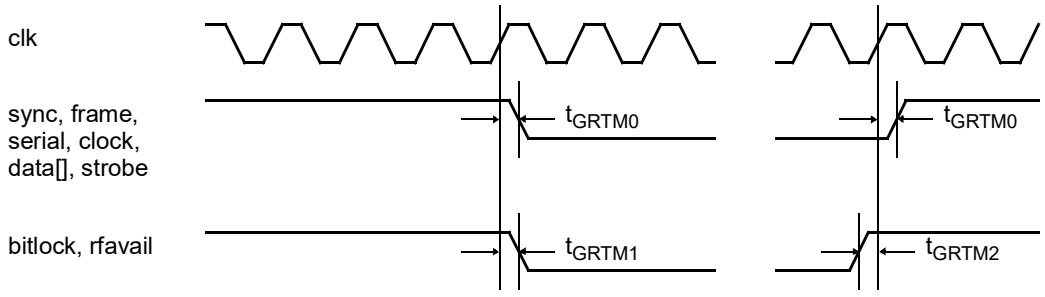


Figure 353. Timing waveforms

*Table 2478.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{GRTM0}	clock to output delay	rising *clk* edge	TBD	TBD	ns
t_{GRTM1}	input to clock hold	rising *clk* edge	-	-	ns
t_{GRTM2}	input to clock setup	rising *clk* edge	-	-	ns

Note: The inputs are re-synchronized inside the core. The signals do not have to meet any setup or hold requirements.

142.16 Library dependencies

Table 2479 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2479.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_TYPES	Signals, component	Component declaration

143 GRTM_DESC - CCSDS/ECSS Telemetry Encoder - Descriptor

143.1 Overview

The CCSDS/ECSS/PSS Telemetry Encoder Descriptor implements an automatic descriptor handler for external Telemetry Virtual Channels implemented in hardware (Telemetry Encoder Virtual Channel Generation function), not requiring software support.

143.2 Operation

143.2.1 Introduction

Warning: software should not read or write the descriptor table. All interaction is performed automatically by hardware.

The bandwidth is allocated equally between the external Telemetry Virtual Channels. Note that the descriptor table will be continuously checked by the Telemetry Encoder, even when all descriptors have their Enable (EN) bit cleared. This will go on until the External Enable (XEN) bit in the Telemetry Encoder is cleared by software.

143.2.2 Descriptor definition

A single descriptor is shown in table 2480 and 2481.

Table 2480. Transmit descriptor word 0 (address offset 0x0)

31 16	15	14	13 10	9	8	7	6	5	4	3	2	1	0
RESERVED	UE	TS	0000	VCE	MCB	FSHB	OCFB	FHECB	IZB	FECFB	IE	WR	EN

31: 16	RESERVED
15	Underrun Error (UE) - underrun occurred while transmitting frame (status bit only)
14	Time Strobe (TS) - generate a time strobe for this frame (static 0)
13: 10	RESERVED
9	Virtual Channel Counter Enable (VCE) - enable virtual channel counter generation (using the Idle Frame virtual channel counter) (static 0)
8	Master Channel Counter Bypass (MCB) - bypass master channel counter generation (TM only) (static 0)
7	Frame Secondary Header Bypass (FSHB) - bypass frame secondary header generation (TM only) (static 0)
6	Operational Control Field Bypass (OCFB) - bypass operational control field generation (static 0)
5	Frame Error Header Control Bypass (FECHB) - bypass frame error header control generation (AOS) (static 0)
4	Insert Zone Bypass (IZB) - bypass insert zone generation (AOS) (static 0)
3	Frame Error Control Field Bypass (FECFB) - bypass frame error control field generation (static 0)
2	Interrupt Enable (IE) - an interrupt will be generated when the frame from this descriptor has been sent provided that the transmitter interrupt enable bit in the control register is set. (static 0)
1	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 8. The pointer automatically wraps to zero when the 1 kB boundary of the descriptor table is reached. (Set to 1 for last descriptor entry, otherwise static 0).
0	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields. (Automatically set an cleared by hardware)

Table 2481. Transmit descriptor word 1 (address offset 0x4)

31	2	1	0
ADDRESS			RES

31: 2 Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.

1: 0 RESERVED

143.3 Registers

The core does not implement any memory mapped registers.

143.4 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x084. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

143.5 Implementation

143.5.1 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

143.6 Configuration options

Table 2482 shows the configuration options of the core (VHDL generics).

*Table 2482.*Configuration options

Generic name	Function	Allowed range	Default
hindex	Selects which AHB select signal (HSEL) will be used to access the memory.	0 to NAHBMAX-1	0
haddr	ADDR field of the AHB BAR	0 to 16#FFF#	0
hmask	MASK field of the AHB BAR	0 to 16#FFF#	16#FFF#
channels	Number of Virtual Channels	power of 2	2
frames	Number of frames per Virtual Channel	2 -	2

143.7 Signal descriptions

Table 2483 shows the interface signals of the core (VHDL ports).

*Table 2483.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
VCRI	-	Input	Virtual Channel Request Interface	-
VCRO	-	Output		-
DI	-	Input	Descriptor Interface	-
DO	-	Output		-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-

* see GRLIB IP Library User's Manual

143.8 Signal definitions and reset values

None.

143.9 Timing

None.

143.10 Library dependencies

Table 2484 shows the libraries used when instantiating the core (VHDL libraries).

Table 2484. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_TYPES	Signals, component	Component declaration
TMTC	GRTM_PKG	Signals, component	Component declaration

144 GRTM_VC - CCSDS/ECSS Telemetry Encoder - Virtual Channel Generation

144.1 Overview

The CCSDS/ECSS/PSS Telemetry Encoder Virtual Channel Generation function implements:

- Transfer Frame Primary Header insertion
- Transfer Frame Data Field insertion (with support for different lengths due to OCF and FECF)
- First Header Pointer (FHP) handling and insertion

The function keeps track of the number of octets received and the packet boundaries in order to calculated the First Header Pointer (FHP). The data are stored in pre-allocated slots in the buffer memory comprising complete Transfer Frames. The module fully supports the FHP generation and does not require any alignment of the packets with the Transfer Frame Data Field boundary.

The data input format can be CCSDS Space Packet [CCSDS-133.0-B-1] or any user-defined data-block. Data is input via a separate Virtual Channel Generation function input interface.

The function communicates with the Telemetry Encoder Virtual Channel Frame Service by means of a buffer memory space. The buffer memory space allocated to the Virtual Channel is treated as a circular buffer. The buffer memory space is accessed by means of a AMBA AHB master interface.

144.2 Registers

The core does not implement any memory mapped registers.

144.3 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x085. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

144.4 Implementation

144.4.1 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

144.5 Configuration options

Table 2485 shows the configuration options of the core (VHDL generics).

*Table 2485.*Configuration options

Generic name	Function	Allowed range	Default
hindex	Selects which AHB select signal (HSEL) will be used to access the memory.	0 to NAHBMAX-1	0
index	Channel index		0
id	0-7: Virtual Channel Identifier > 7: Virtual Channel Identifier is picked from the DYN-VCID input.		0
roomsize	Packet size for ready signalling		0
frames	Number of frames		2
framepitch	Distance between consecutive frames		2048
frameoffset	Offset for all frames		4096
framelength	Frame length (transmitted)		1115
frameaddress	Start address of frames		16#000#

144.6 Signal descriptions

Table 2486 shows the interface signals of the core (VHDL ports).

*Table 2486.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
FECF	N/A	Input	Frame Error Control Field	High
OCF	N/A	Input	Operational Control Field	High
SCID	N/A	Input	Spacecraft Identifier	-
VCII	-	Input	Virtual Channel Input Interface	-
VCIO	-	Output		-
VCRI	-	Input	Virtual Channel Request Interface	-
VCRO	-	Output		-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-
DYNVCID	N/A	Input	Dynamic Virtual Channel Identifier to be used instead of the id generic in transfer frame generation if id > 7	-

* see GRLIB IP Library User's Manual

144.7 Signal definitions and reset values

None.

144.8 Timing

None.

144.9 Library dependencies

Table 2487 shows the libraries used when instantiating the core (VHDL libraries).

Table 2487. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_TYPES	Signals, component	Component declaration
TMTC	GRTM_PKG	Signals, component	Component declaration

145 GRTM_PAHB - CCSDS/ECSS Telemetry Encoder - Virtual Channel Generation Input - AMBA

145.1 Overview

The Telemetry Encoder Virtual Channel Generation function input interface implements an interfaces towards the automatic Virtual Channel Generation function of the Telemetry Encoder (also called external Virtual Channels). Space Packets or any other user-defined data block can be input.

Data is transferred to the Virtual Channel Generation function by writing to the AMBA AHB slave interface, located in the AHB I/O area. Writing is only possible when the packet valid delimiter is asserted, else the access results in an AMBA access error. It is possible to transfer one, two or four bytes at a time, following the AMBA big-endian convention regarding send order. The last written data can be read back via the AMBA AHB slave interface. Data are output as octets to the Virtual Channel Generation function.

In the case the data from a previous write access has not been fully transferred over the interface, a new write access will result in an AMBA retry response. The progress of the interface can be monitored via the AMBA APB slave interface. An interrupt is generated when the data from the last write access has been transferred. An interrupt is also generated when the ready for input packet indicator is asserted.

The core incorporates status and monitoring functions accessible via the AMBA APB slave interface. This includes:

- Busy and ready signaling from Virtual Channel Generation function
- Interrupts on ready for new word, or ready for new packet (size 518 octets (set with *roomsize* VHDL generic))

145.2 Interrupts

Two interrupts are implemented by the interface:

Index:	Name:	Description:
0	NOT BUSY	Ready for a new data (word, half-word or byte)
1	READY	Ready for new packet

The interrupts are configured by means of the *pirq* VHDL generic.

145.3 Registers

The core is programmed through registers mapped into APB address space.

Table 2488. GRTM_PAHB registers

APB address offset	Register
16#004#	Status Register
16#008#	Control Register

145.3.1 Status Register (R)

*Table 2489.*0x04 - STAT - Status Register

31 2	1	0
RESERVED	BUSY	READY
0	*	*
r	r	r

1: BUSY Not ready for new input, busy with octet
0: READY Ready for new packet of maximum size

All bits are cleared to 0 at reset.

145.3.2 Control Register (R/W)

*Table 2490.*0x08 - CTRL - Control Register

31 10	9	8	7 3	2	1	0
RESERVED	BUSYEN	READYEN	RESERVED	VALID	RST	EN
0	0	0	0	0	0	0
r	rw	rw	r	rw*	rw	rw

8: BUSYEN
9: READYENEnable ready for packet interrupt when 1
2: VALID Packet valid delimiter, packet valid when 1, in-between packets when 0 (read-only)
1: RST Reset complete core when 1
0: EN Enable interface when 1

All bits are cleared to 0 at reset. Note that RST is read back as 0.

145.4 AHB I/O area

Data to be transferred to the Virtual Channel Generation function is written to the AMBA AHB slave interface which implements a AHB I/O area. See [GRLIB] for details.

Note that the address is not decoded by the core. Address decoding is only done by the AMBA AHB controller, for which the I/O area location and size is configured by means of the *ioaddr* and *iomask* VHDL generics. It is possible to transfer one, two or four bytes at a time, following the AMBA big-endian convention regarding send order. The last written data can be read back via the AMBA AHB slave interface. Data are output as octets on the Virtual Channel Generation interface.

*Table 2491.*AHB I/O area - data word definition

31 24	23 16	15 8	7 0
DATA [31:24]	DATA [23:16]	DATA [15:8]	DATA [7:0]

*Table 2492.*AHB I/O area - send order

Transfer size	Address offset	DATA [31:24]	DATA [23:16]	DATA [15:8]	DATA [7:0]	Comment
Word	0	first	second	third	last	Four bytes sent
Halfword	0	first	last	-	-	Two bytes sent
	2	-	-	first	last	Two bytes sent
Byte	0	first	-	-	-	One byte sent
	1	-	first	-	-	One byte sent
	2	-	-	first	-	One byte sent
	3	-	-	-	first	One byte sent

145.5 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x088. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

145.6 Implementation

145.6.1 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual).

145.7 Configuration options

Table 2493 shows the configuration options of the core (VHDL generics).

*Table 2493.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index.	1 - NAHBSLV-1	0
ioaddr	Addr field of the AHB IO bar.	0 - 16#FFF#	0
iomask	Mask field of the AHB IO bar.	0 - 16#FFF#	16#F00#
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFC#
pirq	Interrupt line used by the GRTM_PAHB.	0 - NAHBIRQ-1	0
syncrst	Only synchronous reset	0, 1	1
roomsize	Size of packet		518

145.8 Signal descriptions

Table 2494 shows the interface signals of the core (VHDL ports).

Table 2494. Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB slave input signals	-
AHBO	*	Output	AHB slave output signals	-

* see GRLIB IP Library User's Manual

145.9 Library dependencies

Table 2495 shows the libraries used when instantiating the core (VHDL libraries).

Table 2495. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_Types	Signals, component	Component declarations, signals.
TMTC	GRTM_PKG	Signals, component	Component declaration

146 GRTM_PW - CCSDS/ECSS Telemetry Encoder - Virtual Channel Generation Input - PacketWire

The PacketWire (PW) interface to a telemetry encoder is a simple bit synchronous protocol. There is one PacketWire interface for each telemetry Virtual Channel.

The data can be any CCSDS supported packets. The interface comprises three input signals; bit data, bit clock and packet delimiter. There is an additional discrete signal provided for busy signalling.

Data should consist of multiples of eight bits otherwise the last bits will be lost. The input packet delimiter signal is used to delimit packets. It should be asserted while a packet is being input, and de-asserted in between. In addition, the input packet delimiter signal should define the octet boundaries in the input data stream, the first octet explicitly and the following octets each subsequent eight bit clock cycles.

The interface is based on the de facto standard PacketWire interface used by the *European Space Agency* (ESA). At the time of writing there were no relevant documents available from the *European Cooperation for Space Standardization* (ECSS).

146.1 Operation

The PacketWire interface accepts and generates the waveform format shown in figure 354.

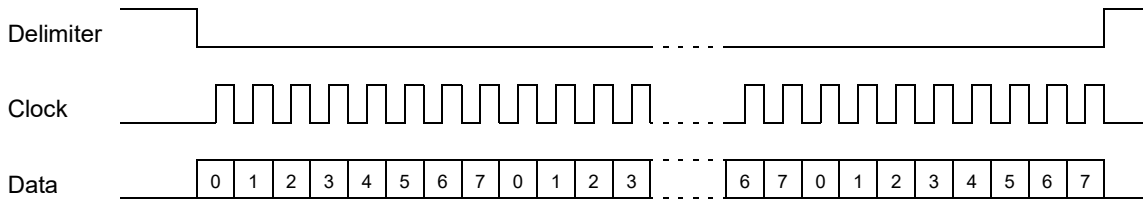


Figure 354. Synchronous bit serial waveform

The PacketWire protocol follows the CCSDS transmission convention, the most significant bit being sent first. Transmitted data should consist of multiples of eight bits otherwise the last bits will be lost. The input message delimiter port is used to delimit messages (packets). It should be asserted while a message is being input, and deasserted in between. In addition, the message delimiter port should define the octet boundaries in the data stream, the first octet explicitly and the following octets each subsequent eight bit clock cycles.

The maximum receiving input baud rate is defined as half the frequency of the system clock input. There is no lower limit for the input bit rate in the receiver.

The handshaking between the PacketWire links and the interface is implemented with a busy port. When a message is sent, the busy signal on the PacketWire input link will be asserted as soon as the input interface is not ready to receive more data, it will then be deasserted as soon as the interface is ready to receive the next octet. This gives the transmitter ample time to stop transmitting after the completion of an octet and wait for the busy signal deassertion before starting the transmission of the next octet. The handshaking is continued through out the message.

146.2 Signal definitions and reset values

The signals and their reset values are described in table 2496.

*Table 2496.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
*pw*valid_n*	Input	Delimiter: This input port is the message delimiter for the input interface. It should be deasserted between messages.	Low	-
*pw*clk*	Input	Bit clock: This input port is the PacketWire bit clock. The receiver registers are clocked on the rising edge.	Rising	-
*pw*data*	Input	Data: This input port is the serial data input for the interface. Data are sampled on the rising *pw*clk* edge when *pw*valid_n* is asserted.	-	-
*pw*busy*	Output	Not ready for octet: This port indicates whether the receiver is ready to receive one octet.	High	Logical 0

146.3 Timing

The timing waveforms and timing parameters are shown in figure 355 and are defined in table 2497.

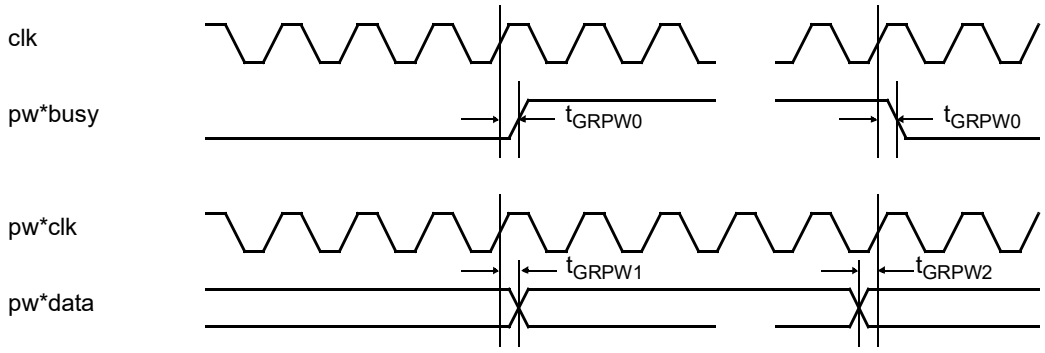


Figure 355. Timing waveforms

*Table 2497.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{GRPW0}	clock to output delay	rising *clk* edge	TBD	TBD	ns
t_{GRPW1}	input to clock hold	rising *pw*clk* edge	TBD	-	ns
t_{GRPW2}	input to clock setup	rising *pw*clk* edge	TBD	-	ns
t_{GRPW3}	*pw*valid_n* to *pw*clk* edge	rising *pw*clk* edge	TBD	-	ns
t_{GRPW3}	*pw*valid_n* de-asserted period	-	TBD		system clock periods

147 GRTM_UART - CCSDS/ECSS Telemetry Encoder - Virtual Channel Generation Input - UART

The PacketAsynchronous (PA) interface to a telemetry encoder is a simple bit asynchronous protocol. There can be one bit asynchronous interface per telemetry Virtual Channel.

The protocol has a fixed or programmable baud rate, 1 start bit, 8 data bits, optional odd parity, 1 or 2 stop bits, with a BREAK command for message delimiting (sending 13 bits of logical zero).

The data can be any CCSDS supported packet. The interface comprises one input signal with bit asynchronous data. There are two additional discrete signals provided for busy signalling.

147.1 Asynchronous bit serial data format

The asynchronous bit serial interface complies to the data format defined in [EIA 232]. It also complies to the data format and waveform shown in table 2498 and figure 356. The interface is independent of the transmitted data contents. Positive logic is considered for the data bits. The number of stop bits can optionally be either one or two. The parity bit can be optionally included.

Asynchronous bit serial format	start	D0	D1	D2	D3	D4	D5	D6	D7	parity	stop	stop
	first	*lsb*							*msb*			*last*
General data format $i = \{0, n\}$		8**i*+7	8**i*+6	8**i*+5	8**i*+4	8**i*+3	8**i*+2	8**i*+1	8**i*			
		last							*first*			

Table 2498. Asynchronous bit serial data format

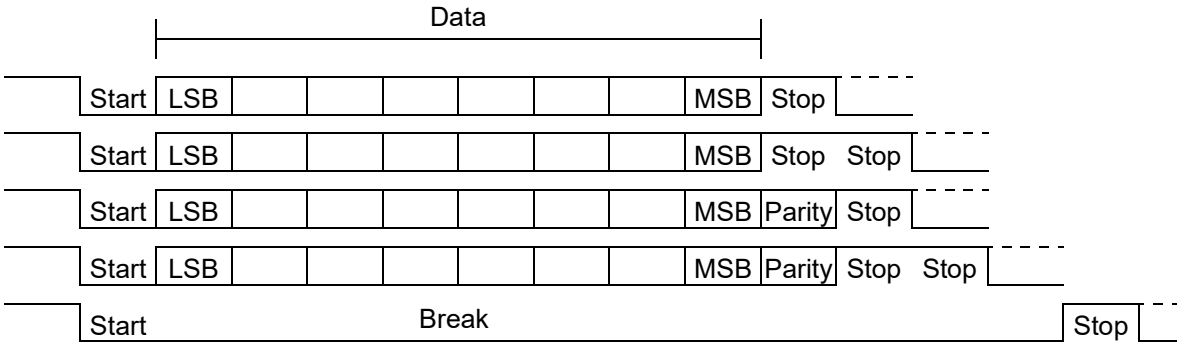


Figure 356. Asynchronous bit serial protocol / waveform

The handshaking for the bit asynchronous links is implemented with a busy signal. When a message is sent, the busy signal on the input link will be asserted as soon as the input interface is not ready to receive more data, it will then be deasserted as soon as the interface is ready to receive the next octet. The handshaking is continued through out the packet.

147.2 Registers

The core has not user accessible registers.

147.3 Vendor and device identifiers

The core has has neither a vendor identifier nor a device identifier.

147.4 Configuration options

Table 2499 shows the configuration options of the core (VHDL generics).

*Table 2499.*Configuration options

Generic	Function	Description	Allowed range	Default
gSystemClock	System frequency	[Hz]	Integer	8000000
gBaud	Baud rate	[Baud]	Integer	115200
gProgrammable	Baud rate	Enables programmable baud rate when 1	0 - 1	0
syncreset	Sync reset	Synchronous reset when 1	0 - 1	0

147.5 Signal descriptions

Table 2500 shows the interface signals of the core (VHDL ports).

*Table 2500.*Signal descriptions

Signal name	Field	Type	Function	Description	Active
RSTN	N/A	Input	Reset		Low
CLK	N/A	Input	Clock		-
IGNOREPARITY	N/A	Input		Generate odd parity, else none	High
TWOSTOPBITS	N/A	Input		Generate two stop bits, else one	High
BAUDTHRESH-OLD	N/A	Input		Sets baud rate, if gProgrammable is 1	-
VCIO	*	Output		Virtual Channel Interface	-
VCII	*	Input		Virtual Channel Interface	-
TMUART	N/A	Input		Data	-
TMBUSYN	N/A	Output		Not ready for octet	Low
TMRDY	N/A	Output		Ready for packet	High

147.6 Signal definitions and reset values

The signals and their reset values are described in table 2501.

*Table 2501.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
*tm*data*	Input	Data: This input port is the serial data input for the interface. Data are sampled on the rising *pw*clk* edge when *pw*valid_n* is asserted.	-	-
*tm*busyn*	Output	Not ready for octet: This port indicates whether the receiver is ready to receive one octet.	Low	Logical 1
*tm*ready*	Output	Not ready for octet: This port indicates whether the receiver is ready to receive one octet.	High	Logical 0

147.7 Timing

The timing waveforms and timing parameters are shown in figure 357 and are defined in table 2502.

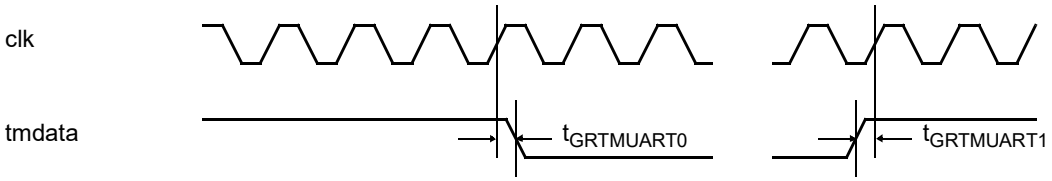


Figure 357. Timing waveforms

*Table 2502.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{GRTMUART0}$	hold from clock	rising *clk* edge	-	-	ns
$t_{GRTMUART1}$	setup to clock	rising *clk* edge	-	-	ns

Note: The inputs are re-synchronized internally. The signals do not have to meet any setup or hold requirements.

148 GEFFE - CCSDS/ECSS Telemetry Encoder - Geffe Generator

148.1 Overview

The Geffe Generator implements a simple stream based cipher to encrypt Telemetry Transfer Frames.

The Geffe generator acts as a keystream generator. It uses three Linear Feed-back Shift Registers (LFSR) combined in a nonlinear manner. Two of the LFSRs (LFSR1 and LFSR2) are inputs into a multiplexer, and the third LFSR (LFSR0) controls the output of the multiplexer. Suppose LFSR0, LFSR1, and LFSR2 are the outputs of the three LFSRs, then the generator output is as follows:

```
b(x) = (LFSR0 and LFSR1) xor (not LFSR0 and LFSR2)
```

The generator is programmable, in terms of the power of the polynomials used (i.e. stages or states), the polynomials (POLY0, POLY1 and POLY2) and the initialization values (INIT0, INIT1 and INIT2), individually for each of the three LFSRs.

The Geffe Generator interfaces with the GRTM CCSDS Telemetry Encoder via its external encryption/cipher interface located between the Data Link Protocol Sub-Layer and the Synchronization and Channel Coding Sub-Layer. This corresponds to Link Layer Security Option A according CCSDS definitions (see CCSDS 350.0-G-2 *"The application of CCSDS protocols to secure systems"*). This corresponds to Link Layer Security Option A according to CCSDS 350.0-G-2, i.e. between Logical Link Sublayer and Coding Sublayer.

The implementation of security services is below the Transfer Frame level. The full Transfer Frame is encrypted after any other necessary security services are applied. The generated keystream is XOR-ed bit wise with the Transfer Frame bit stream, starting with first bit transmitted. The LFSRs are only shifted when there is data to be encrypted. The shift is made after each bit has been encrypted.

The three LFSRs are initialized once when the generator is enabled, with the contents of the programmable INIT0, INIT1 and INIT2 register fields, respectively.

The LFSRs initialization value can be optionally XOR-ed with the Transfer Frame Master Channel Counter (only supported for hardware implemented Master Channel Counter in telemetry encoder), according to the following formula:

```
LFSR(31:24) = INIT(31:24) xor MCC(0:7),  LFSR(23:16) = INIT(23:16) xor MCC(0:7)
LFSR(15: 8) = INIT(15: 8) xor MCC(0:7),  LFSR( 7: 0) = INIT( 7: 0) xor MCC(0:7)
```

where INIT(31:0) is the programmable register field (bit 31 being the most significant), and MCC(0:7) is the Transfer Frame Master Channel Counter (bit 0 being the most significant).

The keystream is optionally restarted for each Transfer Frame, i.e. the LFSRs are re-initialized.

The Attached Synchronization Marker remains in plaintext to enable the ground systems to delimit the Channel Access Service Data Units. Also the Reed-Solomon check symbols are in plaintext, i.e. error detection & correction before decrypting begins.

Encryption for Idle Transfer Frames can be optionally disabled (only supported for hardware implemented Idle Transfer Frame generation in telemetry encoder).

A Cyclic Redundancy Code (CRC) can be optionally calculated over the encrypted data according to CCSDS/ECSS algorithm and replace the Frame Error Control Field (FECF) when transmitted in a Transfer Frame (only supported for hardware implemented CRC calculation in telemetry encoder).

The Geffe generator provides support for the following programmable functions:

- Enable/disable encryption on Transfer Frame boundaries
- Initialize LFSRs in-between Transfer Frames
- Initialize LFSRs taking Master Channel Counter value into account
- Skip encryption for Idle Transfer Frames
- Recalculate Cyclic Redundancy Code (CRC) and replace Frame Error Control Field (FECF)

148.2 Linear Feed-back Shift Registers

The three Linear Feed-back Shift Registers (LFSR) are Many-to-One implementation, i.e. Fibonacci version of LFRS. The One-to-Many implementation, i.e. Galois version of LFSR is not supported.

There are two mathematical representations for Fibonacci LFSR:

- output has the highest power, h(x)
- output has the lowest power, l(x)

The implementation of the Geffe Generator supports both representations. Note that both representations generate the same keystream.

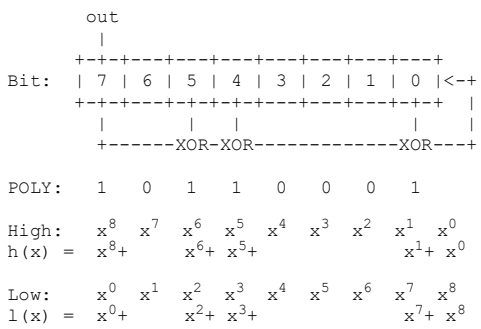
The LFSRs are always shifted to the left. The feedback is always entered to the right. The output bits corresponds to the leftmost set bits in the POLY fields in the polynomial configuration registers. The POLY field defined the feedback taps. The initialization values of the LFSRs is defined in the INIT fields in the polynomial initialization value configuration registers.

Note that the bit index of the configuration registers is not directly related to the polynomial powers. The interpretation of the bit index values depends on which of the above representations is assumed.

A first example of an LFSR configuration and the two possible interpretations is shown hereafter. The POLY field is set to 0x0000000B1. Since the leftmost bit set has index 7, the LFSR has 8 stages, and the output is taken from bit index 7, indicated with *out* in the figure.

Assuming that the output has the highest power, indicated with *High* and *h(x)*, the resulting polynomial is $h(x) = x^8 + x^6 + x^5 + x^1 + x^0$.

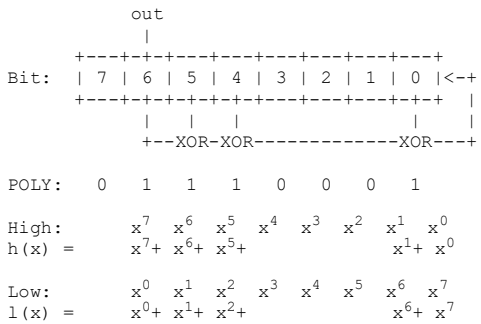
Assuming that the output has the highest power, indicated with *Low* and *l(x)*, the resulting polynomial is $l(x) = x^0 + x^2 + x^3 + x^7 + x^8$.



A second example of an LFSR configuration and the two possible interpretations is shown hereafter. The POLY field is set to 0x000000071. Since the leftmost bit set has index 6, the LFSR has 7 stages, and the output is taken from bit index 6, indicated with *out* in the figure.

Assuming that the output has the highest power, indicated with *High* and *h(x)*, the resulting polynomial is $h(x) = x^7 + x^6 + x^5 + x^1 + x^0$.

Assuming that the output has the highest power, indicated with *Low* and *l(x)*, the resulting polynomial is $l(x) = x^0 + x^1 + x^2 + x^6 + x^7$.



148.3 Connectivity

The connectivity of the Geffe Generator is tightly coupled to the functionality of the GRTM CCSDS Telemetry Encoder.

The input to the Geffe Generator can be connected to:

- Packet Telemetry and AOS encoder

The output from the Geffe Generator can be connected to:

- Reed-Solomon encoder
- Pseudo-Randomiser
- Non-Return-to-Zero Mark encoder
- Convolutional encoder
- Sub-Carrier modulator

148.4 Registers

The core is programmed through registers mapped into APB address space.

*Table 2503.*GEFFE registers

APB address offset	Register
0x00	GEFFE Control Register
0x10	GEFFE LFSR 0 Polynomial Configuration Register
0x14	GEFFE LFSR 1 Polynomial Configuration Register
0x18	GEFFE LFSR 2 Polynomial Configuration Register
0x20	GEFFE LFSR 0 Initialization Value Configuration Register
0x24	GEFFE LFSR 1 Initialization Value Configuration Register
0x28	GEFFE LFSR 2 Initialization Value Configuration Register

148.4.1 Control Register

*Table 2504.*0x00 - CTRL - GEFFE control register

31 13	12 8	7	6	5	4	3	2	1	0
RESERVED	LENGTH-1	ON	-	RI	MCC	IDLE	CRC	EN	RST
0	*	0		0	0	0	0	0	0
r	r	r		rw	rw	rw	rw	rw	rw

31: 13 RESERVED

12: 8 LFSR Length -1 (LENGTH-1) - the bit length of the implemented LFSR registers - 1 bit (read-only)

7 On-going (ON) - encryption of Transfer Frame is on-going when set, wait till cleared before changing ann configuration settings (read-only)

6 RESERVED

5 Re-initialize (RI) - when set, LFSRs are re-initialized in-between Transfer Frame (Reset value 0)

4 Master Channel Counter (MCC) - when set, LFSRs are initialized with the Transfer Frame Master Channel Counter value being XOR-ed with the INIT field values (Reset value 0)

3 Skip idle (IDLE) - when set, if an Idle Transfer Frame is transmitted, skip encryption (Reset value 0)

2 CRC replace (CRC) - when set, if FECF is transmitted in Transfer Frame, recalculates CRC value over encrypted data and replace FECF with the result (Reset value 0)

1 Enable (EN) - enable Geffe Generator when set (Reset value 0)

0 Reset (RST) - reset Geffe Generator when set. Does not reset configuration bits and polynomial and initialization registers. (Reset value 0)

148.4.2 LFSR Polynomial Configuration Registers 0 to 2

Table 2505. 0x10 - PcRn - GEFFE LFSR Polynomial Configuration Registers 0 to 2

31 0
POLY
0
rw

31: 0 LFSR Polynomial - Each bit in POLY corresponds to a tap in the LFSR. The LFSR is always shifted to the left. Feedback taps are always entered to the right, i.e. bit 0. The output bit corresponds to the leftmost set bit in POLY. Should not be changed while encryption is enabled. (Reset value all zero)

148.4.3 Initialization Value Configuration Registers 0 to 2

Table 2506. 0x20 - ICRn - GEFFE LFSR Initialization Value Configuration Registers 0 to 2

31 0
INIT
0
rw

31: 0 LFSR Initialization Value - The LFSR is initialized with this value, or optionally combined with the Transfer Frame Master Channel Counter value. Should not be changed while encryption is enabled. (Reset value all zero)

148.5 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x086. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

148.6 Configuration options

Table 2507 shows the configuration options of the core (VHDL generics).

*Table 2507.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR	0 - 16#FFF#	0
pmask	MASK field of the APB BAR	0 - 16#FFF#	16#FFF#
len	LFSR length	1 - 32	32

148.7 Signal descriptions

Table 2508 shows the interface signals of the core (VHDL ports).

*Table 2508.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
CI	CIPHER			
	ENABLE	Input	Encryption enabled from Telemetry Encoder	High
	DATA(0:7)		Parallel data, octet	-
	SYNC		ASM indicator	High
	FRAME		Frame indicator	High
	FECF		FECF/CRC indicator	High
	CHECK		Reed-Solomon checksymbol indicator	High
	TIME		Time strobe	High
	IDLE		Idle Transfer Frame indicator	High
	MCCNTR(0:7)		Transfer Frame Master Channel Counter	-
CO	CIPHER			
	ENABLE	Output	Unused	-
	DATA(0:7)		Parallel data, octet	-
	SYNC		ASM indicator	High
	FRAME		Frame indicator	High
	FECF		Unused	-
	CHECK		Reed-Solomon checksymbol indicator	High
	TIME		Time strobe	High
	IDLE		Unused	-
	MCCNTR(0:7)		Transfer Frame Master Channel Counter	-
OCLK	N/A	Input	Octet clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-

* see GRLIB IP Library User's Manual

148.8 Signal definitions and reset values

There are not external signals.

148.9 Timing

The are no external timing waveforms and timing parameters.

148.10 Library dependencies

Table 2509 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2509.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_TYPES	Signals, component	Component declaration

149 GRTMRX - CCSDS/ECSS Telemetry Receiver

149.1 Overview

The CCSDS/ECSS/PSS Telemetry Receiver implements part of the Data Link Layer, covering the Protocol Sub-layer and the Frame Synchronization and Coding Sub-layer and part of the Physical Layer of the packet telemetry protocol.

The operation of the Telemetry Receiver is highly programmable by means of control registers.

The Telemetry Receiver comprises several decoders and modulators implementing the Consultative Committee for Space Data Systems (CCSDS) recommendations, European Cooperation on Space Standardization (ECSS) and the European Space Agency (ESA) Procedures, Standards and Specifications (PSS) for telemetry and channel coding. The Telemetry Receiver comprises the following:

- Operation Control Field (OCF)
- Frame Error Control Field (FECF)
- Pseudo-De-Randomiser (PSR)
- Attached Sync Marker search (ASM)
- Non-Return-to-Zero Mark decoder (NRZ)
- Convolutional Quick-Look Decoder (CE)
- Split-Phase Level de-modulator (SP)
- Sub-Carrier de-modulator (SC)

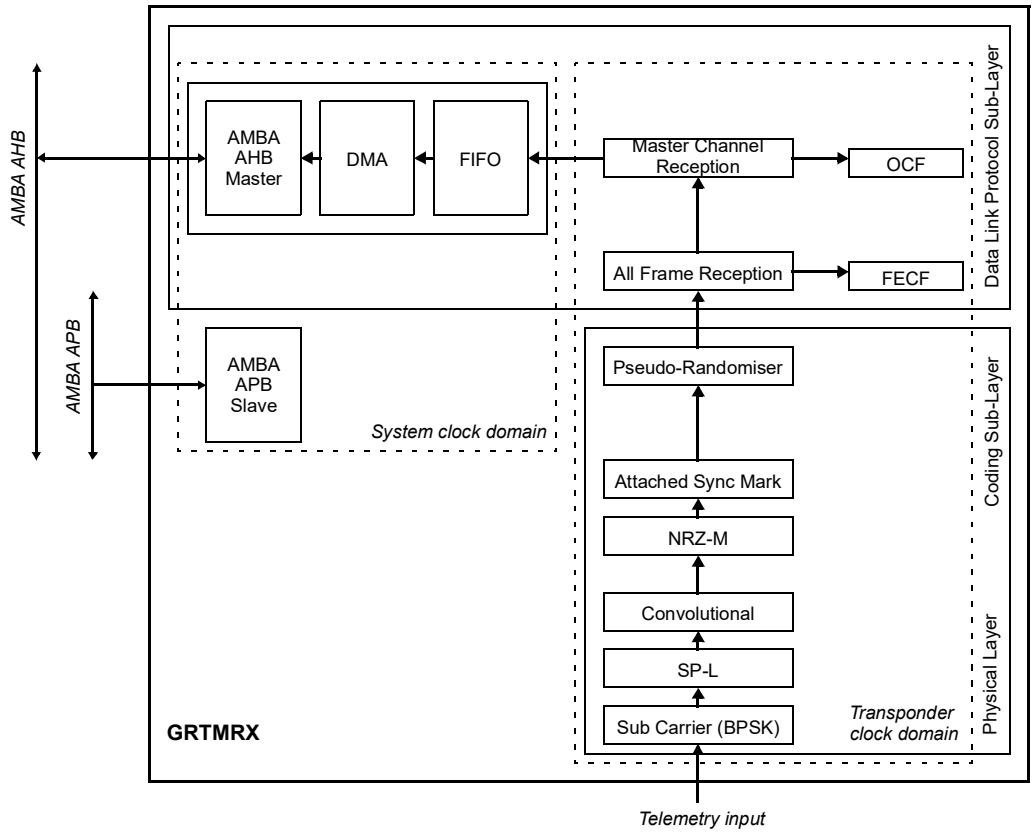


Figure 358. Block diagram

149.2 References

149.2.1 Documents

[C131] CCSDS 131.0-B-2 TM Synchronization and Channel Coding

[C132] CCSDS 132.0-B-1 TM Space Data Link Protocol

[C133] CCSDS 133.0-B-1 Space Packet Protocol

[C732] CCSDS 732.0-B-2 AOS Space Data Link Protocol

[ECSS01] ECSS-E-50-01C Space engineering - Space data links - Telemetry synchronization and channel coding

[ECSS03] ECSS-E-50-03C Space engineering - Space data links - Telemetry transfer frame protocol

[ECSS05] ECSS-E-50-05C Space engineering - Radio frequency and modulation

[PPS103] ESA PSS-04-103 Telemetry channel coding standard

[PPS105] ESA PSS-04-105 Radio frequency and modulation standard

[PPS106] ESA PSS-04-106 Packet telemetry standard

149.2.2 Acronyms and abbreviations

AOSAdvanced Orbiting Systems

ASMAttached Synchronization Marker

CCSDSConsultative Committee for Space Data Systems

CLCWCommand Link Control Word

CRCCyclic Redundancy Code

DMADirect Memory Access

ECSSEuropean Cooperation for Space Standardization

ESAEuropean Space Agency

FECFFrame Error Control Field

GF Galois Field

NRZNon Return to Zero

OCFOperational Control Field

PSR Pseudo Randomiser

PSS Procedures, Standards and Specifications

RS Reed-Solomon

SP Split-Phase

TE Turbo Encoder

TM Telemetry

149.3 Layers

149.3.1 Introduction

The Packet Telemetry (or simply Telemetry or TM) and Advanced Orbiting System (AOS) standards are similar in their format, with only some minor variations. The AOS part covered here is the down-link or transmitter, not the uplink or receiver.

The relationship between these standards and the Open Systems Interconnection (OSI) reference model is such that the OSI Data Link Layer corresponds to two separate layer, namely the Data Link Protocol Sub-layer and Synchronization and Channel Coding Sub-Layer. The OSI Data Link Layer is covered here.

The OSI Physical Layer is also covered here to some extended, as specified in [ECSS05] and [PPS105].

The OSI Network Layer or higher layers are not covered here.

149.3.2 Data Link Protocol Sub-layer

The following functionality is implemented in the core:

- Master Channel Reception / Virtual Channel Reception
  - •Operation Control Field (OCF) extraction
- All Frame Reception:
  - •Frame Error Control Field (FECF) extraction and check
  - •Frame filtering on the four first received octets (of the Transfer Frame after ASM)

149.3.3 Synchronization and Channel Coding Sub-Layer

The following functionality is implemented in the core:

- Attached Synchronization Marker (ASM) search
- Pseudo de-randomiser
- Convolutional quick-look decoding

149.3.4 Physical Layer

The following functionality is implemented in the core:

- Non-Return-to-Zero de-modulation
- Split-Phase de-modulation
- Sub-Carrier de-modulation

149.4 Operation

149.4.1 Introduction

The DMA interface provides a means for the user to receive blocks of data of arbitrary length, normally this is Telemetry Transfer Frames, with ASM and optional Reed-Solomon checkbits.

149.4.2 Descriptor setup

The receiver DMA interface is used for receiving data on the downlink. The reception is done using descriptors located in memory.

A single descriptor is shown in tables 2510 through 2513.

The number of bits to be received is set globally. The the address field of the descriptor should point to the start of the transfer frame. The address must be word-aligned. If the interrupt enable (IE) bit is set, an interrupt will be generated when the transfer frame has been received (this requires that the interrupt enable bit in the control register is also set). The interrupt will be generated regardless of whether the transfer frame was received successfully or not. The wrap (WR) bit is also a control bit that should be set before reception and it will be explained later in this section.

Table 2510. GRTMRX descriptor word 0 (address offset 0x0)

31 16	15 11	10	9	8	7	6 3	2	1	0
LEN	RESERVED	ALOCK	INV	CERR	OV	RESERVED	WR	IE	EN

31: 16	(LEN) - received length in bytes (read only)
15: 11	RESERVED
10:	ASM Lock (ALOCK) - (read only) Set to one when ASM search is in lock.
9:	Inverted bit stream (INV) - (read only) Set to one when an inverted bit stream detected.
8:	CRC Error (CERR) - (read only) Set to one when a CRC error was detected (speculative, only useful if FECF present in frame)
7:	Overrun (OV) - (read only) Set to one when an overrun has occurred during reception.
6: 3	RESERVED
2:	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 16. The pointer automatically wraps to zero when the 16 kB boundary of the descriptor table is reached.
1:	Interrupt Enable (IE) - an interrupt will be generated when the frame from this descriptor has been sent provided that the receiver interrupt enable bit in the control register is set. The interrupt is generated regardless if the frame was received successfully or if it terminated with an error.
0:	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.

Table 2511. GRTMRX descriptor word 1 (address offset 0x4)

31 2	1	0
ADDRESS	-	

31: 2	Address (ADDRESS) - Pointer to the buffer area to where data will be stored.
1: 0	RESERVED

Table 2512. GRTMRX descriptor word 2 (address offset 0x8)

31 0
STATUS0

31: 0	(STATUS0) - External status information

Table 2513. GRTMRX descriptor word 3 (address offset 0xC)

31 0
STATUS1

31: 0	(STATUS1) - External status information

To enable a descriptor the enable (EN) bit should be set and after this is done, the descriptor should not be touched until the enable bit has been cleared by the core.

149.4.3 Starting reception

Enabling a descriptor is not enough to start reception. A pointer to the memory area holding the descriptors must first be set in the core. This is done in the descriptor pointer register. The address must be aligned to a 16 kByte boundary. Bits 31 to 14 hold the base address of descriptor area while

bits 13 to 4 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the core, the pointer field is incremented by 16 to point at the next descriptor. The pointer will automatically wrap back to zero when the next 16 kByte boundary has been reached. The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 16 kByte boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when reception is active.

The final step to activate the reception is to set the enable bit in the DMA control register. This tells the core that there are more active descriptors in the descriptor table. This bit should always be set when new descriptors are enabled, even if reception is already active. The descriptors must always be enabled before the reception enable bit is set.

149.4.4 Descriptor handling after reception

When the reception of a frame has finished, status is written to the first word in the corresponding descriptor. The other bits in the first descriptor word are set to zero after reception. The enable bit should be used as the indicator when a descriptor can be used again, which is when it has been cleared by the core. Additionally, the last two words in the corresponding descriptor are written with external status information.

There are multiple bits in the DMA status register that hold status information.

The Receiver Interrupt (RI) bit is set each time a DMA reception ended successfully. The Receiver Error (RE) bit is set each time an DMA reception ended with an overrun error. For either event, an interrupt is generated for descriptor for which the Interrupt Enable (IE) was set. The interrupt is maskable with the Interrupt Enable (IE) bit in the control register.

The Receiver AMBA error (RA) bit is set when an AMBA AHB error was encountered either when reading a descriptor or when writing data. Any active reception was aborted and the DMA channel was disabled. It is recommended that the receiver is reset after an AMBA AHB error. The interrupt is maskable with the Interrupt Enable (IE) bit in the control register.

149.4.5 Demodulator clock recovery

Demodulators are provided, that support Split-Phase Level (SP) demodulation (SP), and Binary Phase Shift Key (BPSK) demodulation (or Phase Shift Key Square) as per [ECSS-E-ST-50-05C]. The demodulators can operate separately or in combination.

The demodulators can operate either with a bit serial clock input signal, or with just the bit serial data stream.

For the former, it is assumed that a bit serial clock input, which is synchronous with the bit serial data input, is provided. The demodulator can decode symbol rates up the system clock frequency.

For the latter, a bit serial clock signal can be re-generated locally for a bit serial data input that is Split-Phase-L (SP-L) and/or Sub-Carrier (SC) modulated. The clock recovery is enabled by setting the EN bit in the GRTMRX Demodulator register. With clock recovery, the demodulator can decode a bit serial data input where the symbol period has a duration of down to four times the system clock period. Note that when Sub Carrier (SC) modulation is used, a symbol corresponds to a single phase of the sub-carrier waveform, thus the sub-carrier frequency can be up to an eighth of the system clock frequency.

149.5 Registers

The core is programmed through registers mapped into APB address space.

*Table 2514.*GRTMRX registers

APB address offset	Register
0x00	GRTMRX DMA Control register
0x04	GRTMRX DMA Status register
0x08	GRTMRX DMA Descriptor Pointer register
0x80	GRTMRX Control register
0x84	GRTMRX Status register
0x88	GRTMRX Configuration register
0x8C	GRTMRX Size register
0x90	GRTMRX Physical Layer register
0x94	GRTMRX Coding Sub-Layer register
0x98	GRTMRX Attached Synchronization Marker register
0x9C	GRTMRX Attached Synchronization Marker Mask register
0xAC	GRTMRX Data Rate register
0xB0	GRTMRX Filter register
0xB4	GRTMRX Filter Mask register
0xD0	GRTMRX Operational Control Field register
0xD4	GRTMRX Frame Error Control Field register
0xD8	GRTMRX Demodulator register

149.5.1 GRTMRX DMA Control Register

*Table 2515.*0x00 - DCR - DMA control register

31 2	1	0
RESERVED	IE	EN
0	0	0
r	rw	rw

31: 2 RESERVED

1: Interrupt Enable (IE) - enable interrupts RA, RI, and RE

0: Enable (EN) - enable DMA transfers

149.5.2 GRTMRX DMA Status Register

*Table 2516.*0x04 - DSR - DMA status register

31 4	3	2	1	0
RESERVED	ACTIVE	RA	RI	RE
0	NR	0	0	0
r	r	wc	wc	wc

31: 4 RESERVED

3: Active (ACTIVE) - DMA access ongoing

2: Receiver AMBA Error (RA) - DMA AMBA AHB error, cleared by writing a logical 1

1: Receiver Interrupt (RI) - DMA interrupt, cleared by writing a logical 1

0: Receiver Error (RE) - DMA receiver error (e.g.underrun), cleared by writing a logical 1

149.5.3 GRTMRX DMA Descriptor Pointer Register

Table 2517. 0x08 - DPR - DMA descriptor pointer register

31 14	13 4	3 0
BASE	INDEX	RESERVED
NR	NR	0
rw	rw	r

31: 14 Descriptor base (BASE) - base address of descriptor table

13: 4 Descriptor index (INDEX) - index of active descriptor in descriptor table

3: 0 Reserved - fixed to “0000”

149.5.4 GRTMRX Control Register

Table 2518. 0x80 - CTRL - control register

31 4	3	2	1	0
RESERVED	ScRST	RST	RxRST	RxEN
0	0	0	1	0
r	rw	rw	rw	rw

31: 4 RESERVED

3: Sub Carrier Reset (ScRST) - resets sub carrier receiver

2: Reset (RST) - resets complete core

1: Receiver Reset (RxRST) - resets telemetry receiver

0: Receiver Enable (RxEN) - enables telemetry receiver (should be done after the complete configuration of the telemetry receiver)

149.5.5 GRTMRX Status Register (read-only)

Table 2519. 0x84 - STAT - status register (read-only)

31 15	14	13 12	11	10	9	8	7	6	5	4	3	2	1 0
RESERVED	DEBUG	ASM Cntr	R	CRC Error	Invert	DLock	Sync Mark	Frame Mark	Frame Ready	ASM Free	ASM Lock	ASM Sync	State
0	*	0	0	0	0	1	*	*	0	0	0	0	0
r	r	r	r	r	r	r	r	r	r	r	r	r	r

31: 15 RESERVED

14: (DEBUG) - Indicates when set that additional debug information is available in bit 31:15.

13: 12 (ASMCTNR) - ASM counter (diagnostics)

11: RESERVED

10: (CRCERR) - CRC error (speculative, only useful if FECF present in frame) [1]

9: (INVERT) - Inverted bit stream (diagnostics)

8: (DLOCK) - Convolutional decoder in lock (diagnostics)

7: (SYNCMARK) - Sync Marker (external signal status, if available)

6: (FRAMEMARK) - Frame Marker (external signal status, if available)

5: (FRAMEREADY) - Frame ready (diagnostics)

4: (ASMFREE) - ASM free running (diagnostics) [1]

3: (ASMLOCK) - ASM in lock (diagnostics) [1]

2: (ASMSYNC) - ASM synchronized (diagnostics)[1]

1: 0 Receiver state (STATE) - 00=Idle, 01=Search, 10=Filter, 11=Receive (diagnostics)

149.5.6 GRTMRX Configuration Register

Table 2520. 0x88 - CONF - configuration register

31 24	23 8	7 3	2	1	0
REVISION	FIFOSIZE	RESERVED	Free Wheel	Raw	Endian
1	25b	0	0	0	0
r	r	r	rw	rw	rw

31: 24 (REVISION) - Revision number

23: 8 (FIFOSIZE) - FIFO size bytes

7: 3 RESERVED

2: (FREEWHEEL) - Free wheeling enabled when set (i.e. accepts missing ASM) (not to be used with RAW set or MATCH and IGNORE usage)

1: (RAW) - Raw data capture when set (no ASM search)

0: (ENDIAN) - Little Endian when set, Big Endian when cleared

149.5.7 GRTMRX Size Register

Table 2521. 0x8C - SIZE - size register

31 29	28 16	15 0
RESERVED	OCFINDEX	NOOFBITS
0	0xE3	0x718
r	rw	rw

31: 29 RESERVED

28: 16 (OCFINDEX) Defines the byte number of the last octet of OCF, counting from the ASM start [1]

Defines the byte number of the octet preceding FECF, counting from the ASM start [1]

15: 0 (NOOFBITS) - Number of bits for block to receive (e.g. including ASM, Reed-Solomon code, etc.)

149.5.8 GRTMRX Physical Layer Register

Table 2522. 0x90 - PLR - physical layer register

31	30 16	15	14 0
SF	SYMBOLRATE	SCF	SUBRATE
0	0	0	0
rw	r	rw	r

31 Symbol Fall (SF) - symbol clock has a falling edge at start of symbol bit

30: 16 Symbol Rate (SYMBOLRATE) - symbol rate division factor -1 (read-only) (diagnostics)

When Sub Carrier (SC) is disabled, this field corresponds to the lower 15 bits of the number of system clock periods -1 counted per incoming symbol clock period duration.

When Sub Carrier (SC) is enabled, this field corresponds to the number of system clock periods -1 counted per incoming symbol clock period duration.

15 Sub Carrier Fall (SCF) -sub carrier output start with a falling edge for logical 1

14: 0 Sub Carrier Rate (SUBRATE) - sub carrier division factor - 1 (read only) (diagnostics)

When Sub Carrier (SC) is disabled, this field corresponds to the upper 15 bits of the number of system clock periods -1 counted per incoming symbol clock period duration.

When Sub Carrier (SC) is enabled, this field corresponds to the number of sub-carrier clock phases - 1 counted per incoming symbol bit duration.

Telemetry Rate (TELEMTRY RATE) - is the bit rate after potential Sub Carrier (SC) demodulation, Split-Phase Level (S) demodulation and Convolutional Encoding (CE) decoding, and corresponds to the telemetry bit rate.

The following variables are used in the tables hereafter: f = system clock frequency, n = SYMBOLRATE+1 (GRTMRX physical layer register field +1), and m = SUBRATE+1 (GRTMRX physical layer register field +1).

When Sub Carrier (SC) is disabled, the telemetry rate equals

TELMETRY RATE = (SUBRATE * 2**15 + SYMBOLRATE+1) / (1+CE) / (1+SP) = m * n / (1+CE) / (1+SP)

When Sub Carrier (SC) is enabled, the telemetry rate equals

TELEMETRY RATE = (SUBRATE +1)* (SYMBOLRATE+1) / (1+CE) / (1+SP) = m * n / (1+CE) / (1+SP)

149.5.9 GRTMRX Coding Sub-Layer Register

Table 2523. 0x94 - CSL - coding sub-layer register

31 19	18 17	16 8	7	6	5	4 2	1	0
RESERVED	SEL	RESERVED	PSR	NRZ	CE	RESERVED	SP	SC
0	0	0	0	0	0	0	0	0
r	rw	r	rw	rw	rw	r	rw	rw

31: 19 RESERVED

18: 17 Selection (SEL) - selection of external telemetry clock and data source (application specific)

16: 8 RESERVED

7: Pseudo-Randomiser (PSR) - pseudo-de-randomiser enable [1]

6: Non-Return-to-Zero (NRZ) - non-return-to-zero - mark decoding enable

5: Convolutional Encoding (CE) - convolutional decoding enable

4: 2 RESERVED

1: Split-Phase Level (SP) - split-phase level de-modulation enable

0: Sub Carrier (SC) - sub carrier de-modulation enable

149.5.10GRTMRX Attached Synchronization Marker Register

Table 2524. 0x98 - ASM - attached synchronization marker register

31	0
ASM	
0xIACFFCID	
rw	

31: 0 Attached Synchronization Marker (ASM) - pattern for ASM, (bit 31 MSB sent first, bit 0 LSB sent last), reset values 0x1ACFFC1D [1]

149.5.11GRTMRX Attached Synchronization Mask Register

Table 2525. 0x9C - AMM - attached synchronization mask register

31	0
ASMMASK	
0xFFFFFFFF	
r	

31: 0 Attached Synchronization Marker Mask (ASMMASK) - mask for ASM pattern, bit used when set [1]

149.5.12GRTMRX Data Rate Register (read-only)

Table 2526. 0xAC - DRR - data rate register (read-only)

31	30	29	0
R		DATARATE	
0		0	
r		r	

31: 30 RESERVED

29: 0 (DATARATE) - data rate division factor -1 (read-only) (diagnostics)

This field corresponds to the number of system clock periods -1 counted per incoming telemetry bit period duration.

149.5.13GRTMRX Filter Register

Table 2527. 0xB0 - FLT - Filter register

31	16	15	0
MATCH		IGNORE	
0		0	
rw		rw	

31: 16 (MATCH) - Matching pattern for the first two received octets after ASM (bit 31 MSB, bit 16 LSB) [1]

15: 0 (IGNORE) - Non-matching pattern for the first two received octets after ASM (bit 15 MSB, bit 0 LSB) [1]

149.5.14GRTMRX Filter Mask Register

Table 2528. 0B4 - FLTM - Filter Mask register

31 16	15 0
MATCHMASK	IGNOREMASK
0	0
rw	rw

31: 16 (MATCHMASK) - Mask for matching pattern (bit 31 MSB, bit 16 LSB), bit used when set [1]

15: 0 (IGNOREMASK) - Mask for non-matching pattern (bit 15 MSB, bit 0 LSB), bit used when set [1]

149.5.15GRTMRX OCF Register (read-only)

*Table 2529.*0x00 - OCF - OCF register (read-only)

31 0
OCF
0xFFFFFFFF
r

31: 0 Operational Control Field (OCF) - OCF/CLCW data (bit 31 MSB, bit 0 LSB) [1]

149.5.16GRTMRX FECF Register (read-only)

Table 2530. 0x04 - FECF - FECF register (read-only)

31 16	15 0
RESERVED	FECF
0	0xFFFF
r	r

31: 16 RESERVED

15: 0 Frame Error Control Field (FECF) - FECF/CRC data (bit 15 MSB, bit 0 LSB) [1]

Note 1: This function does is not supported in raw capture mode, when the RAW register bit is set.

149.5.17GRTMRX Demodulator Register

*Table 2531.*0x08 - DEM - Demodulator register

31	30	29 0
EN	DM	SYMBOLRATE
0	1	*
rw	r	r

31: Enable (EN) - Enable clock recovery demodulator when logical 1

30: Demodulator (DM) - Clock recovery demodulator implemented if logical 1(read-only)

29: 0 Symbol Rate (SYMBOLRATE) - Symbol rate division factor -1 (read-only) (diagnostics)

This field corresponds to the number of system clock periods -1 counted per incoming symbol duration (i.e. locally generated symbol clock period). Note that when Sub Carrier (SC) modulation is used, a symbol corresponds to one phase of the sub-carrier clock.

149.6 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x082. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

149.7 Implementation

149.7.1 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

149.8 Configuration options

Table 2532 shows the configuration options of the core (VHDL generics).

*Table 2532.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR	0 - 16#FFF#	0
pmask	MASK field of the APB BAR	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by core	0 - NAHBIRQ-1	0
memtech	Memory technology	0 to NTECH	0
clktech	Clock buffer technology	0 to NTECH	0
buftype	Clock buffer type	TBD	0
demod	Demodulator for local clock recovery	0 - 1	0

149.9 Signal descriptions

Table 2533 shows the interface signals of the core (VHDL ports).

*Table 2533.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
TMI	BITLOCK	Output	Bit Lock	High
	RFAVAIL		RF Available	High
	OCF_SDU[0:31]		OCF_SDU Bypass	-
	CIPHER		Encryption/ Cipher Interface	-
	EXREQUEST		External VC Request	High
	CONF		External configuration	-

*Table 2533.*Signal descriptions

Signal name	Field	Type	Function	Active
TMO	TIME	Input	Time strobe	High
	SYNC		ASM indicator	High
	FRAME		Frame indicator	High
	SERIAL		Serial bit data	High
	CLOCK		Serial bit data clock	High
	DATA [0:7]		Parallel data, octet	High
	STROBE		Parallel data strobe	High
	C1		C1 bit from convolutional encoder	-
	C2		C2 bit from convolutional encoder	-
	CLKSEL[0:1]		External clock selection	-
	FRAMELSB[0:1]		MC Counter least significant bits	-
	CIPHER		Encryption/ Cipher Interface	-
	SCID[0:9]		SCID setting	-
	OCF		OCF setting	High
	FECF		FECF setting	High
	EXENABLE		External VC enable	High
	EXGRANT		External VC grant	High
	EXREADY		External VC ready	High
	TIMESTAMP		Time Stamp	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-

* see GRLIB IP Library User's Manual

149.10 Signal definitions and reset values

The signals and their reset values are described in table 2534.

*Table 2534.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
sync	Input	ASM indicator	High	-
frame	Input	Frame indicator	High	-
serial	Input	Serial bit data	-	-
clock	Input	Serial bit data clock	High	-

149.11 Timing

The timing waveforms and timing parameters are shown in figure 359 and are defined in table 2535.

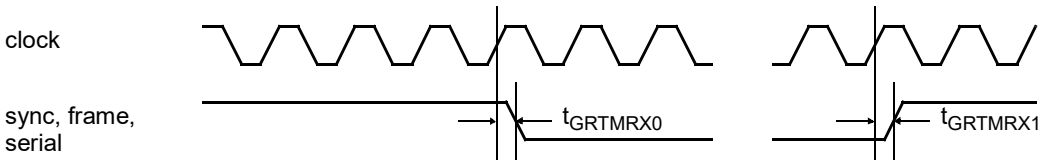


Figure 359. Timing waveforms

*Table 2535.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{GRTMRX0}$	input to clock hold	rising *clock* edge	TBD	TBD	ns
$t_{GRTMRX1}$	input to clock setup	rising *clock* edge	TBD	TBD	ns

149.12 Library dependencies

Table 2536 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2536.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_TYPES	Signals, component	Component declaration

150 GRCE/GRCD - CCSDS/ECSS Convolutional Encoder and Quicklook Decoder

The *Basic Convolutional Encoder* (GRCE) comprises a synchronous bit serial input and a synchronous bit serial output. The output frequency is twice the input frequency.

The *Basic Convolutional Quicklook Decoder* (GRCD) comprises a synchronous bit serial input and a synchronous bit serial output. The input frequency is twice the output frequency. The quicklook decoder decodes the incoming bit stream without correcting for bit errors.

The GRCE / GRCD models are based on the Basic Convolutional Code specified by *Consultative Committee for Space Data Systems* (CCSDS) and *European Cooperation for Space Standardization* (ECSS).

[CCSDS] TM Synchronization and Channel Coding, CCSDS 131.0-B-2

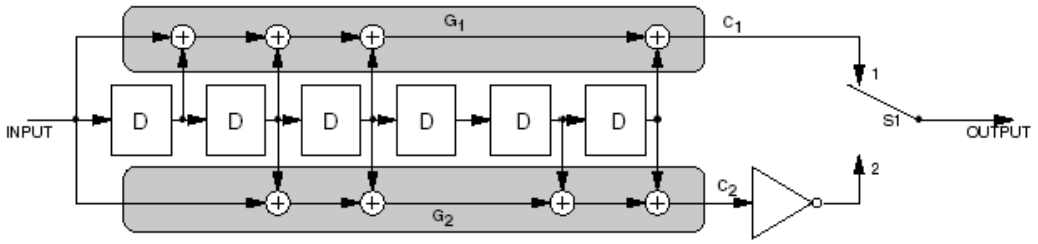
[PSS] Telemetry Channel Coding Standard, ESA PSS-04-103, Issue 1, September 1989

[ECSS] Telemetry synchronization and channel coding, ECSS-E-ST-50-01C

150.1 Protocol

The basic convolutional code is a rate 1/2, constraint-length 7 transparent code which is well suited for channels with predominantly Gaussian noise:

Nomenclature:	Convolutional code
Code rate:	1/2 bit per symbol.
Constraint length:	7 bits.
Connection vectors:	G1 = 1111001 (171 octal); G2 = 1011011 (133 octal).
Symbol inversion:	On output path of G2.



NOTES:

1. → D → = SINGLE BIT DELAY.
2. FOR EVERY INPUT BIT, TWO SYMBOLS ARE GENERATED BY COMPLETION OF A CYCLE FOR S1: POSITION 1, POSITION 2.
3. S1 IS IN THE POSITION SHOWN (1) FOR THE FIRST SYMBOL ASSOCIATED WITH AN INCOMING BIT.
4. ⊕ = MODULO-2 ADDER.
5. → ▷○ = INVERTER.

150.2 Configuration options

Table 2537 shows the configuration options of the cores (VHDL generics).

*Table 2537.*Configuration options

Generic name	Function	Allowed range	Default
syncreset	Synchronous reset when set, else asynchronous	0 - 1	0

150.3 Signal descriptions

Table 2539 shows the interface signals of the Basic Convolutional Encoder (GRCE) core (VHDL ports).

*Table 2538.*Signal descriptions - GRCE

Signal name	Field	Type	Function	Active
Rst_N	N/A	Input	This active low input port synchronously resets the model. The port is assumed to be deasserted synchronously with the ***Cout*** system clock.	Low
Cin	N/A	Input	This input port is the bit clock for the data input ***Din***. The port is sampled on the rising ***Cout*** edge. When *Cin* is sampled as asserted, a new bit is present on the ***Din*** port. ***Cin*** is assumed to have been generated from the rising ***Cout*** edge, normally with a delay, and ***Din*** is assumed to be stable after the falling ***Cin*** edge.	High
Din	N/A	Input	This input port is the serial data input for the interface. Data are sampled on the rising ***Cout*** edge when the ***Cin*** input is asserted. Input data ***Din*** is thus qualified by the input bit clock ***Cin***. For each input data bit on ***Din***, two bits are output on ***Dout***.	-
Cout	N/A	Output	This input port is the system clock for the model. All registers are clocked on the rising ***Cout*** edge. The port also acts as the bit clock for the data output ***Dout***.	Rising
Dout	N/A	Output	This output port is the serial data output for the interface. The output is clocked out on the rising ***Cout*** edge.	-

Table 2539 shows the interface signals of the GRCD core (VHDL ports).

*Table 2539.*Signal descriptions - GRCD

Signal name	Field	Type	Function	Active
Rst_N	N/A	Input	This active low input port synchronously resets the model. The port is assumed to be deasserted synchronously with the ***Cin*** system clock.	Low
Cin	N/A	Input	This input port is the system clock for the model. All registers are clocked on the falling ***Cin*** edge. The port also acts as the bit clock for the data input ***Din***.	Falling
Din	N/A	Input	This input port is the serial data input for the interface. Data are sampled on the falling ***Cin*** edge. For two input data bits on ***Din***, one bit is output on ***Dout***.	-
Cout	N/A	Output	This output port is the output bit clock. The output is clocked out on the falling ***Cin*** edge.	-
Dout	N/A	Output	This output port is the serial data output for the interface. The output is clocked out on the falling ***Cin*** edge. ***Dout*** is assumed to be sampled externally on the falling ***Cout*** edge.	-
Dlock	N/A	Output	This output port is asserted when the quick look decoder is in lock and producing decoded data. The output is clocked out on the falling ***Cin*** edge.	High

150.4 Signal definitions and reset values

The signals and their reset values for the Basic Convolutional Encoder (GRCE) are described in table 2540.

*Table 2540.*Signal definitions and reset values - GRCE

Signal name	Type	Function	Active	Reset value
cin	Input	Input data qualifier	High	-
din	Input	Input data	-	-
cout	Input	Bit clock	Rising	-
dout	Output	Output data	-	Logical 0

The signals and their reset values for the Basic Convolutional Quicklook Decoder (GRCD) are described in table 2540.

*Table 2541.*Signal definitions and reset values - GRCD

Signal name	Type	Function	Active	Reset value
cin	Input	Bit clock	Falling	-
din	Input	Input data	-	-
cout	Output	Output bit clock	High	Logical 0
dout	Output	Output data	-	Logical 0
dlock	Output	Decoder in lock	High	Logical 0

150.5 Timing

The timing waveforms and timing parameters for the Basic Convolutional Encoder (GRCE) are shown in figure 360 and are defined in table 2542.

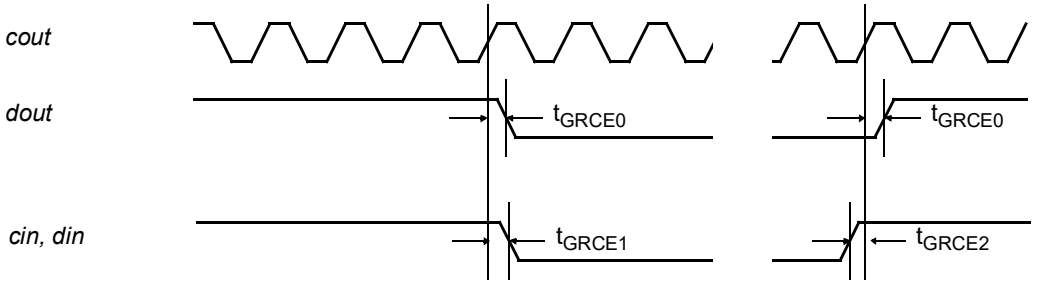


Figure 360. Timing waveforms - GRCE

*Table 2542.*Timing parameters - GRCE

Name	Parameter	Reference edge	Min	Max	Unit
t_{GRCE0}	clock to output delay	rising *cout* edge	TBD	TBD	ns
t_{GRCE1}	input to clock hold	rising *cout* edge	-	-	ns
t_{GRCE2}	input to clock setup	rising *cout* edge	-	-	ns

The timing waveforms and timing parameters for the Basic Convolutional Quicklook Decoder (GRCD) are shown in figure 360 and are defined in table 2542.

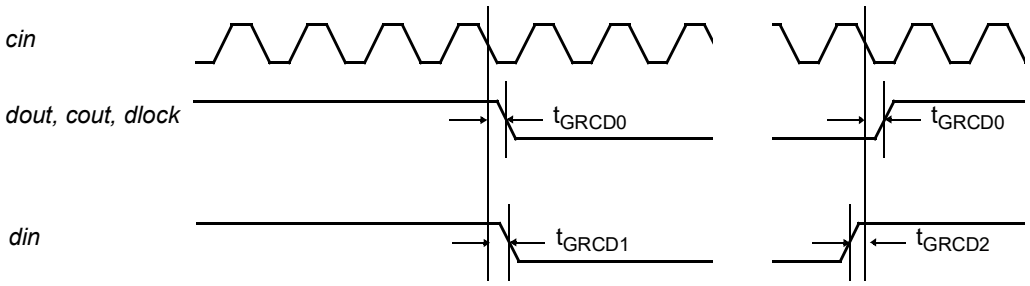


Figure 361. Timing waveforms - GRCD

*Table 2543.*Timing parameters - GRCD

Name	Parameter	Reference edge	Min	Max	Unit
t_{GRCD0}	clock to output delay	rising *cin* edge	TBD	TBD	ns
t_{GRCD1}	input to clock hold	rising *cin* edge	-	-	ns
t_{GRCd2}	input to clock setup	rising *cin* edge	-	-	ns

150.6 Library dependencies

Table 2544 shows the libraries used when instantiating the cores (VHDL libraries).

*Table 2544.*Library dependencies

Library	Package	Imported unit(s)	Description
TMTC	TMTC_Types	Component	Component declaration

150.7 Instantiation

The GRCE/ GRCD cores are fully synchronous designs based on a single clock strategy. All registers in the cores are reset synchronously or asynchronously, controlled by the syncrst VHDL generic. The reset input requires external synchronisation to avoid any setup and hold time violations.

This example shows how the cores can be instantiated.

```
   library  IEEE;
   use      IEEE.Std_Logic_1164.all;
   library  TMTC;

...

   component GRCE
   port(
      Rst_n:      in    Std_ULogic; -- Synchronous reset
      Cin:        in    Std_ULogic; -- Input  data clock
      Din:        in    Std_ULogic; -- Input  data
      Cout:       in    Std_ULogic; -- Output data clock
      Dout:       out   Std_ULogic);-- Output data
   end component GRCE;

   component GRCD
   port(
      Rst_N:      in    Std_ULogic; -- Synchronous reset
      Cin:        in    Std_ULogic; -- Input  data clock
      Din:        in    Std_ULogic; -- Input  data
      Cout:       out   Std_ULogic; -- Output data clock
      Dout:       out   Std_ULogic; -- Output data
      Dlock:      out   Std_ULogic);-- Output locked
   end component GRCD;
```

151 GRTC - CCSDS/ECSS Telecommand Decoder

151.1 Overview

The Telecommand Decoder (GRTC) is compliant with the Packet Telecommand protocol and specification defined by [ECSS-E-ST-50-04C]. The decoder is compatible with the [PSS-04-107] and [PSS-04-151] standards. The decoder is compatible with the CCSDS recommendations [CCSDS-231.0-B-2], [CCSDS-232.0-B-2] and [CCSDS-232.1-B-2]. The Telecommand Decoder (GRTC) only implements the Coding Layer (CL).

In the Coding Layer (CL), the telecommand decoder receives bit streams on multiple channel inputs. The streams are assumed to have been generated in accordance with the Physical Layer specifications. In the Coding Layer, the decoder searches all input streams simultaneously until a start sequence is detected. Only one of the channel inputs is selected for further reception. The selected stream is bit-error corrected and the resulting corrected information is passed to the user. The corrected information received in the CL is transfer by means of Direct Memory Access (DMA) to the on-board processor.

The Command Link Control Word (CLCW) and the Frame Analysis Report (FAR) can be read and written as registers via the AMBA AHB bus. Parts of the two registers are generated by the Coding Layer (CL). The CLCW can be automatically transmitted to the Telemetry Encoder (TM) for transmission to the ground. Note that most parts of the CLCW and FAR are not produced by the Telecommand Decoder (GRTC) hardware portion. This is instead done by the software portion of the decoder.

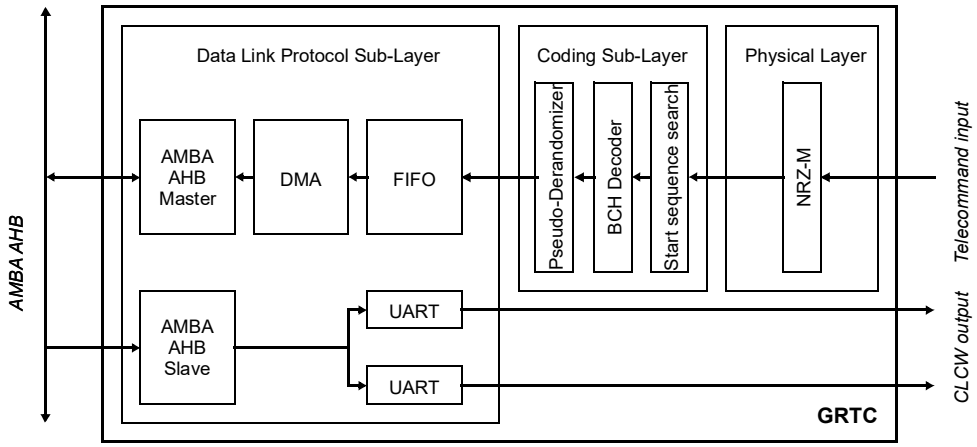


Figure 362. Block diagram

151.1.1 Concept

A telecommand decoder in this concept is mainly implemented by software in the on-board processor. The supporting hardware in the GRTC core implements the Coding Layer, which includes synchronisation pattern detection, channel selection, codeblock decoding, Direct Memory Access (DMA) capability and buffering of corrected codeblocks. The hardware also provides a register via which the Command Link Control Word (CLCW) is made available to a Telemetry Encoder. The CLCW is to be generated by the software.

The GRTC has been split into several clock domains to facilitate higher bit rates and partitioning. The two resulting sub-cores have been named Telecommand Channel Layer (TCC) and the Telecommand Interface (TCI). Note that TCI is called AHB2TCI. A complete CCSDS packet telecommand decoder can be realized at software level according to the latest available standards, staring from the Transfer Layer.

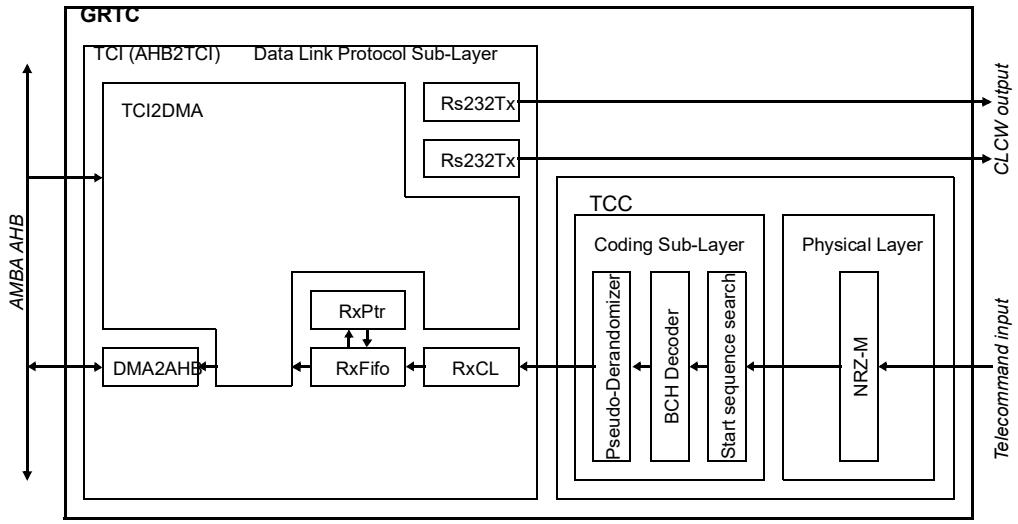


Figure 363. Detailed block diagram showing the internal structure

151.1.2 Functions and options

The Telecommand Decoder (GRTC) only implements the Coding Layer of the telecommand protocol standard [ECSS-E-ST-50-04C]. All other layers are to be implemented in software, e.g. Authentication Unit (AU). The Command Pulse Distribution Unit (CPDU) is not implemented.

The following functions of the GRTC are programmable by means of registers:

- Pseudo-De-Randomisation
- Non-Return-to-Zero – Mark decoding

The following functions of the GRTC are pin configurable:

- Polarity of RF Available and Bit Lock inputs
- Edge selection for input channel clock

151.2 Data formats

151.2.1 Reference documents

[PSS-04-107]	Packet Telecommand Standard, Issue 2
[PSS-04-151]	Telecommand Decoder Standard, Issue 1
[CCSDS 231.0-B-2]	TC Synchronization and Channel Coding
[CCSDS 232.0-B-2]	TC Space Data Link Protocol
[CCSDS 232.1-B-2]	Communications Operation Procedure-1
[ECSS-E-ST-50-04C]	Space engineering - Space data links - Telecommand protocols, synchronization and channel coding

151.2.2 Waveforms

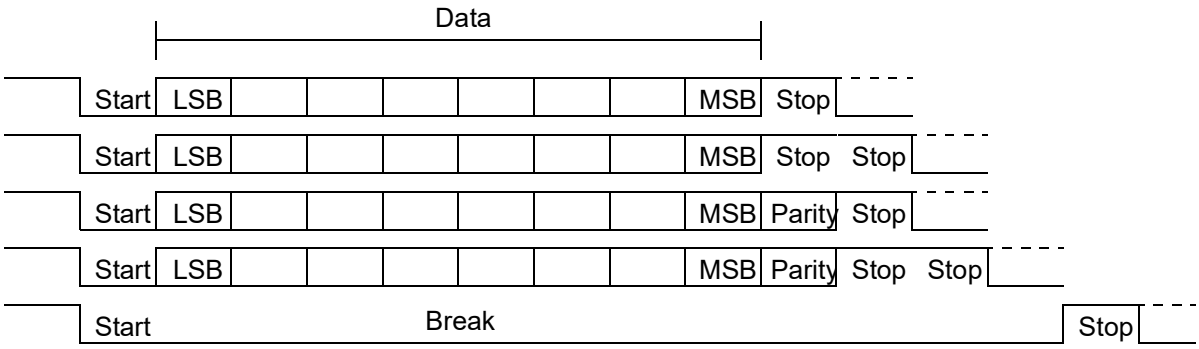


Figure 364. Bit asynchronous protocol versions

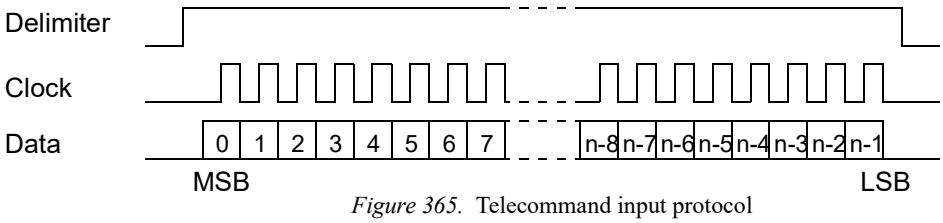


Figure 365. Telecommand input protocol

151.3 Coding Layer (CL)

The Coding Layer synchronises the incoming bit stream and provides an error correction capability for the Command Link Transmission Unit (CLTU). The Coding Layer receives a dirty bit stream together with control information on whether the physical channel is active or inactive for the multiple input channels.

The bit stream is assumed to be NRZ-L encoded, as the standards specify for the Physical Layer. As an option, it can also be NRZ-M encoded. There are no assumptions made regarding the periodicity or continuity of the input clock signal while an input channel is inactive. The most significant bit (Bit 0 according to [ECSS-E-ST-50-04C]) is received first.

Searching for the Start Sequence, the Coding Layer finds the beginning of a CLTU and decodes the subsequent codeblocks. As long as no errors are detected, or errors are detected and corrected, the Coding Layer passes clean blocks of data to the Transfer Layer which is implemented in software. When a codeblock with an uncorrectable error is encountered, it is considered as the Tail Sequence, its contents are discarded and the Coding Layer returns to the Start Sequence search mode.

The Coding Layer also provides status information for the FAR, and it is possible to enable an optional de-randomiser according to [ECSS-E-ST-50-04C].

151.3.1 Synchronisation and selection of input channel

Synchronisation is performed by means of bit-by-bit search for a Start Sequence on the channel inputs. The detection of the Start Sequence is tolerant to a single bit error anywhere in the Start Sequence pattern. The Coding Layer searches both for the specified pattern as well as the inverted pattern. When an inverted Start Sequence pattern is detected, the subsequent bit-stream is inverted till the detection of the Tail Sequence.

The detection is accomplished by a simultaneous search on all active channels. The first input channel where the Start Sequence is found is selected for the CLTU decoding. The selection mechanism is restarted on any of the following events:

- The input channel active signal is de-asserted, or
- a Tail Sequence is detected, or

- a Codeblock rejection is detected, or
- an abandoned CLTU is detected, or the clock time-out expires.

As a protection mechanism in case of input failure, a clock time-out is provided for all selection modes. The clock time-out expires when no edge on the bit clock input of the selected input channel in decode mode has been detected for a specified period. When the clock time-out has expired, the input channel in question is ignored (i.e. considered inactive) until its active signal is de-asserted (configurable with gTimeoutMask=1).

151.3.2 Codeblock decoding

The received Codeblocks are decoded using the standard (63,56) modified BCH code. Any single bit error in a received Codeblock is corrected. A Codeblock is rejected as a Tail Sequence if more than one bit error is detected. Information regarding Count of Single Error Corrections and Count of Accept Codeblocks is provided to the FAR. Information regarding Selected Channel Input is provided via a register.

151.3.3 De-Randomiser

In order to maintain bit synchronisation with the received telecommand signal, the incoming signal must have a minimum bit transition density. If a sufficient bit transition density is not ensured for the channel by other methods, the randomiser is required. Its use is optional otherwise. The presence or absence of randomisation is fixed for a physical channel and is managed (i.e., its presence or absence is not signalled but must be known a priori by the spacecraft and ground system). A random sequence is exclusively OR-ed with the input data to increase the frequency of bit transitions. On the receiving end, the same random sequence is exclusively OR-ed with the decoded data, restoring the original data form. At the receiving end, the de-randomisation is applied to the successfully decoded data. The de-randomiser remains in the “all-ones” state until the Start Sequence has been detected. The pattern is exclusively OR-ed, bit by bit, to the successfully decoded data (after the Error Control Bits have been removed). The de-randomiser is reset to the “all-ones” state following a failure of the decoder to successfully decode a codeblock or other loss of input channel.

151.3.4 Non-Return-to-Zero – Mark

An optional Non-Return-to-Zero – Mark decoder can be enabled by means of a register.

151.3.5 Design specifics

The coding layer is supporting 1 to 8 channel inputs ([PSS-04-151] requires at least 4).

A codeblock is fixed to 56 information bits (as per CCSDS/ECSS).

The CCSDS/ECSS (1024 octets) or [PSS-04-151] (256 octets) standard maximum frame lengths are supported, being programmable via bit PSS in the GCR register. The former allows more than 37 codeblocks to be received.

The Frame Analysis Report (FAR) interface supports 8 bit CAC field, as well as the 6 bit CAC field specified in [PSS-04-151] When the PSS bit is cleared to '0', the two most significant bits of the CAC will spill over into the "LEGAL/ILLEGAL" FRAME QUALIFIER field in the FAR. These bits will however be all-zero when [PSS-04-151] compatible frame lengths are received or the PSS bit is set to '1'. The saturation is done at 6 bits when PSS bit is set to '1' and at 8 bits when PSS bit is cleared to '0'.

The Pseudo-Randomiser decoder is included (as per CCSDS/ECSS), its usage being input signal programmable.

The Physical Layer input can be NRZ-L or NRZ-M modulated, allowing for polarity ambiguity. NRZ-L/M selection is programmable. This is an extension to ECSS: Non-Return to Zero - Mark decoder added, with its internal state reset to zero when channel is deactivated.

Note: If input clock disappears, it will also affect the codeblock acquired immediately before the codeblock just being decoded (accepted by [PSS-04-151]).

In state S1, all active inputs are searched for start sequence, there is no priority search, only round robin search. The search for the start sequence is sequential over all inputs: maximum input frequency = system frequency /(gIn+2)

The [PSS-04-151] specified CASE-1 and CASE-2 actions are implemented according to aforementioned specification, not leading to aborted frames.

Extended E2 handling is implemented:

- E2b Channel Deactivation - selected input becomes inactive in S3
- E2c Channel Deactivation - too many codeblocks received in S3
- E2d Channel Deactivation - selected input is timed-out in S3
  (design choice being: S3 => S1, abandoned frame)

151.3.6 Direct Memory Access (DMA)

This interface provides Direct Memory Access (DMA) capability between the AMBA bus and the Coding Layer. The DMA operation is programmed via an AHB slave interface.

The DMA interface is an element in a communication concept that contains several levels of buffering. The first level is performed in the Coding Layer where a complete codeblock is received and kept until it can be corrected and sent to the next level of the decoding chain. This is done by inserting each correct information octet of the codeblock in an on-chip local First-In-First-Out (FIFO) memory which is used for providing improved burst capabilities. The data is then transferred from the FIFO to a system level ring buffer in the user memory (e.g. SRAM located in on-board processor board) which is accessed by means of DMA.

The following storage elements can thus be found in this design:

The shift and hold registers in the Coding Layer

The local FIFO (parallel; 32-bit; 4 words deep)

The system ring buffer (for example external SRAM; 32-bit; 1 to 256 kByte deep).

151.4 Transmission

The transmission of data from the Coding Layer to the system buffer is described hereafter.

The serial data is received and shifted in a shift register in the Coding Layer when the reception is enabled. After correction, the information content of the shift register is put into a hold register.

When space is available in the peripheral FIFO, the content of the hold register is transferred to the FIFO. The FIFO is of 32-bit width and the byte must thus be placed on the next free byte location in the word.

When the FIFO is filled for 50%, a request is done to transfer the available data towards the system level buffer.

If the system level ring buffer isn’t full, the data is transported from the FIFO, via the AHB master interface towards the main processor and stored in e.g. external SRAM. If no place is available in the system level ring buffer, the data is held in the FIFO.

When the GRTC keeps receiving data, the FIFO will fill up and when it reaches 100% of data, and the hold and shift registers are full, a receiver overrun interrupt will be generated (IRQ_RX_OVERRUN). All new incoming data is rejected until space is available in the peripheral FIFO.

When the receiving data stream is stopped (e.g. when a complete data block is received), and some bytes are still in the peripheral FIFO, then these bytes will be transmitted to the system level ring buf-

fer automatically. Received bytes in the shift and hold register are always directly transferred to the peripheral FIFO.

The FIFO is automatically emptied when a CLTU is either ready or has been abandoned. The reason for the latter can be codeblock error, time out etc. as described in CLTU decoding state diagram.

The operational state machine is shown in figure 366.

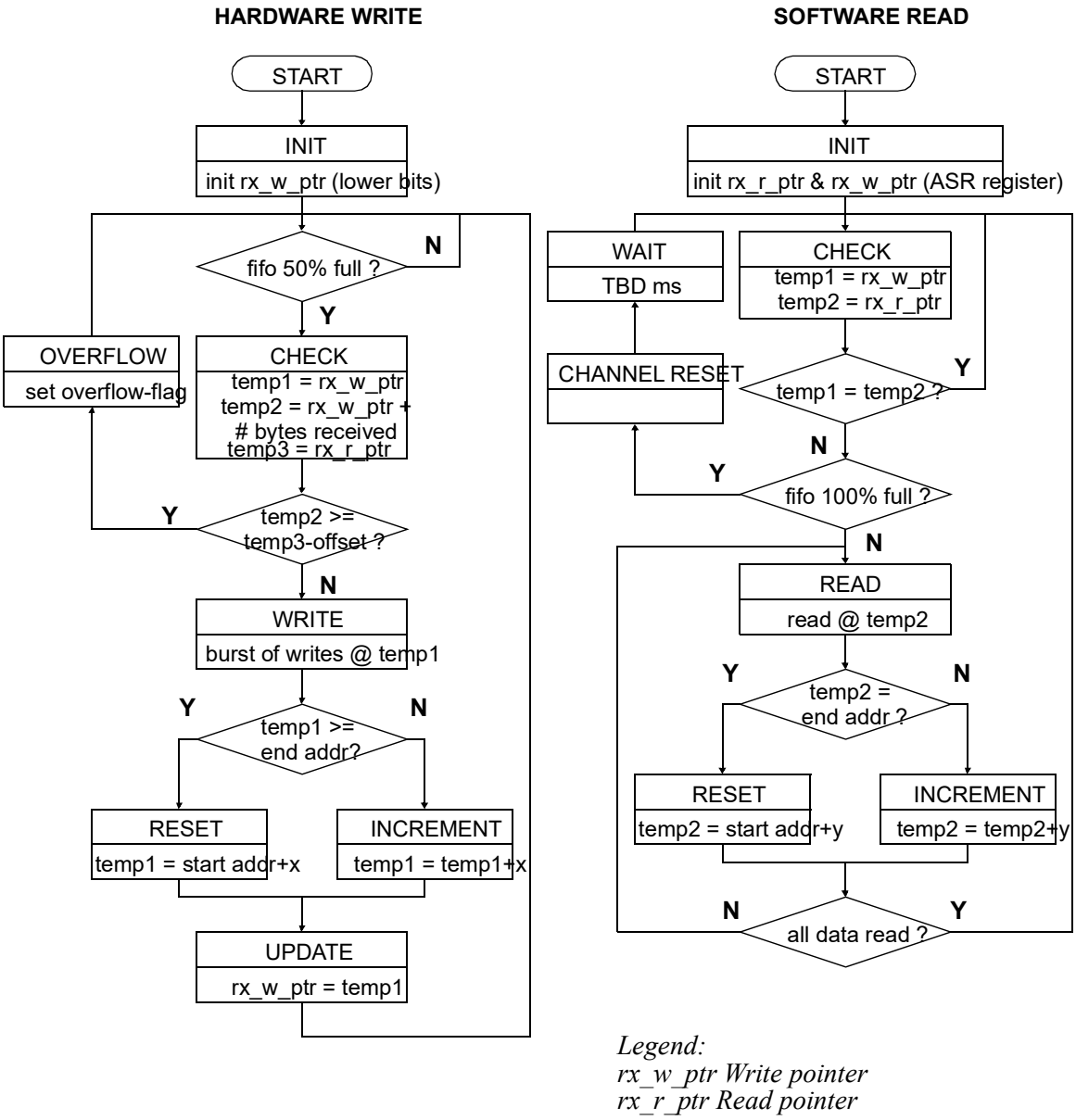


Figure 366. Direct Memory Access

151.4.1 Data formatting

When in the decode state, each candidate codeblock is decoded in single error correction mode as described hereafter.

151.4.2 CLTU Decoder State Diagram

Note that the diagram has been improved with explicit handling of different E2 events listed below.

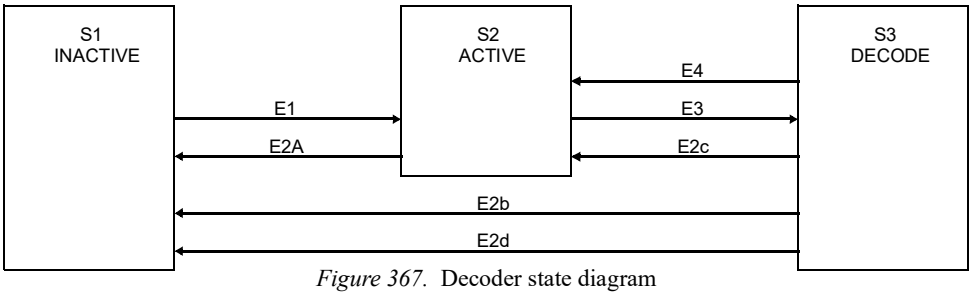


Figure 367. Decoder state diagram

State Definition:

- S1 Inactive
- S2 Search
- S3 Decode

Event Definition:

- E1 Channel Activation
- E2a Channel Deactivation - all inputs are inactive
- E2b Channel Deactivation - selected becomes inactive (CB=0 -> frame abandoned)
- E2c Channel Deactivation - too many codeblocks received (all -> frame abandoned)
- E2d Channel Deactivation - selected is timed-out (all -> frame abandoned)
- E3 Start Sequence Found
- E4 Codeblock Rejection (CB=0 -> frame abandoned)

151.4.3 Nominal

A: When the first “Candidate Codeblock” (i.e. “Candidate Codeblock” 0, which follows Event 3 (E3):START SEQUENCE FOUND) is found to be error free, or if it contained an error which has been corrected, its information octets are transferred to the remote ring buffer as shown in table 2545. At the same time, a “Start of Candidate Frame” flag is written to bit 0 or 16, indicating the beginning of a transfer of a block of octets that make up a “Candidate Frame”. There are two cases that are handled differently as described in the next sections.

*Table 2545.*Data format

	Bit[31……24]	**Bit[23……16]**	**Bit[15……8]**	**Bit[7……..……0]**
0x40000000	information octet0	0x01	information octet1	0x00
0x40000004	information octet2	0x00	information octet3	0x00
0x40000008	information octet4	0x00	end of frame	0x02
…	…		…	
0x400000xx	information octet6	0x01	information octet7	0x00
0x400000xx	information octet8	0x00	abandoned frame	0x03

Legend: Bit [17:16] or [1:0]:

“00” = continuing octet

“01” = Start of Candidate Frame

“10” = End of Candidate Frame

“11” = Candidate Frame Abandoned

151.4.4 CASE 1

When an Event 4 – (E4): CODEBLOCK REJECTION – occurs for any of the 37 possible “Candidate Codeblocks” that can follow Codeblock 0 (possibly the tail sequence), the decoder returns to the SEARCH state (S2), with the following actions:

- The codeblock is abandoned (erased)
- No information octets are transferred to the remote ring buffer
- An “End of Candidate Frame” flag is written, indicating the end of the transfer of a block of octets that make up a “Candidate Frame”.

151.4.5 CASE 2

When an Event 2 – (E2): CHANNEL DEACTIVATION – occurs which affects any of the 37 possible “Candidate Codeblocks” that can follow Codeblock 0, the decoder returns to the INACTIVE state (S1), with the following actions:

- The codeblock is abandoned (erased)
- No information octets are transferred to the remote ring buffer
- An “End of Candidate Frame” flag is written, indicating the end of the transfer of a block of octets that make up a “Candidate Frame”

151.4.6 Abandoned

- B: When an Event 4 (E4), or an Event 2 (E2), occurs which affects the first candidate codeblock 0, the CLTU shall be abandoned. No candidate frame octets have been transferred.
- C: If and when more than 37 Codeblocks have been accepted in one CLTU, the decoder returns to the SEARCH state (S2). The CLTU is effectively aborted and this is will be reported to the software by writing the “Candidate Frame Abandoned flag” to bit 1 or 17, indicating to the software to erase the “Candidate frame”.

151.5 Relationship between buffers and FIFOs

The conversion from the peripheral data width (8 bit for the coding layer receiver), to 32 bit system word width, is done in the peripheral FIFO.

All access towards the system ring buffer are 32-bit aligned. When the amount of received bytes is odd or not 32-bit aligned, the FIFO will keep track of this and automatically solve this problem. For the reception data path, the 32 bit aligned accesses could result in incomplete words being written to the ring buffer. This means that some bytes aren’t correct (because not yet received), but this is no problem due to the fact that the hardware write pointer (rx_w_ptr) always points to the last, correct, data byte.

The local FIFO ensures that DMA transfer on the AMBA AHB bus can be made by means of 2-word bursts. If the FIFO is not yet filled and no new data is being received this shall generate a combination of single accesses to the AMBA AHB bus if the last access was indicating an end of frame or an abandoned frame.

If the last single access is not 32-bit aligned, this shall generate a 32-bit access anyhow, but the receive-write-pointer shall only be incremented with the correct number of bytes. Also in case the previous access was not 32-bit aligned, then the start address to write to will also not be 32-bit aligned. Here the previous 32-bit access will be repeated including the bytes that were previously missing, in order to fill-up the 32-bit remote memory-controller without gaps between the bytes.

The receive-write-pointer shall be incremented according to the number of bytes being written to the remote memory controller.

151.5.1 Buffer full

The receiving buffer is full when the hardware has filled the complete buffer space while the software didn't read it out. Due to hardware implementation and safety, the buffer can't be filled completely without interaction of the software side. A space (offset) between the software read pointer (rx_r_ptr) and the hardware write pointer (rx_w_ptr) is used as safety buffer. When the write pointer (rx_w_ptr) would enter this region (due to a write request from the receiver), a buffer full signal is generated and all hardware writes to the buffer are suppressed. The offset is hard coded to 8 bytes.

Warning: If the software wants to receive a complete 1kbyte block (when RXLEN = 0), then it must read out at least 8 bytes of data from the buffer. In this case, the hardware can write the 1024 bytes without being stopped by the rx buffer full signal.

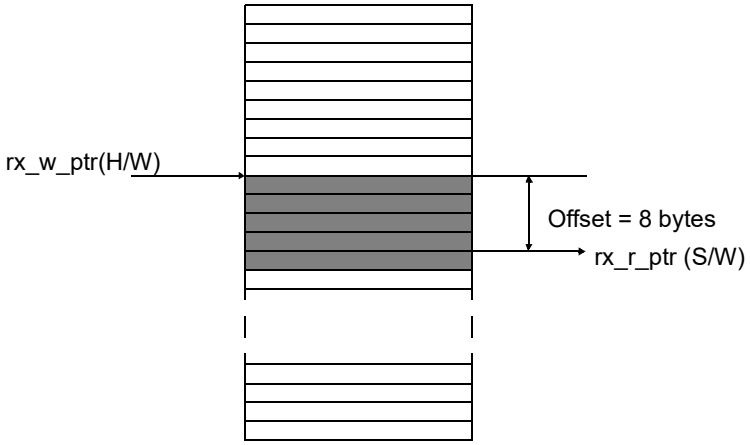


Figure 368. Buffer full situation

151.5.2 Buffer full interrupt

The buffer full interrupt is given when the difference between the hardware write pointer (rx_w_ptr) and the software read pointer (rx_r_ptr) is less than 1/8 of the buffer size. The way it works is the same as with the buffer full situation, only is the interrupt active when the security zone is entered. The buffer full interrupt is active for 1 system clock cycle. When the software reads out data from the buffer, the security zone shifts together with the read pointer (rx_r_ptr) pointer. Each time the hardware write pointer (rx_w_ptr) enters the security zone, a single interrupt is given.

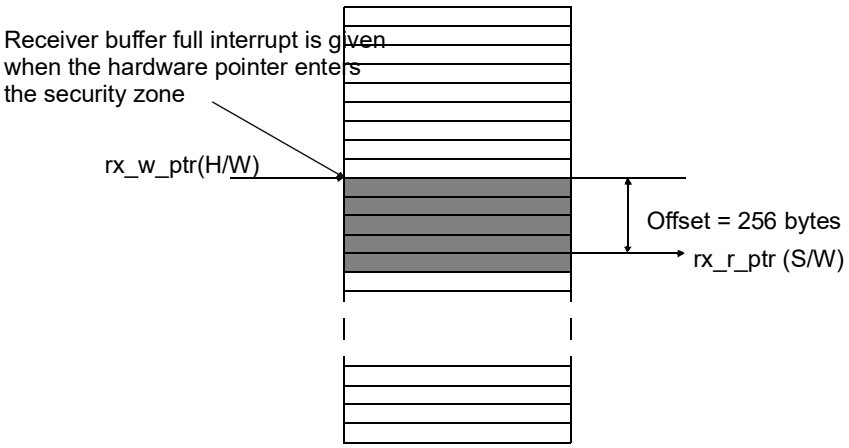


Figure 369. Buffer full interrupt (buffers size is 2kbyte in this example)

151.6 Command Link Control Word interface (CLCW)

The Command Link Control Word (CLCW) is inserted in the Telemetry Transfer Frame by the Telemetry Encoder (TM) when the Operation Control Field (OPCF) is present. The CLCW is created by the software part of the telecommand decoder. The telecommand decoder hardware provides two registers for this purpose which can be accessed via the AMBA AHB bus. Note that bit 16 (No RF Available) and 17 (No Bit Lock) of the CLCW are not possible to write by software. The information carried in these bits is based on discrete inputs.

Two PacketAsynchronous interfaces (PA) are used for the transmission of the CLCW from the telecommand decoder. The protocol is fixed to 115200 baud, 1 start bit, 8 data bits, 1 or 2 stop bits (configured by generics), with a BREAK command for message delimiting (sending 13 bits of logical zero).

The CLCWs are automatically transferred over the PA interface after reset, on each write access to the CLCW register and on each change of the bit 16 (No RF Available) and 17 (No Bit Lock).

Table 2546. CLCW transmission protocol

Byte Number	CLCWR register bits	CLCW contents						
First	[31:24]	Control Word Type	CLCW Version Number	Status Field	COP In Effect			
Second	[23:16]	Virtual Channel Id	Reserved Field					
Third	[15:8]	No RF Available	No Bit Lock	Lock Out	Wait	Retransmit	Farm B Counter	Report Type
Fourth	[7:0]	Report Value						
Fifth	N/A	[RS232 Break Command]						

151.7 Configuration Interface (AMBA AHB slave)

The AMBA AHB slave interface supports 32 bit wide data input and output. Since each access is a word access, the two least significant address bits are assumed always to be zero, address bits 23:0 are decoded. Note that address bits 31:24 are not decoded and should thus be handled by the AHB arbiter/decoder. The address input of the AHB slave interfaces is thus incompletely decoded. Misaligned addressing is not supported. For read accesses, unmapped bits are always driven to zero.

The AMBA AHB slave interface has been reduced in function to support only what is required for the TC. The following AMBA AHB features are constrained:

- Only supports HSIZE=WORD, HRESP_ERROR generated otherwise
- Only supports HMASTLOCK='0', HRESP_ERROR generated otherwise
- Only support HBURST=SINGLE or INCR, HRESP_ERROR generated otherwise
- No HPROT decoding
- No HSPLIT generated
- HRETRY is generated if a register is inaccessible due to an ongoing reset.
- HRESP_ERROR is generated for unmapped addresses, and for write accesses to register without any writeable bits
- Only big-endianness is supported.

During a channel reset the RRP and RWP registers are temporary unavailable. The duration of this reset-inactivity is 8 HCLK clock periods and the AHB-slave generates a HRETRY response during this period if an access is made to these registers.

If the channel reset is initiated by or during a burst-access the reset will execute correctly but a part of the burst could be answered with a HRETRY response. It is therefore not recommended to initiate write bursts to the register.

GRTC has interrupt outputs, that are asserted for at least two clock periods on the occurrence of one of the following events:

- 'CLTU stored' (generated when CLTU has been stored towards the AMBA bus, also issued for abandoned CLTUs)
- 'Receive buffer full' (generated when the buffer has less than 1/8 free) (note that this interrupt is issued on a static state of the buffer, and can thus be re-issued immediately after the corresponding register has been read out by software, it should be masked in the interrupt controller to avoid an immediate second interrupt).
- 'Receiver overrun' (generated when received data is dropped due to a reception overrun)
- 'CLTU ready' (note that this interrupt is also issued for abandoned CLTUs)
- FAR interrupt 'Status Survey Data'
- CLCW interrupt 'Bit Lock'
- CLCW interrupt 'RF Available'

151.7.1 Miscellaneous

The accuracy of the transmission or reception baud rate of the bit asynchronous serial interface is dependent on the selected system frequency and baud rate. The number of system clock periods used for sending or receiving a bit is directly proportional to the integer part of the division of the system frequency with the baud rate.

The BREAK command received on the bit asynchronous serial interface is a sequence of logical zeros that is at least one bit period longer than the normal byte frame, i.e. start bit, eight data bits, optional parity, one or two stop bits. When transmitted, it is always 13 bits.

151.8 Interrupts

The core generates the interrupts defined in table 2547.

*Table 2547.*Interrupts

Interrupt offset	Interrupt name	Description
1:st	RFA	RF Available changed
2:nd	BLO	Bit Lock changed
3:rd	FAR	FAR available
4:th	CR	CLTU ready/aborted
5:th	RBF	Output buffer full
6:th	OV	Input data overrun
7:th	CS	CLTU stored

151.9 Miscellaneous

151.9.1 Numbering and naming conventions

Convention according to the CCSDS recommendations, applying to time structures:

- The most significant bit of an array is located to the left, carrying index number zero.
- An octet comprises eight bits.

*Table 2548.*CCSDS n-bit field definition

CCSDS n-bit field		
most significant		least significant
0	1 to n-2	n-1

Convention according to AMBA specification, applying to the APB/AHB interfaces:

- Signal names are in upper case, except for the following:
- A lower case 'n' in the name indicates that the signal is active low.
- Constant names are in upper case.
- The least significant bit of an array is located to the right, carrying index number zero.
- Big-endian support.

*Table 2549.*AMBA n-bit field definition

AMBA n-bit field		
most significant		least significant
n-1	n-2 down to 1	0

General convention, applying to all other signals and interfaces:

- Signal names are in mixed case.
- An upper case '_N' suffix in the name indicates that the signal is active low.

151.9.2 Performance

The uplink bit rate is supported in the range of 1 kbits/s to 1 Mbits/s.

The bit rate is set to 115200 bit/s for the PacketAsynchronous (PA) interfaces.

151.10 Registers

The core is programmed through registers mapped into AHB I/O address space. Only 32-bit single-accesses to the registers are supported.

*Table 2550.*GRTC registers

AHB address offset	Register
0x00	Global Reset Register (GRR)
0x04	Global Control Register (GCR)
0x08	Physical Interface Mask Register (PMR)
0x0C	Spacecraft Identifier Register (SIR)
0x10	Frame Acceptance Report Register (FAR)
0x14	CLCW Register 1 (CLCWR1)
0x18	CLCW Register 2 (CLCWR2)
0x1C	Physical Interface Register (PHIR)
0x20	Control Register (COR)
0x24	Status Register (STR)
0x28	Address Space Register (ASR)
0x2C	Receive Read Pointer Register (RRP)
0x30	Receive Write Pointer Register (RWP)
0x60	Pending Interrupt Masked Status Register (PIMSR)
0x64	Pending Interrupt Masked Register (PIMR)
0x68	Pending Interrupt Status Register (PISR)
0x6C	Pending Interrupt Register (PIR)
0x70	Interrupt Mask Register (IMR)
0x74	Pending Interrupt Clear Register (PICR)

151.10.1Global Reset Register (GRR)

Table 2551. 0x00 - GRR - Global Reset Register

31 24	23 1	0
SEB	RESERVED	SRST
0	0	0
w	r	rw

31: 24 SEB (Security Byte):

Write: '0x55'= the write will have effect (the register will be updated).
Any other value= the write will have no effect on the register.

Read: All zero.

23: 1 RESERVED

Write: Don't care.

Read: All zero.

0 System reset (SRST): **[1]**

Write: '1'= initiate reset,'0'= do nothing

Read: '1'= unsuccessful reset, '0'= successful reset

151.10.2Global Control Register (GCR)

Table 2552. 0x04 - GCR - Global Control Register

31 24	23 13	12	11	10	9 0
SEB	RESERVED	PSS	NRZM	PSR	RESERVED
0	0	1	0	0	0
w	r	rw*	rw*	rw*	r

31: 24 SEB (Security Byte):

Write: '0x55'= the write will have effect (the register will be updated).
Any other value= the write will have no effect on the register.

Read: All zero.

23: 13 RESERVED

Write: Don't care.

Read: All zero.

12 PSS (ESA/PSS enable) [11]

Write/Read: '0'= disable, '1'= enable [Read-only when gHardware VHDL generic set.]

11 NRZM (Non-Return-to-Zero Mark Decoder enable)

Write/Read: '0'= disable, '1'= enable [Read-only when gHardware VHDL generic set.]

10 PSR (Pseudo-De-Randomiser enable)

Write/Read: '0'= disable, '1'= enable [Read-only when gHardware VHDL generic set.]

9: 0 RESERVED

Write: Don't care.

Read: All zero.

Power-up default: 0x00001000, The default value depends on the TCC_PSS, TCC_Mark, TCC_Pseudo inputs.

151.10.3Physical Interface Mask Register (PMR)

Table 2553. 0x08 -PMR - Physical Interface Mask Register

31 8	7 0
RESERVED	MASK
0	0
r	rw

31: 8 RESERVED
- Write: Don’t care.
- Read: All zero.

7: 0 MASK
- Write: Mask TC input when set, bit 0 corresponds to TC input 0
- Read: Current mask

151.10.4Spacecraft Identifier Register (STR)

Table 2554. 0x0C - SIR - Spacecraft Identifier Register [7]

31 10	9 0
RESERVED	SCID
0	*
r	r

31: 10 RESERVED
- Write: Don’t care.
- Read: All zero.

9: 0 SCID (Spacecraft Identifier)
- Write: Don’t care.
- Read: Bit[9]=MSB, Bit[0]=LSB

Power-up default: Depends on SCID input configuration.

151.10.5Frame Acceptance Report Register (FAR)

Table 2555. 0x10 - FAR - Frame Acceptance Report Register [7]

31	30 25	24 19	18 16	15 14	13 11	10 0
SSD	RESERVED	CAC	CSEC	RESERVED	SCI	RESERVED
0	0	0	0	0	0b111	0
r	r	r	r	r	r	r

31 SSD (Status of Survey Data) (see [PSS-04-151])

Write: Don't care.

Read: Automatically cleared to 0 when any other field is updated by the coding layer. Automatically set to 1 upon a read.

30: 25 RESERVED

Write: Don't care.

Read: All zero.

24: 19 CAC (Count of Accept Codeblocks) (see [PSS-04-151])

Write: Don't care.

Read: Information obtained from coding layer. [2]

18: 16 CSEC (Count of Single Error Corrections) (see [PSS-04-151])

Write: Don't care.

Read: Information obtained from coding layer.

15: 14 RESERVED

Write: Don't care.

Read: All zero.

13: 11 SCI (Selected Channel Input) (see [PSS-04-151])

Write: Don't care.

Read: Information obtained from coding layer.

10: 0 RESERVED

Write: Don't care.

Read: All zero.

151.10.6CLCW Register (CLCWRx)

*Table 2556.*0x14 - CLCWRx - CLCW Register (see [PSS-04-107])

31 30 29	28 26	25 24	23 18	17 16	15	14	13	12	11	10 9	8	7 0
CWTY	VNUM	STAF	CIE	VCI	RESERVED	NRFA	NBLO	LOUT	WAIT	RTMI	FBCO	RTYPE	RVAL
0	0	0	0	0	0	0	0	0	0	0	0	0	0
rw	rw	rw	rw	rw	rw	r	r	rw	rw	rw	rw	rw	rw

31 CWTY (Control Word Type)

30: 29 VNUM (CLCW Version Number)

28: 26 STAF (Status Fields)

25: 24 CIE (COP In Effect)

23: 18 VCI (Virtual Channel Identifier)

17: 16 Reserved (PSS/ECSS requires “00”)

15 NRFA (No RF Available)

Write: Don’t care.

Read: Based on discrete inputs.

14 NBLO (No Bit Lock)

Write: Don’t care.

Read: Based on discrete inputs.

13 LOUT (Lock Out)

12 WAIT (Wait)

11 RTMI (Retransmit)

10: 9 FBCO (FARM-B Counter)

8 RTYPE (Report Type)

7: 0 RVAL (Report Value)

151.10.7Physical Interface Register (PHIR)

*Table 2557.*0x1C - PHIR - Physical Interface Register [7]

31 16	15 8	7 0
RESERVED	RFA	BLO
0	*	*
r	r	r

31: 16 RESERVED

Write: Don’t care.

Read: All zero.

15: 8 RFA (RF Available) [3]

Only implemented inputs are taken into account. All other bits are zero.

Write: Don’t care.

Read: Bit[8] = input 0, Bit[15] = input 7

7: 0 BLO (Bit Lock) [3]

Only implemented inputs are taken into account. All other bits are zero.

Write: Don’t care.

Read: Bit[0] = input 0, Bit[7] = input 7

151.10.8Control Register (COR)

*Table 2558.*0x20 - COR - Control Register

31 24	23 10	9	8 1	0
SEB	RESERVED	CRST	RESERVED	RE
0	0	0	0	*
w	r	rw	r	rw

31: 24 SEB (Security Byte):

Write: '0x55'= the write will have effect (the register will be updated). Any other value= the write will have no effect on the register.

Read: All zero.

23: 10 RESERVED

Write: Don't care.

Read: All zero.

9 CRST (Channel reset) [4]

Write: '1'= initiate channel reset,'0'= do nothing

Read: '1'= unsuccessful reset, '0'= successful reset

8: 1 RESERVED

Write: Don't care.

Read: All zero.

0 RE (Receiver Enable) [Always enabled when gHardware VHDL generic set.]

The TCActive input of the receiver are masked when the RE bit is disabled.

Read/Write: '0'= disabled, '1'= enabled

Power-up default: 0x00000000 [0x00000001 when gHardware VHDL generic set.]

151.10.9Status Register (STR)

*Table 2559.*0x24 - STR - Status Register [7]

31 11	10	9 8	7	6 5	4	3 1	0
RESERVED	RBF	RESERVED	RFF	RESERVED	OV	RESERVED	CR
0	0	0	0	0	0	0	0
r	r	r	r	r	r	r	r

31: 11 RESERVED

Write: Don’t care.

Read: All zero.

10 RBF (RX BUFFER Full)

Write: Don’t care.

Read: ‘0’ = Buffer not full,

‘1’= Buffer full (this bit is set if the buffer has less then 1/8 of free space)

9: 8 RESERVED

Write: Don’t care.

Read: All zero.

7 RFF (RX FIFO Full)

Write: Don’t care.

Read: ‘0’ = FIFO not full, ‘1’ = FIFO full

6: 5 RESERVED

Write: Don’t care.

Read: All zero.

4 OV (Overrun) [5]

Write: Don’t care.

Read: ‘0’= nominal, ‘1’= data lost

3: 1 RESERVED

Write: Don’t care.

Read: All zero.

0 CR (CLTU Ready) [5]

There is a worst case delay from the CR bit being asserted, until the data has actually been transferred from the receiver FIFO to the ring buffer. This depends on the PCI load etc.

Write: Don’t care.

Read: ‘1’= new CLTU in ring buffer. ‘0’= no new CLTU in ring buffer.

151.10.10 Address Space Register (ASR)

*Table 2560.*0x28 - ASR - Address Space Register [8]

31 10	9 8	7 0
BUFST	RESERVED	RXLEN
0	0	0
rw	r	rw

31: 10 BUFST (Buffer Start Address)

22-bit address pointer

This pointer contains the start address of the allocated buffer space for this channel.

Register has to be initialized by software before DMA capability can be enabled.

9: 8 RESERVED

Write: Don't care.

Read: All zero.

7: 0 RXLEN *(RX buffer length)*

Number of 1kB-blocks reserved for the RX buffer.
(Min. 1kByte = 0x00, Max. 256kByte = 0xFF)

151.10.11 Receive Read Pointer Register (RRP)

*Table 2561.*0x2C - RRP - Receive Read Pointer Register [6] [9][10]

31 24	23 0
RxRd Ptr Upper	RxRd Ptr Lower
0	0
r	rw

31: 24 10-bit upper address pointer

Write: Don't care.

Read: This pointer = ASR[31..24].

23: 0 24-bit lower address pointer.

This pointer contains the current RX read address. This register is to be incremented with the actual amount of bytes read.

151.10.12 Receive Write Pointer Register (RWP)

*Table 2562.*0x30 - RWP - Receive Write Pointer Register [6] [9]

31 24	23 0
RxWr Ptr Upper	RxWr Ptr Lower
0	0
r	r

31: 24 10-bit upper address pointer

Write: Don't care.

Read: This pointer = ASR[31..24].

23: 0 24-bit lower address pointer.

This pointer contains the current RX write address. This register is incremented with the actual amount of bytes written.

Legend:

[1] The global system reset caused by the SRST-bit in the GRR-register results in the following actions:

- Initiated by writing a '1", gives '0' on read-back when the reset was successful.

- No need to write a '0' to remove the reset.

- Unconditionally, means no need to check/disable something in order for this reset-function to correctly execute.
- Could of course lead to data-corruption coming/going from/to the reset core.
- Resets the complete core (all logic, buffers & register values)
- Behaviour is similar to a power-up.

[2] The FAR register supports the CCSDS/ECSS standard frame lengths (1024 octets), requiring an 8 bit CAC field instead of the 6 bits specified for PSS. The two most significant bits of the CAC will thus spill over into the "LEGAL/ILLEGAL" FRAME QUALIFIER field, Bit [26:25]. This is only the case when the PSS bit is set to '0'.

[3] The number of channels are controlled with the gRFAvailable and gBitLock VHDL generics, respectively.

[4] The channel reset caused by the CRST-bit in the COR-register results in the following actions:
- Initiated by writing a '1", gives '0' on read-back when the reset was successful.
- No need to write a '0' to remove the reset.
- All other bit's in the COR are neglected (not looked at) when the CRST-bit is set during a write, meaning that the value of these bits has no impact on the register-value after the reset.
- Unconditionally, means no need to check/disable something in order for this reset-function to correctly execute.
- Could of course lead to data-corruption coming/going from/to the reset channel.
- Resets the complete channel (all logic, buffers & register values)
- Except the ASR-register of that channel which remains it's value.
- All read- and write-pointers are automatically re-initialized and point to the start of the ASR-address.
- All registers of the channel (except the ones described above) get their power-up value.
- This reset shall not cause any spurious interrupts.

[5] These bits are sticky bits which means that they remain present until the register is read and that they are cleared automatically by reading the register.

[6] The value of the pointers depends on the content of the corresponding Address Space Register (ASR). During a system reset, a channel reset or a change of the ASR register, the pointers are recalculated based on the values in the ASR register.
The software has to take care (when programming the ASR register) that the pointers never have to cross a 16MByte boundary (because this would cause an overflow of the 24-bit pointers).
It is not possible to write an out of range value to the RRP register. Such access will be ignored with an HERROR.

[7] An AMBA AHB ERROR response is generated if a write access is attempted to a register without any writeable bits.

[8] The channel reset caused by a write to the ASR-register results in the following actions:
- Initiated by writing an updated value into the ASR-register.
- Unconditionally, means no need to check/disable something in order for this reset-function to correctly execute.
- Could of course lead to data-corruption coming/going from/to the reset channel.
- Resets the complete channel (all logic & buffers) but not all register values, only the following:
- COR-register, TE & RE bits get their power-up value, other bits remain their value.
- STR-register, all bits get their power-up value
- Other registers remain their value
- Updates the ASR-register of that channel with the written value
- All read- and write-pointers are automatically re-initialized and point to the start of the ASR-address.
- This reset shall not cause any spurious interrupts

[9] During a channel reset the register is temporarily unavailable and HRETRY response is generated if accessed.

[10] It is not possible to write an out of range value to the RRP register. Such access will be ignored without an error.

[11] The PSS bit usage is only supported if the gPSS generic is set on the TCC module.

151.10.13Interrupt registers

The interrupt registers give complete freedom to the software, by providing means to mask interrupts, clear interrupts, force interrupts and read interrupt status.

When an interrupt occurs the corresponding bit in the Pending Interrupt Register is set. The normal sequence to initialize and handle a module interrupt is:

- Set up the software interrupt-handler to accept an interrupt from the module.
- Read the Pending Interrupt Register to clear any spurious interrupts.
- Initialize the Interrupt Mask Register, unmasking each bit that should generate the module interrupt.
- When an interrupt occurs, read the Pending Interrupt Status Register in the software interrupt-handler to determine the causes of the interrupt.
- Handle the interrupt, taking into account all causes of the interrupt.
- Clear the handled interrupt using Pending Interrupt Clear Register.

Masking interrupts: After reset, all interrupt bits are masked, since the Interrupt Mask Register is zero. To enable generation of a module interrupt for an interrupt bit, set the corresponding bit in the Interrupt Mask Register.

Clearing interrupts: All bits of the Pending Interrupt Register are cleared when it is read or when the Pending Interrupt Masked Register is read. Reading the Pending Interrupt Masked Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register. Selected bits can be cleared by writing ones to the bits that shall be cleared to the Pending Interrupt Clear Register.

Forcing interrupts: When the Pending Interrupt Register is written, the resulting value is the original contents of the register logically OR-ed with the write data. This means that writing the register can force (set) an interrupt bit, but never clear it.

Reading interrupt status: Reading the Pending Interrupt Status Register yields the same data as a read of the Pending Interrupt Register, but without clearing the contents.

Reading interrupt status of unmasked bits: Reading the Pending Interrupt Masked Status Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register, but without clearing the contents.

The interrupt registers comprise the following:

*Table 2563.*Interrupt registers

Description	Name	Mode
Pending Interrupt Masked Status Register	PIMSR	r
Pending Interrupt Masked Register	PIMR	r
Pending Interrupt Status Register	PISR	r
Pending Interrupt Register	PIR	rw
Interrupt Mask Register	IMR	rw
Pending Interrupt Clear Register	PICR	w

*Table 2564.*Interrupt registers template

31 7	6	5	4	3	2	1	0
-	CS	OV	RBF	CR	FAR	BLO	RFA
	0	0	0	0	0	0	0
	*	*	*	*	‘	*	‘

6: CS CLTU stored*
5: OV Input data overrun*
4: RBF Output buffer full*
3: CR CLTU ready/aborted*

2: FAR FAR available*
1: BLO Bit Lock changed*
0: RFA RF Available Changed*

*See table 2563.

151.11 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x031. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

151.12 Implementation

151.12.1Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

151.13 Configuration options

Table 2565 shows the configuration options of the core (VHDL generics).

*Table 2565.*Configuration options

Generic	Function	Description	Allowed range	Default
GRLIB AMBA plug&play settings				
hindex	AHB slave index		Integer	0
hirq	AHB slave interrupt		Integer	0
singleirq	Single interrupt	Enable interrupt registers	Integer	0
inputmask	Maskable input		Integer	0
ioaddr	IO area address		0 - 16#FFF#	0
iomask	IO area mask		0 - 16#FFF#	16#FFF#
syncrst	synchronous reset		0 - 1	0
gHardware	Hardware commands	Coding layer fixed configuration & enable	0 - 1	0
Features settings				
gIn	Number of channels	Number of TC channels	1 - 8	3
gPSS	PSS support enable	Enables PSS support	0 - 1	1
gTimeoutMask	Time out mask	Enables masking of input on time out	0 - 1	0
gTimeout	Time out period	2^n clock periods		24
gRFAvailable	Number of RF inputs	Minimum 1	1 - 8	3
gBitLock	Number of TC inputs	Minimum 1	1 - 8	3
gDepth	FIFO depth	words (2^x)		4
Asynchronous bit serial interface settings (CLCW interface)				
gSystemClock	System frequency	[Hz]	Integer	33333333
gBaud	Baud rate	[Baud]	Integer	115200
gOddParity	Odd parity	Odd parity generated, but not checked	0 - 1	0
gTwoStopBits	Number of stop bits	0=one stop bit, 1=two stop bits	0 - 1	0

151.14 Signal descriptions

Table 2566 shows the interface signals of the core (VHDL ports).

*Table 2566.*Signal descriptions

Signal name	Field	Type	Function	Description	Active
HRESETn	N/A	Input	Reset		Low
HCLK	N/A	Input	Clock		-
TCIN	TCC_SCID	Input	Spacecraft Identity	0 is MSB, 9 is LSB	-
	TCACTIVE		Active	Indicate that sub-carrier lock is achieved (or bit lock). Enable for the clock and data.	-
	TCCLK		Bit clock	Serve as the serial data input bit clocks.	-
	TCDATA		Data	Serve as the serial data input. Data are sampled on the TCCLK clock edges when the corresponding TCActive input is asserted.	-
	TCC_HIGH		Active high setting	1=active high delimiter, 0=active low	-
	TCC_RISE		Rising clock edge	1=rising, 0=falling	-
	PSEUDO		Pseudo-Derando-miser	1=enabled, 0=disabled	-
	MARK		NRZ-M decoder	1=NRZ-M, 0=NRZ-L	-
	PSS		PSS/ECSS mode	1=ESA PSS, 256 octet, fill bit aug-ment 0=ECSS, 1024 octet, no fill bit augment	-
	RFAVAILPOS		RF Available polarity	Active high=1 / low=0, used for CLCW	-
	BITLOCKPOS		Bit Lock polarity	Active high=1 / low=0, used for CLCW	-
	CLCWRFAVAIL-ABLE		RF Available	Used for CLCW	-
	CLCWBITLOCK		Bit Lock	Used for CLCW	-
TCOUT	CLCW1DATA	Output	CLCW Data	Bit serial asynchronous data for CLCW 1 interface.	-
	CLCW2DATA		CLCW Data	Bit serial asynchronous data for CLCW 2 interface.	-
AHBSIN	*	Input	AMB slave input signals		-
AHBSOUT	*	Output	AHB slave output signals		-
	HIRQ(hirq+6)		Interrupts (If *singleirq*=1 only one common interrupt will be generate using *hirq*.)	CLTU stored	Location on HIRQ bus depends on *hirq* generic. If *hirq*=0, no interrupt will be generated.
	HIRQ(hirq+5)			Input data overrun	
	HIRQ(hirq+4)			Output buffer full	
	HIRQ(hirq+3)			CLTU ready/aborted	
	HIRQ(hirq+2)			FAR available	
	HIRQ(hirq+1)			Bit Lock changed	
	HIRQ(hirq+0)			RF Available changed	
AHBMIN	*	Input	AMB master input signals		-
AHBMOUT	*	Output	AHB master output signals		-

* see GRLIB IP Library User's Manual

151.15 Signal definitions and reset values

The signals and their reset values are described in table 2567.

*Table 2567.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
tcc_scid	Input, static	Spacecraft Identity	-	-
tcc_high	Input, static	Active high setting	-	-
tcc_rise	Input, static	Rising clock edge	-	-
pseudo	Input, static	Pseudo-Derandomiser	-	-
mark	Input, static	NRZ-M decoder	-	-
pss	Input, static	PSS/ECSS mode	-	-
rfavailpos	Input, static	RF Available polarity	-	-
bitlockpos	Input, static	Bit Lock polarity	-	-
tcactive	Input, async	Active	-	-
tcclk	Input, async	Bit clock	-	-
tcdata	Input, async	Data	-	-
rfavailable	Input, async	RF Available for CLCW	-	-
bitlock	Input, async	Bit Lock for CLCW	-	-
clcw1data	Output	CLCW output data 1	-	Logical 1
clcw2data	Output	CLCW output data 2	-	Logical 1

151.16 Timing

The timing waveforms and timing parameters are shown in figure 370 and are defined in table 2568.

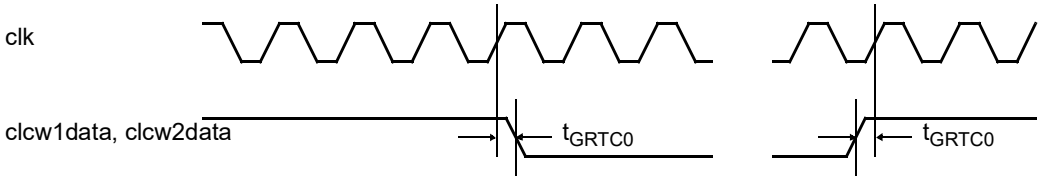


Figure 370. Timing waveforms

*Table 2568.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{GRTC0}	clock to output delay	rising *clk* edge	TBD	TBD	ns

Note: The inputs are re-synchronized internally. The signals do not have to meet any setup or hold requirements. Static signals should not change between resets.

151.17 Library dependencies

Table 2569 shows libraries used when instantiating the core (VHDL libraries).

*Table 2569.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_Types	Signals, component	Signals and component declaration

151.18 Instantiation

This example shows how the core can be instantiated.

```
library  IEEE;
   use      IEEE.Std_Logic_1164.all;
   library  GRLIB;
   use      GRLIB.AMBA.all;
   library  TMTC;
   use      TMTC.TMTC_Types.all;

...

   component GRTC is
   generic(
      hmstndx:       in    Integer                 := 0;
      hslvndx:       in    Integer                 := 0;
      ioaddr:        in    Integer                 := 0;
      iomask:        in    Integer                 := 16#fff#;
      hirq:          in    Integer                 := 0;
      syncrst:       in    Integer                 := 0;  -- synchronous reset
      gIn:           in    Integer range 1 to  8 := 3;    -- number of inputs
      gPSS:          in    Integer range 0 to  1 := 0;    -- enable PSS support
      gTimeoutMask:  in    Integer range 0 to  1 := 0;    -- timeout mask
      gTimeout:      in    Integer                 := 24;  -- timeout 2^n
      gSystemClock:  in    Natural := 333333333;          -- Hz
      gBaud:         in    Natural := 115200;             -- Baud rate setting
      gOddParity:    in    Natural := 0;                  -- Odd parity
      gTwoStopBits:  in    Natural := 0;                  -- Stop bit selection
      gRFAvailable:  in    Natural range 1 to  8 := 3;    -- No. of RF inputs
      gBitLock:      in    Natural range 1 to  8 := 3;    -- No. of BL inputs
      gDepth:        in    Natural := 4;                  -- words (2^x)
      gHardware:     in    Natural range 0 to 1 := 0);    -- enable hardware cmd
   port(
      -- AMBA AHB system signals
      HCLK:          in    Std_ULogic;                    -- System clock
      HRESETn:       in    Std_ULogic;                    -- Synchronised reset

      -- AMBA AHB slave Interface
      AHBSIn:        in    AHB_Slv_In_Type;               -- AHB slave input
      AHBSOut:       out   AHB_Slv_Out_Type;              -- AHB slave output

      -- AMBA AHB master Interface
      AHBMIn:        in    AHB_Mst_In_Type;               -- AHB master input
      AHBMOut:       out   AHB_Mst_Out_Type;              -- AHB master output

      -- Telecommand interfaces
      TCIn:          in    GRTC_In_Type;
      TCOut:         out   GRTC_Out_Type);
   end component GRTC;
```

152 TCAU - Telecommand Decoder Authentication Unit

152.1 Overview

The TCAU is compatible with the Telecommand Decoder Specification defined by [PSS-04-151] and implements authentication in accordance with [CCSDS-350.0] as defined by Option B, authentication of segment. The TCAU is placed between the Transfer Sublayer and the Segmentation Sublayer.

The TCAU authentication method is a plain text with appended signature. It is based on a one-way function called “hard knapsack”, which relies on a 40-bit digital signature generated by applying a secret key on the segment. The same algorithm is used for generating the signature in the transmitting end as in the receiving end.

152.2 Operation

The TCAU receives segments from the transfer sublayer octet by octet, starting with the segment header. By comparing the 5 LSBs of the segment header, the MAP address, with the configuration signal *aumap*, the TCAU determines if the segment is to be authenticated. If the *auenable* input signal is high, all segments where the MAP address is less than or equal to *aumap* will be sent to the Authentication Processor for authentication. Segments where the MAP address is higher than *aumap*, will be sent to the segmentation sublayer without processing. A special case exists for segments where the entire MAP ID equals 63. These are dedicated for the Control Command Processor of the TCAU and are always authenticated, see further section 152.9. The control commands are used to change the internal configuration of the TCAU and are always unsegmented, i.e. the segmentation flags are set to 11b.

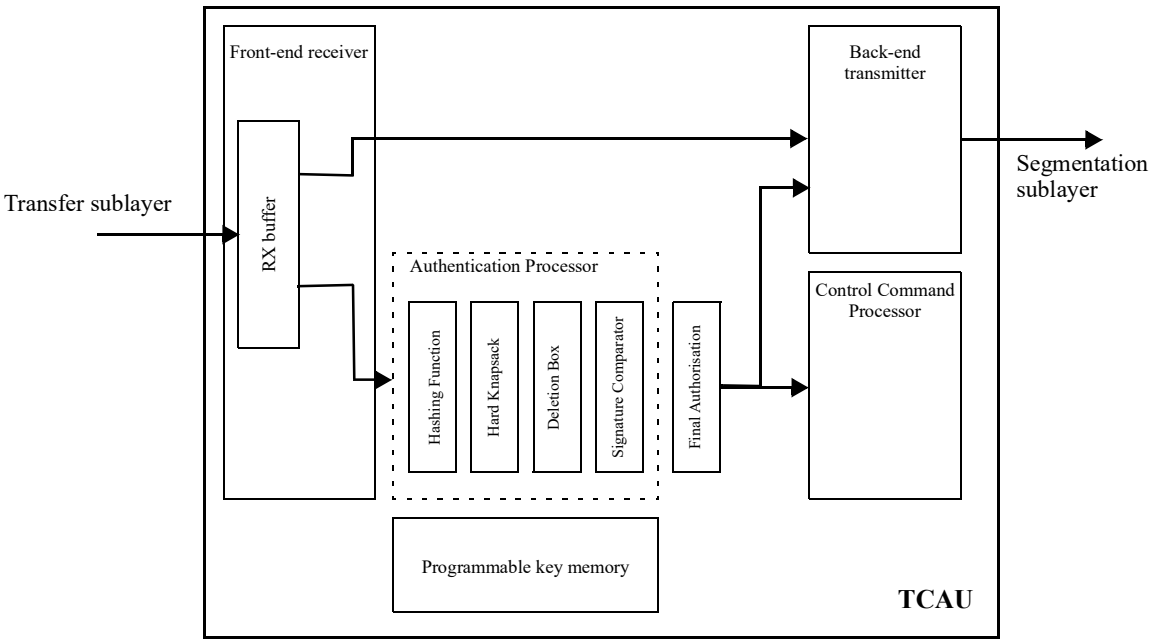


Figure 371. Block diagram

The TCAU contains the following major functions:

- Front-end receiver
- Authentication Processor
- Final Authorisation
- Control Command Processor

- Programmable key
- Back-end transmitter

152.3 References

[PSS-04-151] Telecommand Decoder Specification, ESA PSS-04-151, issue 1

[CCSDS-350.0] The Application of CCSDS Protocols to Secure Systems, CCSDS 350.0-G-2, issue 2

152.4 Data structures

Table 2570 shows the structure of a telecommand segment with the optional authentication tail used by the TCAU. The authentication tail is mandatory for all segments that are to be authenticated.

Table 2570. Telecommand segment with authentication tail

<table>
<tr><td colspan="7">Telecommand segment with optional tail</td></tr>
<tr><td colspan="3">Segment header</td><td rowspan="2">Segment data field</td><td colspan="3">Authentication tail (optional)</td></tr>
<tr><td rowspan="2">Sequence flags</td><td colspan="2">MAP identifier</td><td colspan="2">Logical authentication channel (LAC)</td><td rowspan="2">Signature</td></tr>
<tr><td>Control flag</td><td>MAP address</td><td>LAC ID</td><td>LAC count</td></tr>
<tr><td>2 bits</td><td>1 bit</td><td>5 bits</td><td>0 to 239 octets or</td><td>2 bits</td><td>30 bits</td><td>40 bits</td></tr>
<tr><td colspan="3">1 octet</td><td>0 to 248 octets</td><td colspan="3">0 or 9 octets</td></tr>
<tr><td colspan="7">1 to 249 octets</td></tr>
</table>

152.5 Front-end receiver

The front-end receiver handles reception of octets from the Transfer sublayer using a handshaking protocol. Octets are placed in a 10 octet buffer, in order to accommodate the entire authentication tail. The front-end receiver counts all octets received to determine the segment length. The octets are passed on to the Authentication Processor and the Back-end transmitter.

152.6 Authentication Processor

The Authentication Processor receives the entire segment from the Front-end receiver and calculates the 40-bit digital signature using one of two on-board keys: the fixed key or the programmable key. The 40-bit digital signature is compared to the signature in the authentication tail appended to the segment. The Authentication Processor comprises the following functions:

- Hashing Function
- Hard Knapsack
- Deletion Box
- Signature Comparator

152.6.1 Hashing Function

The Hashing Function generates i 60-bit pre-signature by shifting the extended message, x, into a linear feedback shift register (LFSR). The extended message comprises the segment header, segment data, received LAC from the authentication tail and finally filler bits, comprising 24 zeros, to ensure a minimum length of the extended message. The coefficients of the LFSR are part of the keys.

When a new segment is received, the LFSR is initialized to 1000...000, i.e. bit 0 is set to 1 and all other bits are set to zero. When the entire extended message has been shifted into the LFSR, it will contain the pre-signature, P.

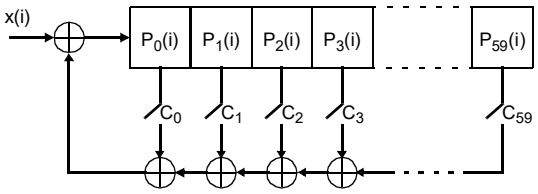


Figure 372. Hashing Function

152.6.2 Hard Knapsack

The purpose of the Hard Knapsack function is to make the overall system non-linear and to serve as a true one-way function, so that it is not possible to calculate the pre-signature from the signature. The Hard Knapsack consists of 60 weights, which are part of the two on-board keys, with each weight being 48 bits long. It is defined by the following transformation:

$$S' = \left(\sum_{j=0}^{59} P_j W_j \right) \ mod \ 2^{48}$$

Where P is the pre-signature calculated by the Hashing Function, i.e. each bit of the pre-signature selects the corresponding weight for the Hard Knapsack function.

The result of the Hard Knapsack function is the 48-bit knapsack sum, S'.

152.6.3 Deletion Box

The Deletion Box removes the 8 least significant bits, i.e. bits 40 to 47, from the knapsack sum, S', to form the authentication signature, S. It has been shown that the least significant bits are too weak from a cryptanalysis standpoint.

152.6.4 Signature Comparator

The Signature Comparator compares the received signature, s, with the calculated signature, S, and presents the result to the Final Authorisation function of the Supervisor.

152.7 Final Authorisation

When the received signature (s) and the calculated signature (S) are found to be identical, the Final Authorisation function checks that the received LAC count is identical to the LAC count of the counter indicated by the received LAC ID. If these match, the used LAC counter is incremented by 1. Note that the LAC counter is incremented even if the segment contains a control command which is found to be incorrect.

If the telecommand segment was transferred on MAP 63, it is transferred to the Control Command Processor.

The Final Authorisation also generates a report on the output *auana*, outputs the MAP on *lastmap* and pulses *auanavalid* for one clock cycle.

152.8 Control Command Processor

The Control Command Processor executes the control commands used for changing the internal configuration of the TCAU. Any commands not conforming to the specified formats are reported as non-executable.

152.9 Control Commands

Before being executed, the following checks are performed on the control commands:

- The sequence flags of the segment header are verified to be 11b (unsegmented)
- The segment is verified to contain a valid control command
- The length of the segment is verified to comply with the expected length of the command
- The LAC ID for a “Set new LAC count value” command is verified to be 00b, 01b or 10b
- The start address of a “Change programmable key block B” command is verified to be less than 112

If a segment is transferred to the Control Command Processor and fails any of these checks, it is discarded and reported in *auana*.

The control commands are listed in table 2571. Table 2572, 2573 and 2574 shows the layout of the three groups of control commands.

*Table 2571.*Authentication Unit Control Commands

Group	Control Command Identifier	Command name
1	0000 0000	Dummy segment
	0000 0101	Select fixed key
	0000 0110	Select programmable key
	0000 0111	Load fixed key in programmable key memory
2	0000 1001	Set new LAC count value
3	0000 1010	Change programmable key block A
	0000 1011	Change programmable key block B

*Table 2572.*Group 1 control commands

Segment header	Control command identifier	Authentication tail
0xFF	00000***	LAC + Signature
1 octet	1 octet	9 octets

*Table 2573.*Group 2 control commands

Segment header	Control command identifier	LAC value to be set		Authentication tail
0xFF	00001001	LAC ID	LAC count	LAC + Signature
1 octet	1 octet	2 bits	30 bits	9 octets

*Table 2574.*Group 3 control commands

Segment header	Control command identifier	Start address	Key-specific pattern	Authentication tail
0xFF	0000101*			LAC + Signature
1 octet	1 octet	1 octet	7 octets	9 octets

152.9.1 Dummy segment

This control command is provided for testing purposes. It has no side effects when executed, but since it is authenticated, the LAC counters will be updated and a new report will be generated.

This command is authenticated with the currently selected key.

152.9.2 Select fixed key

If authentication is successful, the result of the command is that the fixed key will be used for subsequent segments.

This control command is always authenticated using the fixed key.

152.9.3 Select programmable key

If authentication is successful, the result of the command is that the programmable key will be used for subsequent segments.

This control command is always authenticated using the programmable key.

152.9.4 Load fixed key in programmable key memory

This control command will copy the weights and the coefficients of the fixed key to the programmable key.

This command is authenticated with the currently selected key.

152.9.5 Set new LAC count value

This command sets the value of either of the three LAC counters, as determined by the LAC ID identifier in the segment data field. If authentication is successful, the selected LAC will be loaded with the value specified in the LAC count field in the segment data. The new LAC value is set after Final Authorisation, which means that if the same LAC is used for authentication, it will get it’s new value after being incremented.

This command is authenticated with the currently selected key.

152.9.6 Change programmable key block

There are two change programmable key commands:

- Command A is used to change blocks starting at the first 256 octets
- Command B is used to change blocks starting at the last 112 octets

These commands changes five octets of the programmable key, starting at the address defined by the start address field in the segment data.

Generation of the five octets to be written to the programmable key is accomplished as follows:

Once the segment has been authorised by the Final Authorisation, the segment except the signature is inverted and passed once more through the Authentication Processor. The 24 bits of virtual fill z are

inserted without being inverted. The resulting 40-bit pseudo-signature is then loaded into the programmable key memory as follows:

- Bits 32 through 39 of the “pseudo-signature” are loaded into the octet indicated by the start address field in Bank A or Bank B.
- Bits 24 through 31 of the “pseudo-signature” are loaded into the octet at the next higher address (i.e. at start address + 1)
- And so on, until bits 0 through 7 are loaded into the octet indicated by the (start address field + 4)

The organization of the programmable key for the purpose of this command is depicted in table 2575.

*Table 2575.*Programmable key organization

Bank	Address	Authentication key		
A	0	40	W0(40 to 47)	47
	1	32	W0(32 to 39)	39
	2	24	W0(24 to 31)	31
	3	16	W0(16 to 23)	23
	4	8	W0(8 to 15)	15
	5	0	W0(0 to 7)	7
	6	40	W1(40 to 47)	47
	7	32	W1(32 to 39)	39
	255	16	W42(16 to 23)	23
B	0	8	W42(8 to 15)	15
	103	0	W59(0 to 7)	7
	104		59 C(59 to 56)	56
	105	55	C(55 to 48)	48
	106	47	C(47 to 40)	40
	107	39	C(39 to 32)	32
	108	31	C(31 to 16)	24
	109	23	C(23 to 16)	16
	110	15	C(15 to 8)	8
	111	7	C(7 to 0)	0

152.10 Programmable Key Memory

The weights of the programmable key is stored in an internal memory in the TCAU, while the coefficients are stored in registers. The memory is protected by TMR, to avoid errors in the weights. To remove bit flips from the memories, the programmable key is scrubbed (cf. section 152.10.1)

152.10.1Scrubber

The TCAU contains a scrubber that scrubs the programmable key memory whenever the core is idle. The scrubber reads each octet from the memory and writes them back unconditionally, thereby ensuring that all three memories get the same value.

152.11 Fixed Key

The fixed key is provided to the TCAU by means of the input signal *fixedkey*.

152.12 Back-end transmitter

The Back-end transmitter is a serial output that can be connected to a Segmentation sublayer. The TCAU can function without connecting the Back-end transmitter, if the Segmentation sublayer receives the segment independently from the TCAU, e.g. reads them from a memory. In this case, the authentication result signals from the Final Authorisation is sufficient for the Segmentation sublayer to determine if the segment is to be processed.

The Back-end transmitter receives octets directly from the Front-end receiver buffer.

152.12.1Non-authenticated segments

Segments transmitted on non-authenticated MAPs or while the AU is disabled, are output independently of the Authentication Processor. In this case, the entire segment is transmitted on the serial link.

152.12.2Authenticated data segments

For authenticated data segments, the Back-end transmitter transmits the segment in parallel with the authentication process, to avoid having to buffer the entire segment. To be able to remove the authentication tail from the segment, the Back-end transmitter always waits until the Front-end receiver buffer contains at least 10 octets or the Front-end receiver has received the entire segment. When the entire segment has been received and the buffer contains less than 10 octets, all of these belong to the authentication tail.

When the entire segment has been received, the Back-end transmitter waits until the Final Authorisation function has completed before completing the transfer. If the authorisation fails, the transfer is aborted and if the authorisation succeeds, the transfer is completed.

152.12.3Control Commands

If a segment containing a control command is received, the Back-end transmitter consumes all data without transmitting any octets.

152.13 Cold start state

After reset, the TCAU is in the following state:

- Fixed key is in use
- Principal LAC and Auxiliary LAC are all ones
- Recovery LAC is *rlac_rstval_c*
- Programmable key is unknown

152.14 Registers

The core does not implement any memory mapped registers.

152.15 Implementation

152.15.1Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core implements a separate generic for selecting between synchronous and asynchronous reset and resets all registers in both configurations.

152.16 Configuration options

Table 2576 shows the configuration options of the core (VHDL generics).

*Table 2576.*Configuration options

Generic	Function	Allowed range	Default
tech	Selects technology for the programmable key memory.	0 - NTECH	inferred
syncrst	Enable synchronous reset.	0 - 1	0
ft	Enable fault-tolerance against SEU errors. The core can optionally be implemented with fault-tolerance against SEU errors in the programmable key memory. The fault-tolerance is enabled through the ft VHDL generic to 1. If ft is set to 1, TMR memories will be implemented.	0-1	0

152.17 Signal descriptions

Table 2577 shows the interface signals of the core (VHDL ports).

*Table 2577.*Signal descriptions

Signal name	Field	Type	Function	Description	Active
RST_N	N/A	Input	Synchronous reset		Low
ARST_N	N/A	Input	Asynchronous reset		Low
CLK	N/A	Input	Clock		-
TLCRI	BEGINS	Input	Data unit start	Pulses when a new segment is to be sent from the transfer sublayer.	High
	PROGRESSES		In progress	High throughout the segment transfer.	High
	COMPLETES		End of data	Pulses to indicate the successful completion of a segment transfer.	High
	ABORTS		Abort data unit	Pulses to indicate the erroneous completion of a segment transfer.	High
	REQUEST		Data valid strobe	Pulses to transmit a new octet to the TCAU from the transfer sublayer.	High
	DATA		8-bit data	Segment data	-
TLCRO	ACCEPTING	Output	Ready for segment	Indicates that the TCAU is idle and can accept a new segment. When this output transits from low to high, it indicates that a segment has been fully processed, including any control command processing.	High
	BUSY		Not ready for octet	Indicates that the TCAU is ready to receive a new octet in an ongoing segment.	High

*Table 2577.*Signal descriptions

Signal name	Field	Type	Function	Description	Active
AUCRI	BEGINS	Output	Data unit start	Pulses when a new segment is to be sent to the segmentation sub-layer.	High
	PROGRESSES		In progress	High throughout the segment transfer.	High
	COMPLETES		End of data	Pulses to indicate the successful completion of a segment transfer.	High
	ABORTS		Abort data unit	Pulses to indicate the erroneous completion of a segment transfer, i.e. either aborted from the transfer sublayer or from an unsuccessful authentication.	High
	REQUEST		Data valid strobe	Pulses to transmit a new octet from the TCAU to the segmentation sublayer.	High
	DATA		8-bit data	Segment data	-
AUCRO	ACCEPTING	Input	Ready for segment	Indicates that the segmentation sublayer is idle and can accept a new segment.	High
	BUSY		Not ready for octet	Indicates that the segmentation sublayer is ready to receive a new octet in an ongoing segment.	High
TCAUI	ENABLE	Input	Enable the TCAU	If set to ‘0’, the TCAU will not perform any authentication, but will produce reports and forward segments to the segmentation sub-layer	High
	FIXEDKEY		Fixed key	Contains the weights and coefficients of the fixed key	-
	AUMAP		MAP selection for authentication	All segments where the 5 least significant bits of the segment header (the MAP address) are less than or equal to this input will be authenticated. All other segments (except control commands) will be forwarded to the segmentation sub-layer unaltered.	-
	RLAC_RSTVAL		Reset value for Recovery LAC	This value is read on the first clock cycle after reset is released and stored in the Recovery LAC as a start value.	-

*Table 2577.*Signal descriptions

Signal name	Field	Type	Function	Description	Active
TCAUO	AUANAVALID	Output	Analysis valid	Pulses whenever the AUANA and LASTMAP are updated with a new report. This is done when the TCAU has completed authentication or determined that a segment isn't to be authenticated.	High
	AUANA		Authentication analysis	Holds the TCAU report for the current segment when AUANAVALID is high.	-
	LASTMAP		Last addressed MAP	Holds the MAP ID of the current segment when AUANAVALID is high.	-
	PLAC		Principal LAC	Holds the 30-bit counter value of the Principal LAC.	-
	ALAC		Auxiliary LAC	Holds the 30-bit counter value of the Auxiliary LAC.	-
	RLAC		Recovery LAC	Holds the 8-bit counter value of the Recovery LAC.	-
	RLACWR		Recovery LAC write	Pulses when RLAC is updated.	High
	KEYINUSE		Currently selected key	0: fixed key is selected 1: programmable key is selected	-

152.18 Library dependencies

Table 2578 shows libraries used when instantiating the core (VHDL libraries).

*Table 2578.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	STDLIB	All	Common VHDL functions
TMTC	TMTC_TYPES	All	Signals and component declaration
TECHMAP	GENCOMP	All	Memory components

152.19 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.stdlib.all;
use grlib.config.all;
use techmap.gencomp.all;

library tmtc;
use tmtc.tmtc_types.all;

entity authentication_example is
  generic (
    memtech : integer := CFG_MEMTECH
    ft      : integer range 0 to 1 := 1);
  port (
    clk   : in  std_logic;
    rst_n : in  std_logic;
    tlcri : in  oci_request_type;
```

```
    tlcro : out oci_response_type;
    aucri : out oci_request_type;
    aucro : in  oci_response_type;
    tcaui : in  tcau_in_type;
    tcauo : out tcau_out_type);
end entity authentication_example;

architecture rtl of authentication_example is
  constant tcau_syncrst_c : integer := 1 - grlib_config_array_type.grlib_async_reset_enable;

  tcau0: tcau
    generic map (
      syncrst => tcau_syncrst_c,
      tech    => memtech,
      ft      => ft)
    port map (
      clk    => clk,
      arst_n => '1',
      rst_n  => rst_n,
      tlcri  => tlcri,
      tlcro  => tlcro,
      aucro  => aucro,
      aucri  => aucri,
      tcaui  => tcaui,
      tcauo  => tcauo);
end architecture rtl;
```

153 GRTC-COP - CCSDS/ECSS Telecommand Decoder

153.1 Overview

The Telecommand Decoder (GRTC-COP) is compliant with the Packet Telecommand protocol and specification defined by [ECSS-E-ST-50-04C]. The decoder is compatible with the [PSS-04-107] and [PSS-04-151] standards. The decoder is compliant with CCSDS recommendation [CCSDS-232.1-B-2], and compatible with [CCSDS-231.0-B-2] and [CCSDS-232.0-B-2]. The Telecommand Decoder (GRTC-COP) implements the Coding Sublayer (CL), the Transfer Sublayer and optionally the MAP Demultiplex Function of the Segmentation Sublayer.

In the Coding Sublayer (CL), the telecommand decoder receives bit streams on multiple channel inputs. The streams are assumed to have been generated in accordance with the Physical Layer specifications. In the Coding Sublayer, the decoder searches all input streams simultaneously until a start sequence is detected. Only one of the channel inputs is selected for further reception. The selected stream is bit-error corrected and the resulting corrected information is passed to the Transfer Sublayer.

The Transfer Sublayer receives a series of data octets from the Coding Sublayer corresponding to a decoded Transfer Frame. Upon reception of a “Candidate” Transfer Frame the Frame Validation Check Procedure is performed and static fields are verified to have expected values. Valid Transfer Frame handed over to the Communications Operation Procedure-1 (COP-1), consisting of the Frame Acceptance and Reporting Mechanism (FARM-1) in this receiving entity. Data units from Transfer Frames accepted by the FARM-1 are output on a parallel Octet Channel Interface.

The Command Link Control Word (CLCW) and the Frame Analysis Report (FAR), generated by the Coding Sublayer (CL) and the Transfer Sublayer, can be read as registers via the AMBA APB bus. The CLCW can be automatically transmitted to the Telemetry Encoder (TM) for transmission to the ground.

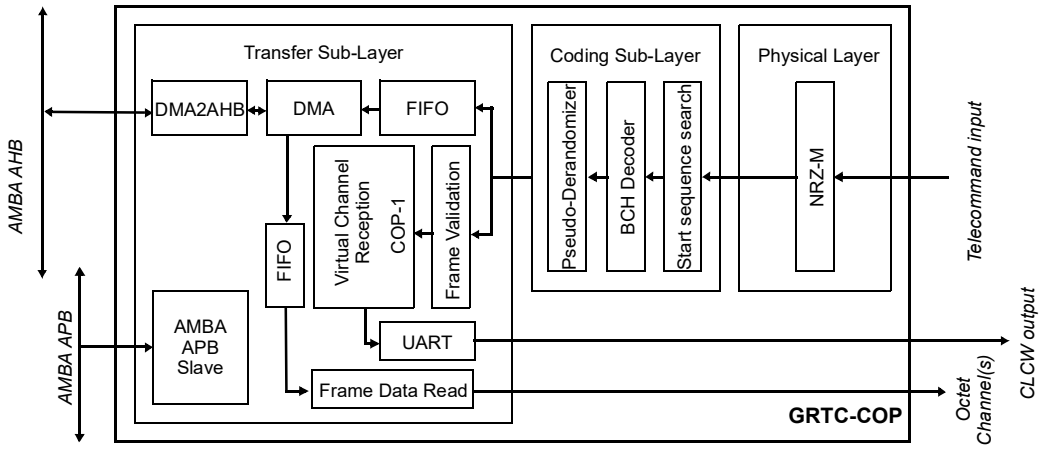


Figure 373. Block diagram

153.1.1 Concept

The telecommand decoder in this concept implements the Coding Sublayer and the Transfer Sublayer including COP-1 in hardware. Part of the Segmentation Sublayer may be implemented in hardware or fully implemented by software in the on-board processor.

The Coding Sublayer in the GRTC-COP core includes synchronisation pattern detection, channel selection, codeblock decoding. The Transfer Sublayer includes frame delimiting, fill removal, frame validation checks and the FARM-1 procedure. The MAP demultiplex function, part of the Segmentation Sublayer, can be enabled in hardware. It provides ability to demultiplex Frame Data Units from different Multiplexer Access Points (MAP) on a Virtual Channel. Other functions are to be implemented externally of the GRTC-COP, e.g. Authentication Unit (AU) and Command Pulse Distribution Unit (CPDU) are not implemented.

The Telecommand Decoder design concept is trailered to allow implementation of optional Security Layers for authentication and/or decryption. An 8-bit wide parallel Octet Channel Interface (shown in figure 377) is utilized for data transfer between sublayers and as output interface from the decoder. The Security Layer Option A may be implemented between the Coding Sublayer and the Transfer Sublayer to encapsulate Transfer Frames. Security Layer Option B may be implemented between the Transfer Sublayer and the Segmentation Sublayer to encapsulate Segments.

The GRTC has been split into two clock domains to facilitate higher bit rates and partitioning. The Telecommand Channel Layer (TCC) sub-core in one partition and the Transfer Sublayer (TCT), MAP Demultiplex and the Telecommand Decoder Interface (APB2TCD) in the other. A complete CCSDS packet telecommand decoder can be realized at software level according to the latest available standards, staring from or within the Segmentation Sublayer.

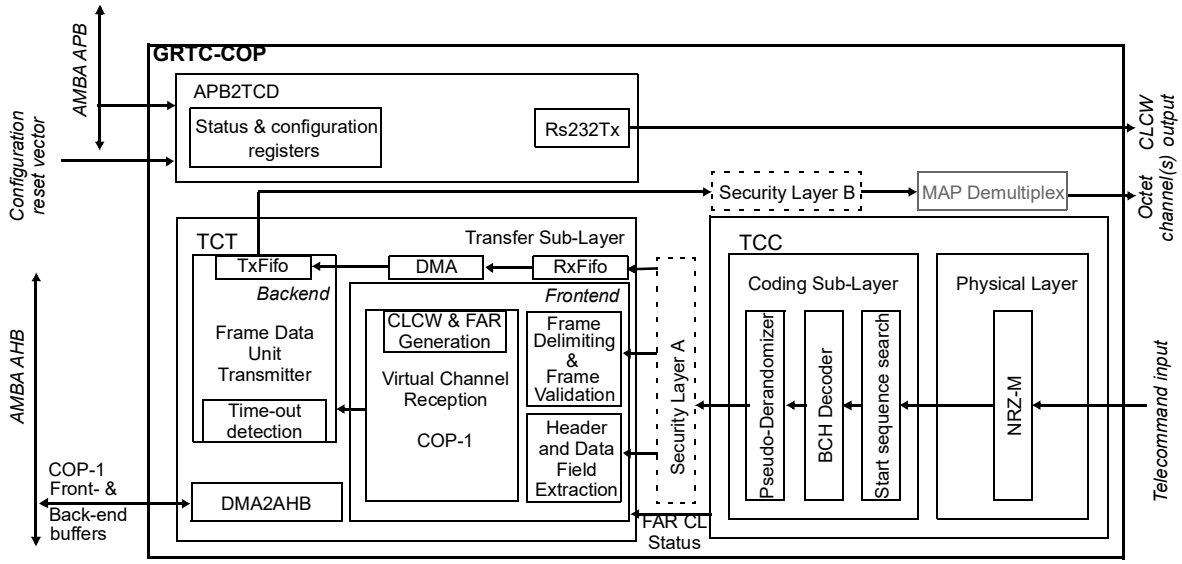


Figure 374. Detailed block diagram showing the internal structure

153.1.2 Functions and options

The following functions of the GRTC-COP are externally (pin) configurable:

- Polarity of RF Available and Bit Lock inputs
- Edge selection for input channel clock
- Segment Header consumption in MAP Demultiplex for chosen MAP interfaces
- User defined routing of MAP Channels to MAP interfaces

The following functions of the GRTC-COP are programmable by means of registers:

- Physical Layer - Non-Return-to-Zero - Mark decoding
- Coding Sub-layer - Pseudo-randomization
- Coding Sub-layer - PSS enable
- Transfer Sublayer - Frame Error Control Field Enabled
- Transfer Sublayer - Master Channel Identifier
- Transfer Sublayer - Virtual Channel Identifier
- Transfer Sublayer - FARM-1 Positive Window
- Transfer Sublayer - FARM-1 Negative Window
- Transfer Sublayer - FARM-1 AMBA Buffer Address

The the GRTC-COP contains pin configurable parameters that are applied at reset. After reset the configuration can be read and modified via the configuration registers on the APB. If the gHardware generic is set the Coding and Transfer Sublayer parameters are read only in the configuration registers, and become exclusively controlled by the value provided on the TCIn vector.

153.2 Data formats

153.2.1 Reference documents

[PSS-04-107]	Packet Telecommand Standard, Issue 2
[PSS-04-151]	Telecommand Decoder Standard, Issue 1
[CCSDS 231.0-B-2]	TC Synchronization and Channel Coding
[CCSDS 232.0-B-2]	TC Space Data Link Protocol
[CCSDS 232.1-B-2]	Communications Operation Procedure-1
[ECSS-E-ST-50-04C]	Space engineering - Space data links - Telecommand protocols, synchronization and channel coding

153.2.2 Waveforms

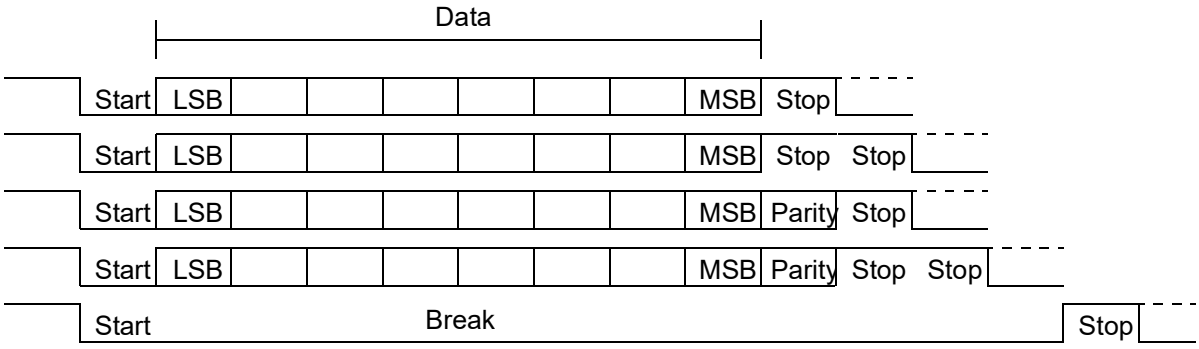


Figure 375. Bit asynchronous protocol versions

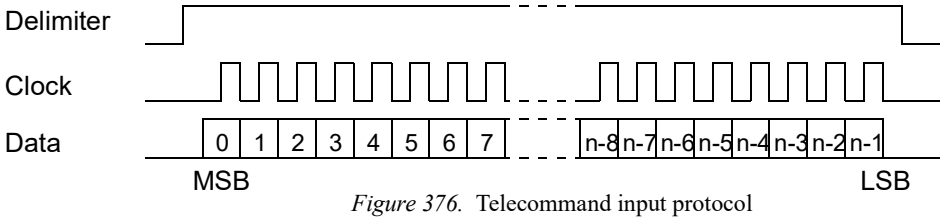


Figure 376. Telecommand input protocol

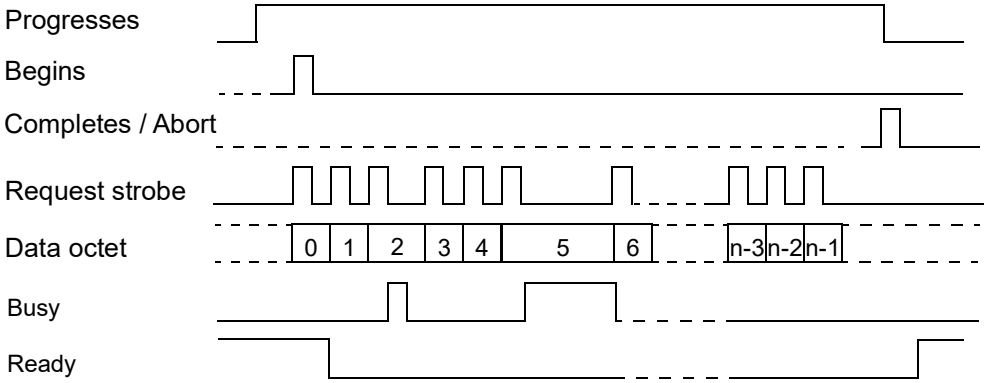


Figure 377. On-Chip parallel Octet Channel Interface

153.2.3 Data Unit Structures

Table 2579. Transfer Frame

Transfer Frame Primary Header	Transfer Frame Data Field	Frame Error Control Field
5 octets	Up to 1017 with FECF (1019 without)	2 octets

Table 2580. Transfer Frame Primary Header

Transfer Frame Version Number	Bypass Flag	Control Command Flag	Rsvd. Spare	Space-craft ID	Virtual Channel ID	Frame length	Frame Sequence Number V(S) / N(S)
2 bits	1 bit	1 bit	2 bits	10 bits	6 bits	10 bits	8 bits
2 octets					2 octets		1 octet

Table 2581. Bypass and Command Flags

Bypass Flag	Control Command Flag	Control Command Flag
0	0	Type-AD Transfer Frame, Sequence Controlled by FARM-1.
0	1	Reserved
1	0	Type-BD Transfer Frame, Expedited, FARM-1 sequence checks bypassed.
1	1	Type-BC Transfer Frame. Control Commands to Sequence Controlled (AD) service.

Table 2582. Segment Header

Sequence Flags	Multiplexer Access Point (MAP) ID
2 bits	6 bits
1 octet	

153.3 Coding Sublayer (CL)

The Coding Sublayer synchronises the incoming bit stream and provides an error correction capability for the Command Link Transmission Unit (CLTU). The Coding Sublayer receives a dirty bit stream together with control information on whether the physical channel is active or inactive for the multiple input channels.

The bit stream is assumed to be NRZ-L encoded, as the standards specify for the Physical Layer. As an option, it can also be NRZ-M encoded. There are no assumptions made regarding the periodicity or continuity of the input clock signal while an input channel is inactive. The most significant bit (Bit 0 according to [ECSS-E-ST-50-04C]) is received first.

Searching for the Start Sequence, the Coding Sublayer finds the beginning of a CLTU and decodes the subsequent codeblocks. As long as no errors are detected, or errors are detected and corrected, the Coding Sublayer passes clean blocks of data to the Transfer Layer which is implemented in software. When a codeblock with an uncorrectable error is encountered, it is considered as the Tail Sequence, its contents are discarded and the Coding Sublayer returns to the Start Sequence search mode.

The Coding Sublayer also provides status information for the CLCW and FAR, and it is possible to enable an optional de-randomiser according to [ECSS-E-ST-50-04C].

153.3.1 Synchronisation and selection of input channel

Synchronisation is performed by means of bit-by-bit search for a Start Sequence on the channel inputs. The detection of the Start Sequence is tolerant to a single bit error anywhere in the Start

Sequence pattern. The Coding Sublayer searches both for the specified pattern as well as the inverted pattern. When an inverted Start Sequence pattern is detected, the subsequent bit-stream is inverted till the detection of the Tail Sequence.

The detection is accomplished by a simultaneous search on all active channels. The first input channel where the Start Sequence is found is selected for the CLTU decoding. The selection mechanism is restarted on any of the following events:

- The input channel active signal is de-asserted, or
- a Tail Sequence is detected, or
- a Codeblock rejection is detected, or
- an abandoned CLTU is detected, or the clock time-out expires.

As a protection mechanism in case of input failure, a clock time-out is provided for all selection modes. The clock time-out expires when no edge on the bit clock input of the selected input channel in decode mode has been detected for a specified period. When the clock time-out has expired, the input channel in question is ignored (i.e. considered inactive) until its active signal is de-asserted (configurable with gTimeoutMask=1).

153.3.2 Codeblock decoding

The received Codeblocks are decoded using the standard (63,56) modified BCH code. Any single bit error in a received Codeblock is corrected. A Codeblock is rejected as a Tail Sequence if more than one bit error is detected. Information regarding Count of Single Error Corrections and Count of Accept Codeblocks is provided to the FAR. Information regarding Selected Channel Input is provided via a register.

153.3.3 De-Randomiser

In order to maintain bit synchronisation with the received telecommand signal, the incoming signal must have a minimum bit transition density. If a sufficient bit transition density is not ensured for the channel by other methods, the randomiser is required. Its use is optional otherwise. The presence or absence of randomisation is fixed for a physical channel and is managed (i.e., its presence or absence is not signaled but must be known a priori by the spacecraft and ground system). A random sequence is exclusively OR-ed with the input data to increase the frequency of bit transitions. On the receiving end, the same random sequence is exclusively OR-ed with the decoded data, restoring the original data form. At the receiving end, the de-randomisation is applied to the successfully decoded data. The de-randomiser remains in the “all-ones” state until the Start Sequence has been detected. The pattern is exclusively OR-ed, bit by bit, to the successfully decoded data (after the Error Control Bits have been removed). The de-randomiser is reset to the “all-ones” state following a failure of the decoder to successfully decode a codeblock or other loss of input channel.

153.3.4 Non-Return-to-Zero – Mark

An optional Non-Return-to-Zero – Mark decoder can be enabled by means of a register.

153.3.5 CLTU Decoder State Diagram

When in the decode state, each candidate codeblock is decoded in single error correction mode as described hereafter. Note that the diagram has been improved with explicit handling of different E2 events listed below.

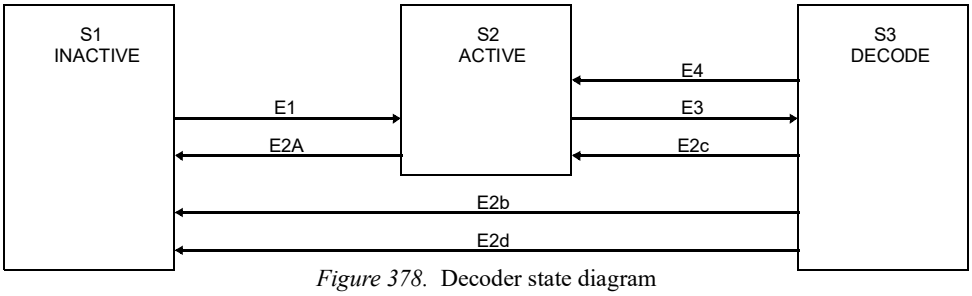


Figure 378. Decoder state diagram

State Definition:

- S1 Inactive
- S2 Search
- S3 Decode

Event Definition:

- E1 Channel Activation
- E2a Channel Deactivation - all inputs are inactive
- E2b Channel Deactivation - selected becomes inactive (CB=0 -> frame abandoned)
- E2c Channel Deactivation - too many codeblocks received (all -> frame abandoned)
- E2d Channel Deactivation - selected is timed-out (all -> frame abandoned)
- E3 Start Sequence Found
- E4 Codeblock Rejection (CB=0 -> frame abandoned)

153.3.6 Design specifics

The Coding Sublayer is supporting 1 to 8 channel inputs ([PSS-04-151] requires at least 4).

A codeblock is fixed to 56 information bits (as per CCSDS/ECSS).

The CCSDS/ECSS (1024 octets) or [PSS-04-151] (256 octets) standard maximum frame lengths are supported, being programmable via bit PSS in the GCR register. The former allows more than 37 codeblocks to be received.

The Pseudo-Randomiser decoder is included (as per CCSDS/ECSS), its usage being input signal programmable.

The Physical Layer input can be NRZ-L or NRZ-M modulated, allowing for polarity ambiguity. NRZ-L/M selection is programmable. This is an extension to ECSS: Non-Return to Zero - Mark decoder added, with its internal state reset to zero when channel is deactivated.

Note: If input clock disappears, it will also affect the codeblock acquired immediately before the codeblock just being decoded (accepted by [PSS-04-151]).

In state S1, all active inputs are searched for start sequence, there is no priority search, only round robin search. The search for the start sequence is sequential over all inputs: maximum input frequency = system frequency /(gIn+2)

The [PSS-04-151] specified CASE-1 and CASE-2 actions are implemented according to aforementioned specification, not leading to aborted frames.

Extended E2 handling is implemented:

- E2b Channel Deactivation - selected input becomes inactive in S3
- E2c Channel Deactivation - too many codeblocks received in S3
- E2d Channel Deactivation - selected input is timed-out in S3
  (design choice being: S3 => S1, abandoned frame)

153.4 Transfer Sublayer

The Transfer Sublayer receives a series of data octets from the lower layer (Coding Sublayer) corresponding to a decoded “Candidate” Transfer Frame, together with statistics later included in a Frame Analysis Report (FAR). Upon reception of a Transfer Frame the Frame Validation Check Procedure is performed on the received frame and static fields are verified to have expected values. If a check fails the Transfer Frame is discarded, otherwise it is handed over to the Frame Acceptance and Reporting Mechanism (FARM-1), compliant with the CCSDS recommendation [CCSDS 232.1-B-2]. Up to 1024 byte long Transfer Frames are supported.

The FARM-1 provides a Sequence-Controlled (AD) Service and an Expedited (BD) Service for receptions of data units on a chosen Virtual Channel. The service used for a received Transfer Frame is identified by the Bypass and Command Flags in the Transfer Frame Header described in table 2580 and 2581. When the FARM-1 procedure accepts a Transfer Frame of either type AD or BD the contained Frame Data Unit (FDU) is output from the Transfer Sublayer.

The Sequence-Controlled Service provides automatic retransmission request and guarantees that data units from Type-AD Transfer Frames are received in order without omission or duplication. The service is realized by an automatic ARQ procedure of the “go-back-n” type. The FARM-1 only accepts Type-AD Transfer Frames where the Frame Sequence Number N(S) in the received frame equals to the expected value V(R) in the FARM-1. After each accepted Type-AD frame the expected value V(R) is incremented.

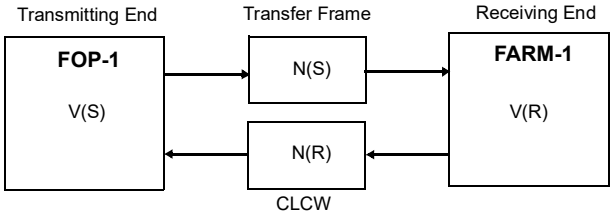


Figure 379. COP-1 Frame and Report Values (Sequence Controlled)

The Expedited (BD) Service to may be used to bypass the Frame Sequence Number check by transmitting Type-BD Transfer Frames. The Expedited Service is normally used either in exceptional operational circumstances, typically during recovery operations, or when a higher layer protocol provides retransmission capability.

The FARM-1 procedure uses the Communications Link Control Word (CLCW) for reporting back the Frame Acceptance status to the FOP-1. For transfer of AD-Frames this report contains the necessary control information allowing the FOP-1 to perform automatic retransmission, such as the Next Expected Frame Sequence Number N(R) and the Retransmit, Wait and Lockout flags. The successful reception of Type BD and BC Frames is observed in the modulo 4 FARM-B Counter in the CLCW. The successful reception and execution of BC-Type Transfer Frames may also be observed as a cleared Lockout flag or modified Report Value N(R) in the CLCW.

In addition Frame Analysis Reports (FAR) are generated by the Transfer Sublayer after each received frame independent from whether the Transfer Frame was declared invalid or accepted by the FARM-1. The FAR is not immediately essential for the Telecommand operation but is useful for diagnostics, and to determine the reason of failed Transfer Frame reception and processing.

153.4.1 FARM-1 Sliding Window

The COP-1 Sliding Window protects the Sequence Controlled service of the FARM-1 against unauthorised transmission of frames. The sequence numbers used for the sliding window are modulo-256 counters meaning that frame n+256 carries the same sequence number as frame n. The entities processing the sequence numbers and the data units involved in transferring them are illustrated in figure 379.

Only AD frames with Frame Sequence Number N(S) matching the V(R) variable in the FARM-1 are accepted. When a received AD-Frame is accepted the internal variable V(R) in the FARM-1 is incremented to the expected sequence number of the next received AD frame.

The positive window (PW) and negative window (NW) area parameters of the Sliding Window are individually configurable through the configuration registers.

- The positive window starts at V(R) and extends PW Transfer Frames in the positive direction. Transfer Frames in the positive area are discarded and the Retransmit Flag is set since a loss of a frame has been detected.
- Negative Window area starts at V(R) - 1 (the last accepted Transfer Frame) and extends NW Transfer Frames in the negative direction. Received frames in the area are discarded and may occur when forced-retransmissions are used by FOP-1.
- The remainder of the FARM Sliding Window is allocated to the Lockout area. Transfer Frames in this area put the FARM-1 into Lockout state. The Unlock BC-Type Frame can be used to unlock the FARM-1 again.

See [CCSDS 232.1-B-2] for detailed description of the FARM-1 states and state transitions.

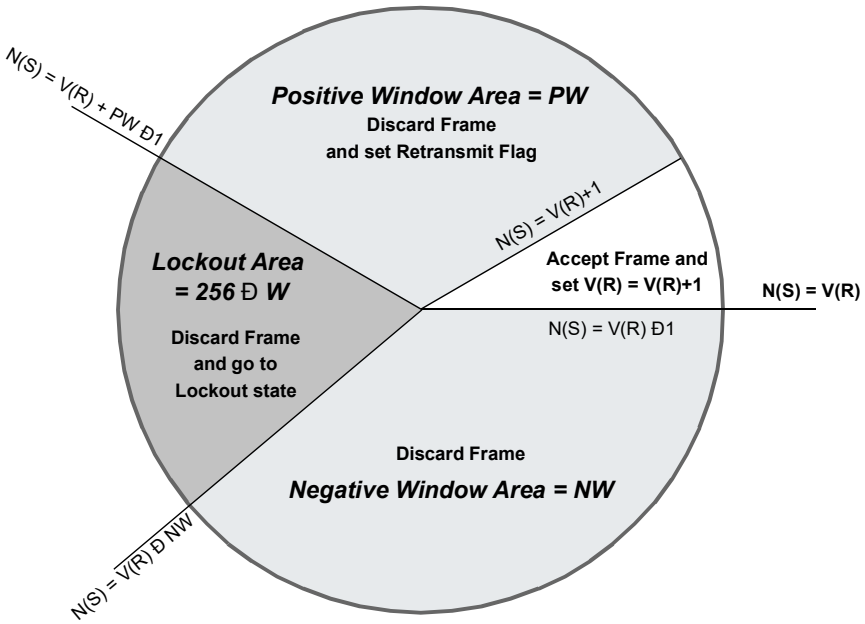


Figure 380. COP-1 FARM-1 Sliding window

Table 2583. Window Area

Window Area	Bounds		
Positive	N(S) > V(R)	and	N(S) ≤ V(R) + PW - 1
Negative	N(S) < V(R)	and	N(S) ≥ V(R) - NW
Lockout	N(S) > V(R) + PW - 1	and	N(S) < V(R) - NW

153.4.2 FARM-1 Control Commands

Two BC-Type Transfer Frames are defined for carrying Control Commands to the FARM-1 Sequence-Controlled service. Frames containing control commands are consumed in the FARM-1 and do not result in any output of data units from the Transfer Sublayer.

Two Control Command are available:

- Unlock Command performs Event 7 in FARM-1 when received. Command is identified by a single octet in the FDU set to zero. The FARM-1 Sequence Controlled service is in a locked state after reset.
- Set V(R) Command performs Event 8 in FARM-1 and update the V(R) variable in the Sequence-Controlled service. The command is identified when the FDU consists of three octets: 10000010 00000000 XXXXXXXX, where XXXXXXXX is the value to which the FARM should set the V(R).

153.4.3 Frame Validation Check Procedure

Each “Candidate” Transfer Frame input into the Frontend is undergoes a Frame Validation Check before being passed into the FARM-1. It undergoes a Frame Length check, a CRC check when Frame Error Control Field (FECF) is present, and validation of fields in the Transfer Frame. If performed checks fail the Transfer Frame is dropped and the Frontend module returns to an idle state.

The Transfer Sublayer is capable to receive Transfer Frames from one Virtual Channel on one Master Channel. Thereby the Virtual Channel Demultiplexing and Master Channel Demultiplexing Functions are implemented by verifying that corresponding Identifier fields in the Transfer Frame Header match with the Transfer Sublayer configuration.

Transfer Frame Header Validation

The following fields in the Transfer Frame Header are checked at the corresponding octets:

- Octet 0: Combination of a Sequence-Controlled Control Frame is reserved, thus flags Bypass=0, Control Command=1 in the Frame Header are illegal.
- Octet 0: Transfer Frame Version Number shall be set to “01” and Reserved field shall be cleared.
- Octet 0-1: Spacecraft Identifier (SCId) shall match the expected value.
- Octet 2: Virtual Channel Identifier (VCId) shall match the expected value. The Frontend will detect and report when the Least Significant Bit or the other bits of the VCId mismatch with two different codes in the FAR.
- Octet 4: The Frame Sequence Number N(S) shall be all zero if the Bypass Flag has been set.

Frame Length Validation and Integrity Check

The procedure verifies that the count of received octets correspond with the Frame Length field in the Frame Header. When the Frame Error Control Field (FECF) is enabled in Transfer Sublayer the CRC of the frame is also verified. The length of the frame is validated by ensuring:

- Enough octets have been received to from a complete Transfer Frame Header.
- The Frame Length Field in the header is greater than 7 when FECF enabled, otherwise greater than 5.
- Count of received octets is not less than the length indicated by the Frame Length Field.

When more octets are received than indicated by the Frame Length Field they are disregarded as fill data.

Optionally the *gCACCheck* generic can be enabled to check that the Frame Length Field in the Transfer Frame Header correlates with the Count of Accepted Codeblocks (CAC) reported from the Coding Sublayer. The validation is performed by ensuring that between 0 and 6 fill octets were received using

the expression:

$$\text{FrameLength} + 1 \leq \text{CAC} \times 7 \leq \text{FrameLength} + 1 + 6$$

When both *gCACCheck* and Frame FECF are enabled the Frontend also checks that at least two code-blocks were received.

FARM-1 Control Command Validation

In addition the following validation checks are performed on BC-Frames before consumed in the FARM-1:

- FDU Octet 0: When octet is zeroed this frame is registered as an Unlock command. The Frame Length Field shall to be 7 if FECF is enabled, otherwise it shall be 5.
- FDU Octet 1: If this octet is zeroed and the previous octet is 0x82 this frames is registered an a SetV(R) command. The Frame Length Field shall to be 9 if FECF is enabled, otherwise 7.
- If both Bypass and Control Command flags are set in the Transfer Frame Header but neither Unlock command or Set V(R) was detected the check fails.

153.4.4 Frame Analysis Report (FAR)

The FAR implementation is compatible with the [PSS-04-151] and contain survey data generated for each received data unit by the Coding and Transfer Sublayers. The survey data is accessible for read on the FAR and FARExt registers.

The Frame Analysis Report (FAR) interface supports 8 bit CAC field, as well as the 6 bit CAC field specified in [PSS-04-151]. When the PSS bit is cleared to '0', the Frame Analysis Report Extended (FARExt) register shall be read to get the full 8 bit CAC field. Otherwise the two most significant bits of the 8 bit CAC will be truncated because of intersection with the “Legal / Illegal” Frame Qualifier field in the FAR. These bits will however be all-zero when [PSS-04-151] compatible frame lengths are received or the PSS bit is set to '1'. The saturation is done at 6 bits when PSS bit is set to '1' and at 8 bits when PSS bit is cleared to '0'.

The “Status of Survey Data” and the “Legal/Illegal Frame Qualifier” fields in the FAR provide the failure reason why a Transfer Frame is not Accepted. The error encoded with the lowest value has the highest priority in both fields. If the Status of Survey Data is set to “ILLEGAL for one or multiple reason(s)” the reason for the failed validation of Transfer Frame fields is indicated in the Legal/Illegal Frame Qualifier field. The detailed FAR and FARExt bit field format can be observed in the Register section.

*Table 2584.*Frame Analysis Report

Field	Value (bin)	Meaning and Cause
Status of Survey Data	000 High Priority	Abandoned CLTU. Abandon (Abort) was signaled by the lower (coding) layer.
	001	Frame declared DIRTY. Enough octets for a Transfer Frame Header were not received, one of the frame length checks failed or the FECF was incorrect.
	010	Frame declared ILLEGAL for one reason. The reason can be read out from the Legal/Illegal Frame Qualifier.
	011	Frame declared ILLEGAL for multiple reasons. The reason with the highest priority (lowest value) can be read out from the Legal/Illegal Frame Qualifier.
	100	Frame (AD) discarded because of Lockout. The Frame arrived when the FARM-1 was in Lockout state or caused it.
	101	Frame (AD) discarded because of Wait The Frame arrived when the FARM-1 was in Wait state or caused it.
	110 Low Priority	Frame (AD) discarded because of N(S) or V(R) In the arrived AD-Frame the N(S) field header is not equal to FARM-1 V(R) variable.
	111	Frame (AD, BD or BC) Accepted by FARM-1
Legal/Illegal Frame Qualifier	000	No ILLEGAL report or “Cold Start”
	001	Error in fixed fields (Version & Reserved)
	010	Illegal combination (AC) of Bypass & Control Command flags
	011	Wrong Spacecraft ID (10 bits)
	100	Wrong VC ID (because of Bits 0 to 4 of ID)
	101	Wrong VC ID (because of Bit 5 of ID) (LSB)
	110	N(S) of BC or BD frame not set to all 0
	111	Wrong BC frame data format (not executable)
Count of Accepted Codeblocks per CLTU	natural	Information propagated from Coding Sublayer. “Cold Start” value 0
Count of Single Error Corrections	natural	Information propagated from Coding Sublayer. “Cold Start” value 0
Legal Frame Qualifier	00	AD frame
	01	Frame Not LEGAL or “Cold Start”
	10	BD frame
	11	BC frame
Last MAP Addressed	natural	Updated when AD or BD-Frame is accepted. Only applicable if Segments are transmitted on the Virtual Channel. “Cold Start” value “111111”
Selected Channel Input	natural	Coding channel Transfer Frame was received on. “Cold Start” value “111”

In addition the FAR is extended with capability to report FIFO overflow and DMA errors that occurred during Transfer Frame reception, by using otherwise unused field combinations shown in table 2585.

Table 2585. Frame Analysis Report - AMBA Bus failure

Status of Survey Data Field	Legal/Illegal Frame Qualifier	Description
Illegal for one reason	No ILLEGAL report	Frontend RxFIFO overflow.
Illegal for multiple reasons	No ILLEGAL report	Frontend DMA write AMBA error.

153.4.5 Detailed Operation

The Transfer Sublayer consist of three main internal modules, shown in figure 374: Frontend, Backend and DMA.

The Frontend module manages the frame reception, delimitation and fill data removal, frame validation and integrity checks, and the FARM-1 state machine. The Backend module provides output of Frame Data Units from accepted AD and BD-Type Frames. The Frontend and Backend modules each require a 1kB buffer, which is large enough to store a complete CCSDS Transfer Frame of maximum size. The buffers shall be located in user RAM on the AMBA bus. They are accessed by the internal DMA module that services buffer write and read requests on behalf of the Frontend and Backend modules respectively.

Frontend

The Frontend of the Transfer Sublayer waits in the idle state until the rising edge of the Progresses signal is detected on the Octet Channel Interface input that connects with lower layer. From the next system clock cycle and onward the Frontend accepts a new “Candidate” Transfer Frame. Individual data octets are sampled from the input interface on the rising edge of the data Request strobe. The Octet Channel Interface input is internally resynchronised with two synchronization registers, but the control signals have one additional register for rising edge detection. Therefore it is required that the data octet is held valid for one cycle after the Request strobe has been set.

As each octet arrives into the Transfer Sublayer the following actions are performed:

- Octets are stored to the frontend buffer using the DMA Write operation to memory via the AMBA bus. If more than 1024 octets are input into the Transfer Sublayer only the first 1024 are stored to the frontend buffer.
- Fields that are expected to be fixed in the Transfer Frame Header, including MCID and VCID, are checked and remaining fields are extracted for later processing. In addition the first three octets of the Frame Data Field are checked for BC-Frames containing FARM-1 commands.
- When FECF is enabled for Transfer Frames the CRC is continuously computed for each received octet until same number of octets have been processed by the CRC computation procedure that is indicated by the extracted Frame Length field.

The reception of the Transfer Frame ends when the rising edge of either the Completes or Abort (Abandoned) signal is detected on the octet input interface. This makes the Frontend to flush the DMA write FIFO buffer and wait until emptied. The FAR input with the status from the lower layer (Coding Sublayer) must be supplied no later than the assertion of either Completes or Abort.

When the write FIFO buffer has been emptied the Frontend checks that no buffer overflow or DMA error has occurred, followed by checks of the Transfer Frame CRC and Frame Validation criteria. If the checks pass the FARM-1 will take action based on the values of previously extracted fields, otherwise the fame is dropped and the Frontend returns to an idle state.

The Frontend will always generate a new FAR just before returning to the idle state independent of whether checks passed. A new CLCW is generated by the Transfer Sublayer whenever a valid frame is proceed by the FARM-1 state machine.

Backend

When the FARM-1 accepts a data frame (AD or BD) and the Backend is idle the fontend buffer is forwarded to the Backend module by internally swapping buffer references between the Frontend and the Backend modules. The Backend module then outputs the FDU on a parallel Octet Channel Interface. The receiving end on the Octet Channel Interface can control the rate that octets are output from the Transfer Sublayer by asserting the Busy and Ready handshake signals of the interface.

The action taken when the FARM-1 receives a valid data type frame but the Backend is busy transmitting a data unit does depend on the frame type residing in the Frontend.

- When Backend module is busy AD-Frames arriving to the FARM-1 become discarded, and the Retransmit and Wait flags in the CLCW become set. When the Backend buffer is later freed the FARM-1 state is updated and a new CLCW is generated with a cleared Wait flag.
- When a BD-Frame is accepted by the FARM-1 and the Backend module is busy the Backend is forced to Abort the FDU already being output in order to immediately output the data from the just accepted BD-Frame. The receiving end shall then discard the incompletely transferred data unit.

If an FDU is not readout fast enough before the timeout counter saturates, configurable via *gTimeoutTL* generic, an abort of the Octet Channel is also signalled.

153.4.6 Direct Memory Access (DMA)

The DMA engine implemented in the Transfer Sublayer is an AMBA AHB bus master that provides the Frontend and Backed modules access to buffers for Transfer Frame storage. The buffers are located in user memory (e.g. SRAM located in on-board processor board or on-chip RAM). The DMA feature set is optimized for the task and does only perform aligned 32 bit accesses. This may lead to that additional bytes within the 32-bit word may be written or unused bytes may be read.

The buffer id currently allocated the Frontend and Backend of the Transfer Sublayer can be monitored on the Transfer Sublayer Status Register. The buffer id is encoded with one bit and corresponds to the 10th AMBA address bit. The AMBA address of the buffer is found by concatenating the FARM-1 Buffer Address register field with the buffer id.

Frontend

The octets received from the lower layer are locally buffered in a FIFO that is at least four 32-bit words deep, before the DMA writes them on the AMBA bus. The conversion from the input data width (8 bit) to 32 bit system word width, is performed in the FIFO. The FIFO buffer prohibits overflow from occurring when the system bus is occupied by other devices and provide improved burst capabilities. When there is enough data in the write buffer the DMA engine has capability to perform double word bursts and single word write accesses to the AHB bus. Whenever enough data exists in the buffer a single word write is performed.

If an overflow or a DMA write error would occur the Fontend continues to push received octets from lower layer into the write FIFO. The Frontend handles the issue after the lower layer has signalled end of Transfer Frame, and then discards the Frontend buffer containing the partially stored frame.

The DMA is configured to first serve the Frontend access then the Backend in order to reduce risk of overflow in the receive FIFO due to high AMBA bus load.

Backend

When the FARM-1 accepts a data frame the Frame Data Unit is fetched by the DMA into at least four 32-bit word deep FIFO buffer. The AMBA read accesses are always performed as 32-bit single accesses. If the DMA encounters an AMBA bus error during read of the FDU the data output from the Transfer Layer is immediately aborted and the backend buffer is freed.

153.5 Segmentation Sublayer

The MAP Demultiplexing function of the Segmentation Sublayer can be enabled in hardware by setting the generic *gNumMap* larger than one. The *gNumMap* generic then specifies number MAP Hardware Interfaces realised as Octet Channel Interface outputs. This unit is disabled when *gNumMap* is set to 1. The output interface from the Transfer Sublayer carrying FDU is then directly interfaced to the GRTC-COP top-level on the Octet Channel Interface 0.

The MAP Demultiplexer routes received Segments to MAP HW Interfaces based on the MAP Channel Id in the Segment Header – the first octet in a Segment, shown in table 2582. This unit does internally buffer one octet and will immediately forward a possible abort signal from the lower layer to currently active MAP HW Interface output.

Two methods of allocating MAP Channels to MAP HW Interfaces can be selected using *gMAPImpl* generic.

- When *gMAPImpl*=0 MAP Channels are sequentially assigned MAP HW Interfaces. Segments with MAP Identifier 0 are output on Octet Channel Interface 0, Segments with MAP ID 1 are output on Octet Channel Interface 1, and so on. If less than 64 MAP HW Interfaces implemented the remaining MAP Channels are output on the last available MAP HW Interfaces.
- When *gMAPImpl*=2 the assignment is configured by the *map_route* input vector where each of the 64 possible MAP Channel shall be assigned a MAP HW Interface.

The MAP Demultiplex unit is able to consume the Segment Header for specific MAP HW Interfaces. The feature is enabled by setting '1' the corresponding bit in the *drop_seghed* input vector, and setting the *gMAPImpl*>0.

153.6 Command Link Control Word interface (CLCW)

The Command Link Control Word (CLCW) is generated by the GRTC-COP Telecommand decoder. It is inserted into Telemetry Transfer Frames by the Telemetry Encoder (TM) when the Operation Control Field (OPCF) is present. The CLCW is made available via three interfaces from the GRTC-COP:

- As an AMBA APB status register of the telecommand decoder.
- As a 32-bit word output from the GRTC-COP top level coupled with a CLCW New Event signal. A pulse is provided when the FARM-1 state machine has executed. Note that a pulse on the Event signal is not generated when bit 16 (No RF Available) and 17 (No Bit Lock) of the CLCW change.
- On a PacketAsynchronous interface (PA) is used for the transmission of the CLCW from the telecommand decoder. The protocol is fixed to 115200 baud, 1 start bit, 8 data bits, 1 or 2 stop bits (configured by generics), with a BREAK command for message delimiting (sending 13 bits of logical zero).

The CLCWs are automatically transferred over the PA interface after reset, on each update from the FARM-1 and on each change of the bit 16 (No RF Available) and 17 (No Bit Lock). The detailed CLCW bit field format can be observed in the Register section.

*Table 2586.*CLCW transmission protocol

Byte Number	CLCWR register bits	CLCW contents						
First	[31:24]	Control Word Type	CLCW Version Number	Status Field	COP In Effect			
Second	[23:16]	Virtual Channel Id	Reserved Field					
Third	[15:8]	No RF Available	No Bit Lock	Lock Out	Wait	Retransmit	Farm B Counter	Report Type
Fourth	[7:0]	Report Value						
Fifth	N/A	[RS232 Break Command]						

153.6.1 Miscellaneous

The accuracy of the transmission or reception baud rate of the bit asynchronous serial interface is dependent on the selected system frequency and baud rate. The number of system clock periods used for sending or receiving a bit is directly proportional to the integer part of the division of the system frequency with the baud rate.

The BREAK command received on the bit asynchronous serial interface is a sequence of logical zeros that is at least one bit period longer than the normal byte frame, i.e. start bit, eight data bits, optional parity, one or two stop bits. When transmitted, it is always 13 bits.

153.7 Optional Security Layer A Interface

The Optional Security Layer A may be implemented to either decipher or authenticate the received TC Transfer Frame block, including the Transfer Frame Header.

The Octet Channel input interface into the Transfer Sublayer provides output signals for data flow control that may be interfaced with a Security Layer located directly below the Transfer Sublayer. A Ready signal to indicate ready for new frame, and a Busy signal to indicate a full receive FIFO. After the Busy signal is asserted two more octet may still be transferred into the transfer-layer without causing overflow. (Octets are temporarily stored in a 8-bit to 32-bit realignment register, before insterted into the FIFO).

Note that the flow control interface on the Transfer Sublayer's Octet Channel Interface input provides flow control for the Frontend, and does not consider the state of the Backend. A slow receiver on the Transfer Sublayer output interface may cause the Backend to still be busy when the next Transfer Frame is reaches the FARM-1. In such case the AD-Type frame residing in the Frontend buffer is discarded and the CLCW Wait flag is asserted by the FARM Event 2.

The the Coding Sublayer provides statistics that the Transfer Sublayer later includes into the FAR. The Security Layer should sample the statistics form the Coding Sublayer when the CLTU arrives and provide them to the Transfer Sublayer when the authenticated/deciphered Transfer Frame is output.

153.8 Configuration Interface (AMBA APB slave)

The AMBA APB slave interface supports 32 bit wide data input and output. Since each access is a word access, the two least significant address bits are assumed always to be zero, address bits 7:0 are decoded. Misaligned addressing is not supported. For read accesses, unmapped bits are always driven to zero.

Channel re-initialisation and reset of the Coding Sublayer, Transfer Sublayer and the MAP Demultiplex can be performed via the Control Register. The duration of this activity is 8 system (HCLK)

clock periods and is complete when the Pending bit is deasserted. The Receiver Enable bit in the Control Register is cleared after re-initialisation and shall be set ‘1’ to enable Coding Sublayer reception again.

A full re-initialisation and reset of the Telecommand Decoder can be performed via the Global Re-initialisation Register that clears APB configuration registers and resets the PacketAsynchronous CLCW transmitter, in addition to the Channel re-initialisation.

153.9 Interrupts

The GRTC-COP has interrupt outputs that are asserted for at least two clock periods. The occurrence of one of the following events may produce interrupts:

- CLCW interrupt ‘FARM-1 CLCW event’
- FAR interrupt ‘Status Survey Data’
- CLCW interrupt ‘Bit Lock’
- CLCW interrupt ‘RF Available’

The events are either generated on four separate consecutive interrupt lines, or are generated on a single shared interrupt when *singleirq* generic is set. The first interrupt line is selected using the *pirq* generic. If separate interrupt lines are used these are allocated consecutively starting with *pirq*.

The core generates the interrupts defined in table 2587. Also see the Interrupt Registers section 153.11.15.

*Table 2587.*Interrupts

Interrupt offset	Interrupt name	Description
1:st	RFA	RF Available changed
2:nd	BLO	Bit Lock changed
3:rd	FAR	New FAR Available
4:th	CLCW	CLCW FARM-1 Event

153.10 Miscellaneous

153.10.1Numbering and naming conventions

Convention according to the CCSDS recommendations, applying to time structures:

- The most significant bit of an array is located to the left, carrying index number zero.
- An octet comprises eight bits.

*Table 2588.*CCSDS n-bit field definition

CCSDS n-bit field		
most significant		least significant
0	1 to n-2	n-1

Convention according to AMBA specification, applying to the APB/AHB interfaces:

- Signal names are in upper case, except for the following:
- A lower case 'n' in the name indicates that the signal is active low.
- Constant names are in upper case.
- The least significant bit of an array is located to the right, carrying index number zero.

- Big-endian support.

*Table 2589.*AMBA n-bit field definition

AMBA n-bit field		
most significant		least significant
n-1	n-2 down to 1	0

153.10.2Performance

The uplink bit rate is supported in the range of 1 kbits/s to 1 Mbits/s.

The bit rate is set to 115200 bit/s for the PacketAsynchronous (PA) interfaces.

153.11 Registers

The core is programmed through registers mapped into APB address space.

*Table 2590.*GRTC registers

APB address offset	**Register**
0x00	Global Re-initialisation Register (GRR)
0x04	Global Control Register (GCR)
0x08	Physical Interface Mask Register (PMR)
0x0C	Physical Interface Register (PHIR)
0x10	Spacecraft Identifier Register (SIR)
0x14	Virtual Channel Identifier Register (VIR)
0x18	FARM-1 Window Area Register (WIN)
0x1C	FARM-1 Buffer Address Register (BUF)
0x20	Frame Acceptance Report Register (FAR)
0x24	Frame Acceptance Report Extended Register (FARExt)
0x28	CLCW Register (CLCW)
0x2C	Transfer Sublayer Status Register (TSR)
0x30	Control Register (COR)
0x40	Capability Register (CAP)
0x60	Interrupt Status Register (ISR)
0x64	Interrupt Mask Register (IMR)

153.11.1 Global Re-initialisation Register (GRR)

Table 2591. 0x00 - GRR - Global Re-initialisation Register

31 24	23 1	0
SEB	RESERVED	SRST
0	0	0
w	r	rw

31: 24 SEB (Security Byte):

Write: '0x55'= the write will have effect (the register will be updated).
Any other value= the write will have no effect on the register.

Read: All zero.

23: 1 RESERVED

Write: Don't care.

Read: All zero.

0 System reset (SRST): **[1]**

Write: '1'= initiate reset,'0'= do nothing

Read: '1'= unsuccessful reset, '0'= successful reset

153.11.2 Global Control Register (GCR)

Table 2592. 0x04 - GCR - Global Control Register

31 24	23 17	16	15 13	12	11	10	9 0
SEB	RESERVED	FECF	RESERVED	PSS	NRZM	PSR	RESERVED
0	0	*	0	*	*	*	0
w	r	rw*	r	rw*	rw*	rw*	r

31: 24 SEB (Security Byte):

Write: '0x55'= the write will have effect (the register will be updated).
Any other value= the write will have no effect on the register.

Read: All zero.

23: 17 RESERVED

Write: Don't care.

Read: All zero.

16 FECF (Transfer Frame FECF enable)

Write/Read: '0'= disable, '1'= enable [Read-only when gHardware VHDL generic set.]

15: 13 RESERVED

Write: Don't care.

Read: All zero.

12 PSS (ESA/PSS enable) [5]

Write/Read: '0'= disable, '1'= enable [Read-only when gHardware VHDL generic set.]

11 NRZM (Non-Return-to-Zero Mark Decoder enable)

Write/Read: '0'= disable, '1'= enable [Read-only when gHardware VHDL generic set.]

10 PSR (Pseudo-De-Randomiser enable)

Write/Read: '0'= disable, '1'= enable [Read-only when gHardware VHDL generic set.]

9: 0 RESERVED

Write: Don't care.

Read: All zero.

Power-up default: 0x00001000, The default value depends on the TCC_PSS, TCC_Mark, TCC_Pseudo, TCT_FECF inputs.

153.11.3 Physical Interface Mask Register (PMR)

Table 2593. 0x08 -PMR - Physical Interface Mask Register

31 24	8 7	0
SEB	RESERVED	MASK
0	0	0
w	r	rw

31: 24 SEB (Security Byte):

Write: '0x55'= the write will have effect (the register will be updated).
Any other value= the write will have no effect on the register.

Read: All zero.

23: 8 RESERVED

Write: Don't care.

Read: All zero.

7: 0 MASK

Write: Mask TC input when set, bit 0 corresponds to TC input 0

Read: Current mask

153.11.4 Physical Interface Register (PHIR)

*Table 2594.*0x0C - PHIR - Physical Interface Register

31 16	15 8	7 0
RESERVED	RFA	BLO
0	*	*
r	r	r

31: 16 RESERVED

Write: Don't care.

Read: All zero.

15: 8 RFA (RF Available) [3]

Only implemented inputs are taken into account. All other bits are zero.

Write: Don't care.

Read: Bit[8] = input 0, Bit[15] = input 7

7: 0 BLO (Bit Lock) [3]

Only implemented inputs are taken into account. All other bits are zero.

Write: Don't care.

Read: Bit[0] = input 0, Bit[7] = input 7

Frontgrade Gaisler AB
Kungsgatan | SE-411 19 | Goteborg | Sweden
+46 31 7758650 | frontgrade.com/gaisler

153.11.5 Spacecraft Identifier Register (SIR)

Table 2595. 0x10 - SIR - Spacecraft Identifier Register

31 24	10 9	0
SEB	RESERVED	SCID
0	0	*
w	r	rw*

31: 24 SEB (Security Byte):

Write: '0x55'= the write will have effect (the register will be updated).
Any other value= the write will have no effect on the register.

Read: All zero.

23: 10 RESERVED

Write: Don't care.

Read: All zero.

9: 0 SCID (Spacecraft Identifier)

Write/Read Bit[9]=MSB, Bit[0]=LSB [Read-only when gHardware VHDL generic set.]

Power-up default: Depends on SCID input configuration.

153.11.6 Virtual Channel Identifier Register (VIR)

Table 2596. 0x14 - VIR - Virtual Channel Identifier Register

31 24	23 6	5 0
SEB	RESERVED	VCID
0	0	*
w	r	rw*

31: 24 SEB (Security Byte):

Write: '0x55'= the write will have effect (the register will be updated).
Any other value= the write will have no effect on the register.

Read: All zero.

23: 6 RESERVED

Write: Don't care.

Read: All zero.

5: 0 VCID (Virtual Channel Identifier)

Write/Read Bit[9]=MSB, Bit[0]=LSB [Read-only when gHardware VHDL generic set.]

Power-up default: Depends on VCID input configuration.

153.11.7 FARM-1 Window Area Register (WIN)

Table 2597. 0x18 - WIN - FARM-1 Window Area Register

31 24	23 16	15 8	7 0
SEB	RESERVED	NW	PW
0	0	*	*
w	r	rw*	rw*

31: 24 SEB (Security Byte):

Write: '0x55'= the write will have effect (the register will be updated).
Any other value= the write will have no effect on the register.

Read: All zero.

23: 6 RESERVED

Write: Don't care.

Read: All zero.

15: 8 NW (Negative Window Area)

Write/Read: Bit[7]=MSB, Bit[0]=LSB. Must be larger than 0. [Read-only when gHardware VHDL generic set.]

7: 0 PW (Positive Window Area)

Write/Read: Bit[7]=MSB, Bit[0]=LSB. Must be larger than 0. [Read-only when gHardware VHDL generic set.]

Power-up default: Depends on NegWin and PosWin input configuration.

153.11.8 FARM-1 Buffer Address Register (BUF)

Table 2598. 0x1C - BUF - FARM-1 Buffer Address Register

31 24	23 3	2 0
SEB	BAD	RESERVED
0	*	0
w	rw*	r

31: 24 SEB (Security Byte):

Write: '0x55'= the write will have effect (the register will be updated).
Any other value= the write will have no effect on the register.

Read: All zero.

23: 5 BAD (FARM-1 Buffer Address)

Write/Read Bit[23]=AMBA bit[31] MSB, Bit[3]=AMBA bit[11]
[Read-only when gHardware VHDL generic set.]

2: 0 RESERVED

Power-up default: Depends on FARMAddr input configuration.

153.11.9 Frame Acceptance Report Register (FAR)

Table 2599. 0x20 - FAR - Frame Acceptance Report Register

31	30 28	27 25	24 19	18 16	15 14	13 11	10 9 5	4 0
SSD	FA	FQ	CAC	CSEC	LFQ	SCI	LMA	RESERVED
0	0	0	0	0	0b01	0b111	0b111111	0
r	r	r	r	r	r	r	r	r

31 SSD (Status of Survey Data) (see [PSS-04-151])
- Write: Don't care.
- Read: Automatically cleared to 0 when any other field is updated. Automatically set to 1 upon a read.

30: 25 FA (Frame Analysis)
- Write: Don't care.
- Read: Information obtained from transfer sublayer.

30: 25 FQ "Legal/Illegal" Frame Qualifier
- Write: Don't care.
- Read: Information obtained from transfer sublayer.

24: 19 CAC (Count of Accept Codeblocks) (see [PSS-04-151])
- Write: Don't care.
- Read: Information obtained from coding sublayer. [2]

18: 16 CSEC (Count of Single Error Corrections) (see [PSS-04-151])
- Write: Don't care.
- Read: Information obtained from coding sublayer.

15: 14 LFQ (Legal Frame Qualifier)
- Write: Don't care.
- Read: Information obtained from transfer sublayer.

13: 11 SCI (Selected Channel Input) (see [PSS-04-151])
- Write: Don't care.
- Read: Information obtained from coding sublayer.

10: 5 LMA (Last MAP Addressed)
- Write: Don't care.
- Read: Information obtained from transfer sublayer.

4: 0 RESERVED (AU not implemented)
- Write: Don't care.
- Read: All zero.

153.11.10 Frame Acceptance Report Extended Register (FARExt)

Table 2600. 0x24 - FARExt - Frame Acceptance Report Extended Register

31 8	7 0
RESERVED	CAC
0	0
r	r

31 RESERVED
- Write: Don't care.
- Read: All zero.

7: 0 CAC (Count of Accept Codeblocks)
- Write: Don't care.
- Read: NOTE: Only updated when FAR register is read.

153.11.11 CLCW Register (CLCW)

*Table 2601.*0x28 - CLCW - CLCW Register (see [CCSDS 232.0-B-2])

31	30 29	28 26	25 24	23 18	17 16	15	14	13	12	11	10 9	8	7 0
CWTY	VNUM	STAF	CIE	VCI	RESERVED	NRFA	NBLO	LOUT	WAIT	RTMI	FBCO	RES	RVAL
0	0	0	0b01	*	0	0	0	1	0	0	0	0	0
r	r	r	r	r	r	r	r	r	r	r	r	r	r

31 CWTY (Control Word Type, set to‘0’)

30: 29 VNUM (CLCW Version Number, set to “00”)

28: 26 STAF (Status Fields, unused)

25: 24 CIE (COP In Effect, set “01” for COP-1)

23: 18 VCI (Virtual Channel Identification). Set by Transfer Sublayer configuration.

17: 16 Reserved (PSS/ECSS requires “00”)

15 NRFA (No RF Available)

- Write: Don’t care.
- Read: Based on discrete inputs.

14 NBLO (No Bit Lock)

- Write: Don’t care.
- Read: Based on discrete inputs.

13 LOUT (Lock Out)

12 WAIT (Wait)

11 RTMI (Retransmit)

10: 9 FBCO (FARM-B Counter)

8 Reserved

7: 0 RVAL (Report Value)

153.11.12 Transfer Sublayer Status Register (TSR)

Table 2602. 0x2C - TSR- Transfer Sublayer Status Register

31 6	5	4	3	2	1	0
RESERVED	ABORT	PRG	RES	BBE	RES	BFE
0	0	0	0	0	0	0
r	r	r	r	r	r	r

31: 3 RESERVED

Write: Don't care.

Read: All zero.

5 ABORT (Last Frame Data Unit aborted)

Write: Don't care.

Read: Set high if last output was aborted. Cleared when new data unit output begins.

4 PRG (Frame Data Unit output in progress)

Write: Don't care.

Read: Address bit[10] of Backend buffer address. Bits [31...11] available on FARM-1 Buffer Address Register. Bit[0]=LSB.

3 RESERVED

Write: Don't care.

Read: All zero.

2: BBE (Buffer Identifier Backend)

Write: Don't care.

Read: Address bit[10] of Backend buffer address. Bits [31...11] available on FARM-1 Buffer Address Register. Bit[0]=LSB.

1 RESERVED

Write: Don't care.

Read: All zero.

0: BFE (Buffer Identifier Frontend)

Write: Don't care.

Read: Address bit[10] of Frontend buffer address. Bits [31...11] available on FARM-1 Buffer Address Register. Bit[0]=LSB.

153.11.13 Control Register (COR)

*Table 2603.*0x30 - COR - Control Register

31 24	23 10	9	8	7 1	0
SEB	RESERVED	CRST	PEND	RESERVED	RE
0	0	0	0	0	*
w	r	rw	r	r	rw

31: 24 SEB (Security Byte):

Write: '0x55'= the write will have effect (the register will be updated).
Any other value= the write will have no effect on the register.

Read: All zero.

23: 10 RESERVED

Write: Don't care.

Read: All zero.

9 CRST (Channel re-initialisation /reset) [4]

Write: '1'= initiate channel re-init,'0'= do nothing

Read: '1'= unsuccessful re-init, '0'= successful re-init

8 PEND (Channel re-initialisation / reset pending)

Write: Don't care.

Read: '1'= channel re-init asserted (for 8 cycles), '0'= channel re-init deasserted

7: 1 RESERVED

Write: Don't care.

Read: All zero.

0 RE (Receiver Enable) [Always enabled when gHardware VHDL generic set.]

The TCActive input of the receiver are masked when the RE bit is disabled.

Read/Write: '0'= disabled, '1'= enabled

Power-up default: 0x00000000 [0x00000001 when gHardware VHDL generic set.]

153.11.14 Capability Register (CAP)

*Table 2604.*0x40 - CAP - Capability Register

31	0
RESERVED	SIRQ
0	*
r	r

31: 1 RESERVED

0 SIRQ (Single IRQ)

Read: '0'=multiple, '1'=single

Legend:

[1] The global system re-initialisation/reset caused by the SRST-bit in the GRR-register results in the following actions:

- Initiated by writing a '1", gives '0' on read-back when the reset was successful.
- No need to write a '0' to remove the reset.
- Unconditionally, means no need to check/disable something in order for this reset-function to correctly execute.
- Could of course lead to data-corruption coming/going from/to the reset core.
- Resets the complete core (all logic, buffers & register values)
- Behaviour is similar to a power-up.

[2] The FAR register supports the CCSDS/ECSS standard frame lengths (1024 octets), requiring an 8 bit CAC field instead of the 6 bits specified for PSS. The full 8-bit CAC field including the two most significant bits is therefore

made available on the FARExt register. This is only the case when the PSS bit is set to '0'. NOTE: FARExt register content only updates when FAR register is read.

[3] The number of channels are controlled with the gRFAvailable and gBitLock VHDL generics, respectively.

[4] The channel re-initialisation / reset caused by the CRST-bit in the COR-register results in the following actions:

- Initiated by writing a ‘1”, gives ‘0’ on read-back when the reset was successful.
- No need to write a ‘0’ to remove the reset.
- All other bit’s in the COR are neglected (not looked at) when the CRST-bit is set during a write, meaning that the value of these bits has no impact on the register-value after the reset.
- Unconditionally, means no need to check/disable something in order for this reset-function to correctly execute.
- Channel reset is asserted for 8 clock cycles. The PEND bit in the COR register is cleared when complete.
- Could of course lead to data-corruption coming/going from/to the reset channel.
- Resets the complete channel (all logic, buffers & register values)
- Except the configuration registers which remains their value.
- This re-initialisation/reset may cause spurious interrupts as Transfer Sublayer and FARM-1 update CLCW and FAR.

[5] The PSS bit usage is only supported if the gPSS generic is set on the TCC module.

153.11.15 Interrupt registers

The interrupt registers give complete freedom to the software, by providing means to mask interrupts, read interrupts and clear interrupts.

When an interrupt occurs the corresponding bit in the Interrupt Status Register is set. The normal sequence to initialize and handle a module interrupt is:

- Set up the software interrupt-handler to accept an interrupt from the module.
- Initialize the Interrupt Mask Register, unmasking each bit that should generate the module interrupt.
- Write the Interrupt Status Register to clear any spurious interrupts.
- When an interrupt occurs, read the Pending Interrupt Status Register in the software interrupt-handler to determine the causes of the interrupt.
- Handle the interrupt, taking into account all causes of the interrupt.
- Clear the handled interrupt writing the corresponding bit in the using Pending Interrupt Status Register.

Masking interrupts: After reset, all interrupt bits are masked, since the Interrupt Mask Register is zero. To enable generation of a module interrupt for an interrupt bit, set the corresponding bit in the Interrupt Mask Register.

Clearing interrupts: Selected bits can be cleared by writing ones to the bits that shall be cleared in the Interrupt Status Register.

Reading interrupt status: Reading the Interrupt Status Register. Note bits in the Interrupt Status Register are set when the corresponding to events occur, independent of Interrupt Mask Register configuration. Events generated in GRTC-COP can still be detected and cleared using the Interrupt Status Register although interrupt generation is disabled.

The interrupt registers comprise the following:

Table 2605. Interrupt registers

Description	Name	Mode
Interrupt Status Register	PISR	wc
Interrupt Mask Register	IMR	rw

*Table 2606.*Interrupt registers template

31 4	3	2	1	0
-	CLCW	FAR	BLO	RFA
	0	0	0	0
	*	‘	*	‘

3: CLCW CLCW updated by Transfer Sublayer*
2: FAR New FAR available*
1: BLO Bit Lock changed*
0: RFA RF Available Changed*

*See table 2605.

153.12 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x0A5. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

153.13 Configuration options

Table 2607 shows the configuration options of the core (VHDL generics).

*Table 2607.*Configuration options

Generic	Function	Description	Allowed range	Default
GRLIB AMBA plug&play settings				
hmstndx	AHB master index		Integer	0
pindex	APB slave index		0 - NAPBSLV-1	0
paddress	Address	Field of the APB bar.	0 - 16#FFF#	0
pmask	Mask	Field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	APB slave interrupt		Integer	0
singleirq	Single interrupt		Integer	0
inputmask	Maskable input		Integer	0
syncrst	Synchronous reset		0 - 1	0
gSystemClock	System frequency	[Hz]	Integer	33333333
gHardware	Hardware commands	Coding and Transfer sublayer fixed configuration & enable	0 - 1	0
Features settings				
gIn	Number of channels	Number of TC channels	1 - 8	3
gPSS	PSS support enable	Enables PSS support	0 - 1	1
gTimeoutMask	Time out mask	Enables masking of input on time out	0 - 1	0
gTimeout	Time out period CL	2^n clock periods		24
gRFAvailable	Number of RF inputs	Minimum 1	1 - 8	3
gBitLock	Number of TC inputs	Minimum 1	1 - 8	3
gFifoABits	FIFO depth	2^n words	2 - 10	2
gTimeoutTL	Time out period TL	TL Backend timeout. 2^n clock periods		24
gNumMAP	MAP Demultiplex	1=Disable, Num MAP targets	1 - 63	1
gMapImpl	MAP I/F impl	0=linear assignment, 1=consume seg. header on selected MAP interfaces, 2=user MAP channel to I/F mapping.	0 - 2	0
Asynchronous bit serial interface settings (CLCW interface)				
gClcw	Enable		0 - 1	1
gBaud	Baud rate	[Baud]	Integer	115200
gOddParity	Odd parity	Odd parity generated, but not checked	0 - 1	0
gTwoStopBits	Number of stop bits	0=one stop bit, 1=two stop bits	0 - 1	0

153.14 Signal descriptions

Table 2608 shows the interface signals of the core (VHDL ports).

*Table 2608.*Signal descriptions

Signal name	Field	Type	Function	Description	Active
HRESETn	N/A	Input	Reset		Low
HCLK	N/A	Input	Clock		-
TCIN	tcactive	Input	Active	Indicate that sub-carrier lock is achieved (or bit lock). Enable for the clock and data.	-
	tcclk		Bit clock	Serve as the serial data input bit clocks.	-
	tcdata		Data	Serve as the serial data input. Data are sampled on the TCCLK clock edges when the corresponding TCActive input is asserted.	-
	tcc_high		Active high setting	1=active high delimiter, 0=active low	-
	tcc_rise		Rising clock edge	1=rising, 0=falling	-
	pseudo		Pseudo-Derando-miser	1=enabled, 0=disabled	-
	mark		NRZ-M decoder	1=NRZ-M, 0=NRZ-L	-
	pss		PSS/ECSS mode	1=ESA PSS, 256 octet, fill bit augment 0=ECSS, 1024 octet, no fill bit augment	-
	rfavailpos		RF Available polarity	Active high=1 / low=0, used for CLCW	-
	bitlockpos		Bit Lock polarity	Active high=1 / low=0, used for CLCW	-
	clcwrfavailable		RF Available	Used for CLCW	-
	clcwbitlock		Bit Lock	Used for CLCW	-
	scid		Spacecraft Identity	0 is MSB, 9 is LSB	-
	framefecf		TC FECF	Enable FECF and CRC check	
	vcid		Virtual Channel Id	0 is MSB, 5 is LSB	
	tct_bufaddr		FARM Buffer location	Transfer Sublayer buffer storage AMBA addr. 31 is MSB, 11 is LSB	
	poswin		Positive Window	FARM-1 Positive Window Area	
	negwin		Negative Window	FARM-1 Negative Window Area	
	map_route		MAP Channel to I/F table	64 MAP Channels to upto 64 MAP HW interfaces, when gMapImpl=2	
	drop_seghead		Drop seg. header	Drop segment header for selected MAP HW I/F, when gMapImpl>0	
TCOUT	clcw1data	Output	CLCW Data	Bit serial asynchronous data for CLCW 1 interface.	-
	clcw1		CLCW Data	32 bit parallel CLCW	-
	clcwevent(1)		CLCW Event	New CLCW on parallel CLCW1	-

Table 2608. Signal descriptions

Signal name	Field	Type	Function	Description	Active
Octet Channel Request Interface(s). Multiple interfaces when MAP Demultiplex function enabled.					
OCRI	begins	Output	Data unit start	Pulse	High
	progresses		In progress	High throughout transfer	High
	completes		End of data	Pulse	High
	abort		Abort data unit	Pulse	High
	request		Data valid strobe	Pulse	High
	data		8-bit data		-
OCRO		Input	Octet Channel Response Interface(s)		
	accepting		Ready for data unit	Level	High
	busy		Not ready for octet	Level	High
APBI	*	Input	APB slave input signals		-
APBO	*	Output	APB slave output signals		-
	PIRQ(*pirq*+3)		Interrupts (If *singleirq*=1 only one common interrupt will be generate on *pirq*.)	CLCW Event	Location on PIRQ bus depends on *pirq* generic.
	PIRQ(*pirq*+2)			FAR available	
	PIRQ(*pirq*+1)			Bit Lock changed	
	PIRQ(*pirq*+0)			RF Available changed	
AHBMIN	*	Input	AMB master input signals		-
AHBMOUT	*	Output	AHB master output signals		-

* see GRLIB IP Library User's Manual

153.15 Signal definitions and reset values

The signals and their reset values are described in table 2609.

*Table 2609.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
tcc_high	Input, static	Active high setting	-	-
tcc_rise	Input, static	Rising clock edge	-	-
pseudo	Input, static	Pseudo-Derandomiser	-	-
mark	Input, static	NRZ-M decoder	-	-
pss	Input, static	PSS/ECSS mode	-	-
rfavailpos	Input, static	RF Available polarity	-	-
bitlockpos	Input, static	Bit Lock polarity	-	-
tcactive	Input, async	Active	-	-
tcclk	Input, async	Bit clock	-	-
tcdata	Input, async	Data	-	-
rfavailable	Input, async	RF Available for CLCW	-	-
bitlock	Input, async	Bit Lock for CLCW	-	-
scid	Input, static	Spacecraft Identity	-	-
vcid	Input, static	Virtual Channel Identity	-	-
framefecf	Input, static	Enable FECF and CRC check	-	-
tct_bufaddr	Input, static	Transfer Sublayer buffer address	-	-
poswin	Input, static	FARM-1 Positive Window Area	-	-
negwin	Input, static	FARM-1 Negative Window Area	-	-
clcw1data	Output	CLCW output data	-	Logical 1
clcwevent	Output	CLCW Event	-	Logical 0
OCRI.begins	Output	Data unit start	High	Logical 0
OCRI.progresses	Output	In progress	High	Logical 0
OCRI.completes	Output	End of data	High	Logical 0
OCRI.abort	Output	Abort data unit (Logical 0 after end of reset)	High	Logical 1
OCRI.request	Output	Data valid strobe	High	Logical 0

153.16 Library dependencies

Table 2610 shows libraries used when instantiating the core (VHDL libraries).

*Table 2610.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_Types	Signals, component	Signals and component declaration
TMTC	TCCOP_Pkg	Signals, component	Internal TC signal definitions

153.17 Instantiation

This example shows how the core can be instantiated.

```
library  IEEE;
   use      IEEE.Std_Logic_1164.all;
   library  GRLIB;
   use      GRLIB.AMBA.all;
   library  TMTC;
   use      TMTC.TMTC_Types.all;

...

component grtccop is
   generic(
      hmstndx:       in    integer                 := 0;
      pindex:        in    integer                 := 0;
      paddr:         in    integer                 := 0;
      pmask:         in    integer                 := 16#fff#;
      pirq:          in    integer                 := 0;
      singleirq:     in    integer                 := 0;
      inputmask:     in    integer                 := 0;
      syncrst:       in    integer                 := 0;  -- synchronous reset
      gSystemClock:  in    integer := 333333333;          -- Hz
      gHardware:     in    integer range 0 to  1   := 0;  -- enable hardware cmd

      gIn:           in    integer range 1 to  8   := 3;  -- number of inputs
      gPSS:          in    integer range 0 to  1   := 1;  -- enable PSS support
      gTimeoutMask:  in    integer range 0 to  1   := 0;  -- timeout mask
      gTimeout:      in    integer                 := 24; -- Coding timeout 2^n
      gRFAvailable:  in    integer range 1 to  8   := 3;  -- No. of RF inputs
      gBitLock:      in    integer range 1 to  8   := 3;  -- No. of BL inputs

      gFifoABits:    in    integer range 2 to 10   := 2;  -- words (2^x)
      gTimeoutTL:    in    integer                 := 24; -- Transfer timeout 2^n
      gNumMap:       in    integer range 1 to 64   := 1;  -- number of HW MAP
      gMapImpl:      in    integer range 0 to  2   := 0;  -- 0=linear, 2=user

      gClcw:         in    integer := 1;                  -- CLCW UART
      gBaud:         in    integer := 115200;             -- Baud rate setting
      gOddParity:    in    integer := 0;                  -- Odd parity
      gTwoStopBits:  in    integer := 0                   -- Stop bit selection
   );
   port(
      -- AMBA AHB system signals
      hclk:          in    std_ulogic;                    -- System clock
      hresetn:       in    std_ulogic;                    -- Synchronised reset

      -- AMBA APB slave Interface
      apbi:          in    apb_slv_in_type;
      apbo:          out   apb_slv_out_type;

      -- AMBA AHB master Interface
      ahbmi:         in    ahb_mst_in_type;               -- AHB master input
      ahbmo:         out   ahb_mst_out_type;              -- AHB master output

      -- Telecommand interfaces
      tcin:          in    grtc_in_type;
      tcout:         out   grtc_out_type;

      -- Octet Channel Interface
      ocri:          out   oci_request_vector( 0 to gNumMap-1);
      ocro:          in    oci_response_vector(0 to gNumMap-1)

   );
   end component grtccop;
```

154 GRTC_HW - CCSDS/ECSS Telecommand Decoder - Hardware Commands

154.1 Overview

154.1.1 Concept

The Telecommand Decoder - Hardware Commands provides access to an output port via telecommands.

The decoder implements the following layers:

- Application Layer:
  - Hardware command decoding and execution
- Space Packet Protocol:
  - Packet Extraction
  - Path Recovery
- Data Link - Protocol Sub-Layer:
  - Virtual Channel Packet Extraction
  - Virtual Channel Reception:
    - Support for Command Link Control Word (CLCW)
  - Virtual Channel Demultiplexing
  - Master Channel Demultiplexing

  •All Frames Reception
- Data Link - Synchronization and Channel Coding Sub-Layer:
  - Pseudo-Derandomization
  - BCH codeblock decoding
  - Start Sequence Search
- Physical Layer:
  - Non-Return-to-Zero Level/Mark de-modulation (NRZ-L/M)

The Channel Coding Sub-Layer and the Physical Layer are shared with the Telecommand Decoder and are therefore not repeated here.

154.2 Operation

In the Application Layer and the Data Link - Protocol Sub-Layer, the information octets from the Channel Coding Sub-Layer are decoded as follows.

154.2.1 All Frames Reception

The All Frames Reception function performs two procedures:

- Frame Delimiting and Fill Removal Procedure; and
- Frame Validation Check Procedure, in this order.

The Frame Delimiting and Fill Removal Procedure is used to reconstitute Transfer Frames from the data stream provided by the Channel Coding Sub-Layer and to remove any Fill Data transferred from the Channel Coding Sub-Layer. The Frame Length field is checked to correspond to a fixed value as listed below. The number of information octets is checked to be a fixed number 21.

The Fill Data is checked to match the 0x55 pattern, or the corresponding pseudo-randomized pattern when pseudo-derandomization is enabled (pin configurable). Note that it is assumed that the Fill Data is not pseudo-randomized at the transmitting end.

The Frame Validation Checks procedure performs the following checks:

- Version Number is checked to be 0
- Bypass Flag is checked to be 1
- Control Command Flag is checked to be 0
- Reserved Spare bits are checked to be 0
- Spacecraft Identifier is compared with a pin configurable input value
- Virtual Channel identifier is compared with a pin configurable input value
- Frame Length field is checked to be a fixed value of 0000010011_b (i.e. 20-1)
- Frame Sequence Number is checked to be a fixed value of 0
- The Frame Error Control Field is checked to match the recomputed CRC value

154.2.2 Master Channel Demultiplexing

The Master Channel Demultiplexing is performed implicitly during the All Frames Reception procedure described above.

154.2.3 Virtual Channel Demultiplexing

The Virtual Channel Demultiplexing is performed implicitly during the All Frames Reception procedure described above.

154.2.4 Virtual Channel Reception

The Virtual Channel Reception supports Command Link Control Word (CLCW) generation and transfer to the Telemetry Encoder, according to the following field description.

- Control Word Type field is 0
- CLCW Version Number field is 0
- Status Field is 0
- COP in Effect field is 1
- Virtual Channel Identification is taken from pin configurable input value
- Reserved Spare field is 0
- No RF Available Flag is 0, but is overwritten by the Telemetry Encoder
- No Bit Lock Flag is 0, but is overwritten by the Telemetry Encoder
- Lockout Flag is 1
- Wait Flag is 0
- Retransmit Flag is 0
- FARM-B Counter is taken from the to least significant bits of a reception counter
- Reserved Spare field is 0
- Report Value field is 0

154.2.5 Virtual Channel Packet Extraction

The Virtual Channel Packet Extraction function extracts the Space Packet from the Frame Data Unit on the Virtual Channel, received from the Virtual Channel Reception function.

No blocking of Space Packets is permitted.

The Packet Version Number is checked to be 000, before delivered to the next function, else the Space Packet is discarded.

154.2.6 Path Recovery

The Path Recovery function receives and demultiplexes Space Packets received from the underlying subnetwork. The Path Recovery function receives Space Packets from the underlying subnetwork and demultiplex, if necessary, the received Space Packets on the basis of the Path Identifier of each Space Packet.

The Path Identifier is derived directly from the Application Identifier (APID) of the Space Packet, which is checked to be 00000000000, else the Space Packet is discarded.

Since the application layer uses the Octet String Service, the received Space Packets are delivered to the user through the Packet Extraction function described hereafter.

154.2.7 Packet Extraction

The Packet Extraction function extracts service data units from Space Packets. The Packet Extraction function extracts Octet Strings by stripping the Packet Primary Header, and the Packet Error Control field.

The following checks are performed before the Octet Strings (i.e. User Data Field) is forwarded to the Application Layer, else it is discarded:

- Packet Version field is 000_b
- Packet Type is 1_b
- Secondary Header Flagis 0_b
- Application Process identifier is compared with a pin configurable input value
- Sequence Flagsare 11_b
- Packet Data Length is 0000000000000110_b

The Packet Extraction Function does not check the continuity of the Packet Sequence Count, since Packet Name is used.

The Packet Extraction function verifies the correctness of the Packet Error Control field to match the recomputed CRC value (calculated over the complete Space Packet), if incorrect the Space Packet is not forwarded to the Application Layer, being discarded.

154.2.8 Application Layer

The Application Layer interprets only Transfer Frames that have successfully passed the Data Link Layer and Space Packet Protocol checks as described above.

The Application Layer interprets the Octet Strings (i.e. User Data Field) of the Packet Data Field.

The User Data Field consists of 5 octets comprising the hardware command, as defined in the bit order hereafter (MSB of first octet corresponds to OUTPUT(0), LSB of last octet corresponds to PULSE(7):

- OUTPUT(0 to 31) (32 bits in total)
- PULSE(0 to 7) (8 bits in total)

Before a hardware command is executed, the following is checked that:

- no hardware command is ongoing

The OUTPUT bits 0 to 31 correspond to the tcgpio[] bits 0 to 31.

The PULSE field has three interpretations: 0 to clear bits, 255 to set bits, 1 to 254 to generate pulses on bits:

- When PULSE field is 0, the corresponding bits that are not set in the OUTPUT field are cleared on the tcgpio outputs (AND function).
- When PULSE field is 255, the corresponding bits that are set in the OUTPUT field are set on the tcgpio outputs (OR function).
- When PULSE field is in the range 1 to 254, the corresponding bits that are set in the OUTPUT field are set on the tcgpio outputs for a duration of PULSE * 8192 system clock cycles, after which they are cleared again.

The tcgpio[0:31] outputs are cleared to logical 0 at reset.

154.3 Telecommand Transfer Frame format - Hardware Commands

The telecommand Transfer Frame for Hardware Commands has the following structures.

Transfer Frame				
Transfer Frame Primary Header	Transfer Frame Data Field			Frame Error Control Field (FECF)
	Space Packet			
	Packet Primary Header	Packet Data Field		
		User Data Field	Packet Error Control	
		Hardware Command		
0:39	40:87	88:127	128:143	144:159
5 octets	6 octets	5 octets	2 octets	2 octets
20 octets				

Table 2611. Telecommand Transfer Frame format

Transfer Frame Primary Header							
Version	Bypass Flag	Control Command Flag	Reserved Spare	S/C Id	Virtual Channel Id	Frame Length	Frame Sequence Number
00_b	1_b	0_b	00_b	PIN	PIN	0000010011_b	00000000_b
0:1	2	3	4:5	6:15	16:21	22:31	32:39
2 bits	1 bit	1 bit	2 bits	10 bits	6 bits	10 bits	8 bits
2 octets					2 octets		1 octet

Table 2612. Telecommand Transfer Frame Primary Header format

Space Packet								
Packet Primary Header							Packet Data Field	
Packet Version Number	Packet Identification			Packet Sequence Control		Packet Data Length	User Data Field	Packet Error Control
	Type	Secondary Header Flag	Application Process Id	Sequence Flags	Packet Name			
000_b	1_b	0_b	PIN	11_b	Don't care	0006_{16}	Hardware Command	CRC
40:42	43	44	45:55	56:57	58:71	72:87	88:127	128:143
3 bits	1 bit	1 bit	11 bits	2 bits	14 bits	16 bits	40 bits	16 bits
6 octets							5 octets	2 octets

Table 2613. CCSDS Space Packet format

Hardware Command	
OUTPUT (0:31)	PULSE (0:7)
88:119	120:127
32 bits	8 bits
4 octets	1 octet

Table 2614. Hardware Command format

154.4 Registers

The core has not user accessible registers.

154.5 Vendor and device identifiers

The core has has neither a vendor identifier nor a device identifier.

154.6 Configuration options

The core has no configuration options.

154.7 Signal descriptions

Table 2615 shows the interface signals of the core (VHDL ports).

*Table 2615.*Signal descriptions

Signal name	Field	Type	Function	Description	Active
RSTN	N/A	Input	Reset		Low
CLK	N/A	Input	Clock		-
TCI	SCID	Input	Spacecraft Identity	0 is MSB, 9 is LSB	-
	PSEUDO		Pseudo-Derando-miser	1=enabled, 0=disabled	-
	RFAVAILPOS		RF Available polarity	Active high=1 / low=0, used for CLCW	-
	BITLOCKPOS		Bit Lock polarity	Active high=1 / low=0, used for CLCW	-
	CLCWRFAVAIL-ABLE		RF Available	Used for CLCW	-
	CLCWBITLOCK		Bit Lock	Used for CLCW	-
TCO	CL_*	Input	Coding Layer	Coding Layer interface	-
TCVCID	N/A	Input	Virtual Channel Identifier		-
TCAPID	N/A	Input	Application Identifier		-
TCGPIO	N/A	Output	Hardware command		-
CLCW-DATA	N/A	Output	CLCW parallel data		-
CLCWEV-ENT	N/A	Output	CLCW changed		High

154.8 Signal definitions and reset values

The signals and their reset values are described in table 2616.

*Table 2616.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
tcgpio[0:31]	Output	Hardware command output	Logical 1	Logical 0

154.9 Timing

The timing waveforms and timing parameters are shown in figure 381 and are defined in table 2617.

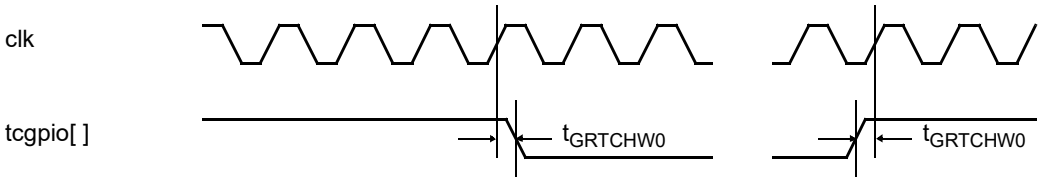


Figure 381. Timing waveforms

*Table 2617.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{GRTCHW0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns

Note: The inputs are re-synchronized internally. The signals do not have to meet any setup or hold requirements. Static signals should not change between resets.

154.10 Library dependencies

Table 2618 shows libraries used when instantiating the core (VHDL libraries).

*Table 2618.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	StdLib	Functions	Operators
TMTC	TMTC_Types	Signals, component	Signals and component declaration

155 GRTC_UART - CCSDS/ECSS Telecommand Decoder - UART

155.1 Overview

A UART is used for the transmission of the CLTU from the telecommand decoder to the suer. The protocol has a fixed to baud rate, 1 star bit, 8 data bits, optional odd parity, 1 or 2 stop bits, with a BREAK command for message delimiting (sending 13 bits of logical zero).

The output contains the corrected information octets, which comprises the Telecommand Transfer Frame and any filler data octets. This is followed by the transmission of a BREAK command.

Note: The Telecommand Decoder Coding Layer does not inspect the contents of the corrected information octets. All corrected information octets are output on the UART interface.

155.2 Asynchronous bit serial data format

The asynchronous bit serial interface complies to the data format defined in [EIA 232]. It also complies to the data format and waveform shown in table 2619 and figure 382. The interface is independent of the transmitted data contents. Positive logic is considered for the data bits. The number of stop bits can optionally be either one or two. The parity bit can be optionally included.

Asynchronous bit serial format	start	D0	D1	D2	D3	D4	D5	D6	D7	parity	stop	stop
	first	*lsb*							*msb*			*last*
General data format $i = \{0, n\}$		8*i+7	8*i+6	8*i+5	8*i+4	8*i+3	8*i+2	8*i+1	8*i			
		last							*first*			

Table 2619. Asynchronous bit serial data format

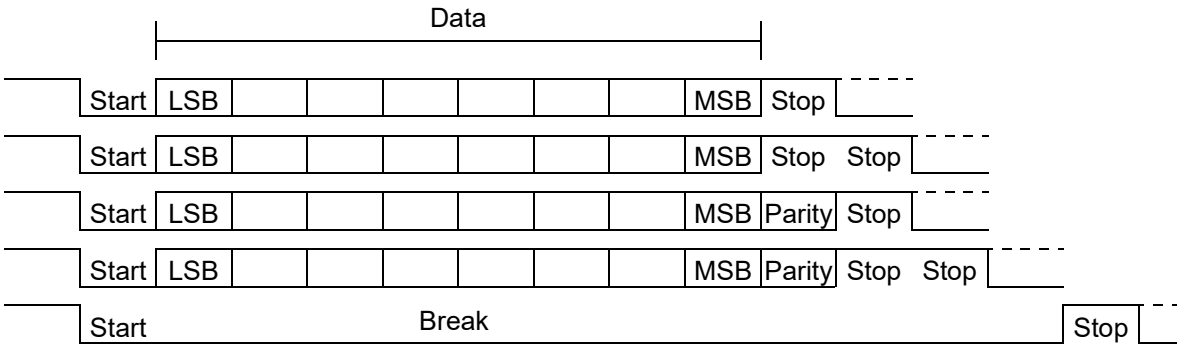


Figure 382. Asynchronous bit serial protocol / waveform

155.3 Registers

The core has not user accessible registers.

155.4 Vendor and device identifiers

The core has has neither a vendor identifier nor a device identifier.

155.5 Configuration options

Table 2620 shows the configuration options of the core (VHDL generics).

*Table 2620.*Configuration options

Generic	Function	Description	Allowed range	Default
gSystemClock	System frequency	[Hz]	Integer	33333333
gBaud	Baud rate	[Baud]	Integer	115200
syncreset	Sync reset	Synchronous reset when 1	0 - 1	0

155.6 Signal descriptions

Table 2621 shows the interface signals of the core (VHDL ports).

*Table 2621.*Signal descriptions

Signal name	Field	Type	Function	Description	Active
RSTN	N/A	Input	Reset		Low
CLK	N/A	Input	Clock		-
ODDPARITY	N/A	Input		Generate odd parity, else none	High
TWOSTOPBITS	N/A	Input		Generate two stop bits, else one	High
TCO	CL_*	Input	Coding Layer	Coding Layer interface	-
TCUART	N/A	Output		Telecommand UART output	-

155.7 Signal definitions and reset values

The signals and their reset values are described in table 2622.

*Table 2622.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
tcuart	Output	CLTU UART output	-	Logical 1

155.8 Timing

The timing waveforms and timing parameters are shown in figure 383 and are defined in table 2623.

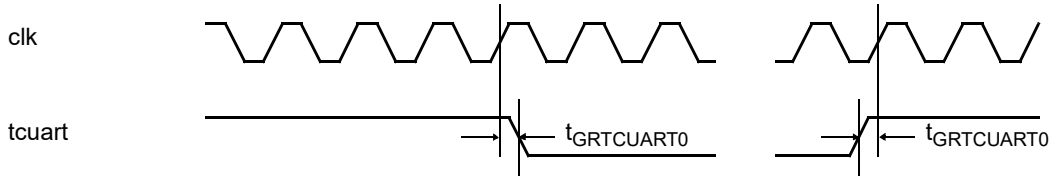


Figure 383. Timing waveforms

*Table 2623.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{GRTCUART0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns

Note: The inputs are re-synchronized internally. The signals do not have to meet any setup or hold requirements. Static signals should not change between resets.

155.9 Library dependencies

Table 2624 shows libraries used when instantiating the core (VHDL libraries).

*Table 2624.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	StdLib	Functions	Operators
TMTC	TMTC_Types	Signals, component	Signals and component declaration

156 GRTCTX - CCSDS/ECSS Telecommand Transmitter

156.1 Overview

The CCSDS/ECSS/PSS Telecommand Transmitter implements part of the Data Link Layer, covering the Protocol Sub-layer and the Frame Synchronization and Coding Sub-layer and part of the Physical Layer of the packet telecommand protocol.

The operation of the Telecommand Transmitter is highly programmable by means of control registers.

The Telecommand Transmitter comprises several encoders and modulators implementing the Consultative Committee for Space Data Systems (CCSDS) recommendations, European Cooperation on Space Standardization (ECSS) and the European Space Agency (ESA) Procedures, Standards and Specifications (PSS) for telecommand and channel coding. The Telecommand Transmitter comprises the following:

- Frame Error Control Field (FECF)
- Pseudo-Randomiser (PSR) / Bit Transition Generator (BTG)
- Communications Link Transmission Unit (CLTU)
  - Start Sequence insertion
  - Bose-Chaudhuri-Hocquenghem (BCH)
  - Tail Sequence insertion
- Physical Layer Operations Procedures (PLOP-1, PLOP-2)
- Non-Return-to-Zero Mark/Level (NRZ)

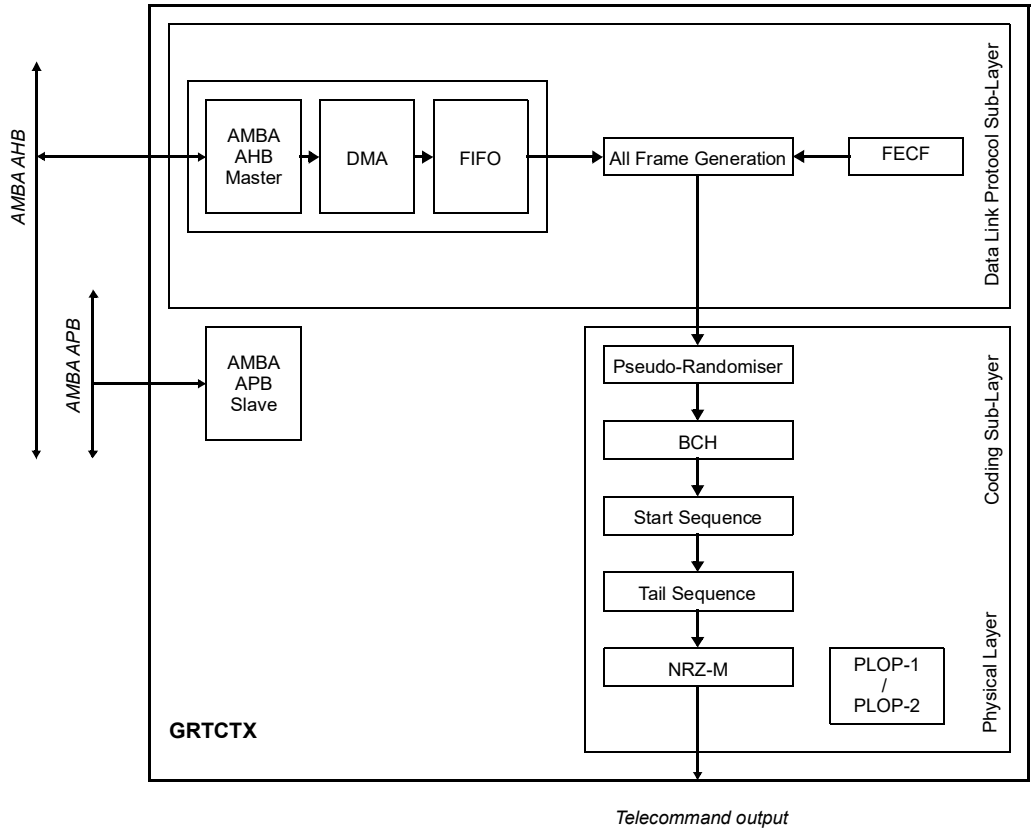


Figure 384. Block diagram

156.2 References

156.2.1 Documents

[C231] CCSDS 231.0-B-2 TC Synchronization and Channel Coding

[C232] CCSDS 232.0-B-2 TC Space Data Link Protocol

[ECSS04] ECSS-E-50-04C: Space engineering - Space data links - Telecommand protocols, synchronization and channel coding

[ECSS05] ECSS-E-50-05C: Space engineering - Radio frequency and modulation

[PPS107] PSS-04-107: Packet telecommand standard

[PPS105] PSS-04-105 Radio frequency and modulation standard

156.2.2 Acronyms and abbreviations

BTG Bit Transition Generator

CCSDS Consultative Committee for Space Data Systems

CLCW Command Link Control Word

CLTU Communications Link Transmission Unit

CMM Carrier Modulation Mode

CRC Cyclic Redundancy Code

DMA Direct Memory Access

ECSS European Cooperation for Space Standardization

ESA European Space Agency

FECF Frame Error Control Field

MSB Most Significant Bit

NRZ Non Return to Zero

OCF Operational Control Field

PLOP Physical Layer Operations Procedure

PSR Pseudo Randomiser

PSS Procedures, Standards and Specifications

TC Telecommand -- BCH Bose-Chaudhuri-Hocquenghem

156.3 Layers

156.3.1 Introduction

The relationship between Packet Telecommand (or simply Telecommand or TC) standard and the Open Systems Interconnection (OSI) reference model is such that the OSI Data Link Layer corresponds to two separate layer, namely the Data Link Protocol Sub-layer and Synchronization and Channel Coding Sub-Layer. The OSI Data Link Layer is covered here.

The OSI Physical Layer is also covered here to some extended, as specified in [ECSS04] and [PPS107]. The OSI Network Layer or higher layers are not covered here.

156.3.2 Data Link Protocol Sub-layer

The following functionality is implemented in the core:

- All Frame Generation:

•Frame Error Control Field (FECF) calculation and insertion

156.3.3 Synchronization and Channel Coding Sub-Layer

The following functionality is implemented in the core:

- Pseudo-Randomiser (PSR) / Bit Transition Generator (BTG)
- Communications Link Transmission Unit (CLTU)
  - Start Sequence insertion
  - Bose-Chaudhuri-Hocquenghem (BCH)
  - Tail Sequence insertion

156.3.4 Physical Layer

The following functionality is implemented in the core:

- Physical Layer Operations Procedures (PLOP-1, PLOP-2)
- Non-Return-to-Zero (NRZ) modulation

156.4 Operation

156.4.1 Introduction

The DMA interface provides a means for the user to send blocks of data of arbitrary length, normally this is Communications Link Transmission Units (CLTU).

156.4.2 Descriptor setup

The DMA interface is used for sending data on the uplink. The transmission is done using descriptors located in memory. A single descriptor is shown in tables 2625 through 2628. The address field of the descriptor should point to the start of the data to be sent. The address need not be word-aligned. If the interrupt enable (IE) bit is set, an interrupt will be generated when the transfer has completed (this requires that the interrupt enable bit in the control register is also set). The interrupt will be generated regardless of whether the transfer was successful or not. The wrap (WR) bit is also a control bit that should be set before transmission and it will be explained later in this section.

Table 2625. GRTCTX descriptor word 0 (address offset 0x0)

31 16	15 8	7	6 4	3	2	1	0
LEN	RESERVED	UR	SEL	-	WR	IE	EN

31: 16	(LEN) - Length in bytes (note that maxium length is limited to 2048 bytes, and minimum to 2 bytes)
15: 8	RESERVED
7:	Underrun (UR) - Underrun detected during transmission.
6: 4	Select output (SEL) - Select output for bit clock, bit lock and bit data (0 to 7)
3:	RESERVED
2:	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 16. The pointer automatically wraps to zero when the 16 kB boundary of the descriptor table is reached.
1:	Interrupt Enable (IE) - an interrupt will be generated when the data for this descriptor has been sent provided that the transmitter interrupt enable bit in the control register is set. The interrupt is generated regardless if it was transferred successfully or if it terminated with an error.
0:	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.

Table 2626. GRTCTX descriptor word 1 (address offset 0x4)

31	0
ADDRESS	

31: 0 Address (ADDRESS) - Pointer to the buffer area to where data will be fetched.

Table 2627. GRTCTX descriptor word 2 (address offset 0x8)

31	0
STATUS0	

31: 0 (STATUS0) - External status information

Table 2628. GRTCTX descriptor word 3 (address offset 0xC)

31	0
STATUS1	

31: 0 (STATUS1) - External status information

To enable a descriptor the enable (EN) bit should be set and after this is done, the descriptor should not be touched until the enable bit has been cleared by the core.

156.4.3 Starting transmission

Enabling a descriptor is not enough to start transmission. A pointer to the memory area holding the descriptors must first be set in the core. This is done in the descriptor pointer register. The address must be aligned to a 16 kByte boundary. Bits 31 to 14 hold the base address of descriptor area while bits 13 to 4 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the core, the pointer field is incremented by 16 to point at the next descriptor. The pointer will automatically wrap back to zero when the next 16 kByte boundary has been reached. The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 16 kByte boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when transmission is active.

The final step to activate the transmission is to set the enable bit in the DMA control register. This tells the core that there are more active descriptors in the descriptor table. This bit should always be set when new descriptors are enabled, even if transmission is already active. The descriptors must always be enabled before the transmission enable bit is set.

156.4.4 Descriptor handling after transmission

When the transmission has finished, status is written to the first word in the corresponding descriptor. The other bits in the first descriptor word are set to zero after transmission while the second word is left untouched. The enable bit should be used as the indicator when a descriptor can be used again, which is when it has been cleared by the core. Additionally, the last two words in the corresponding descriptor are written with external status information.

There are multiple bits in the DMA status register that hold status information.

The Transmitter Interrupt (TI) bit is set each time a DMA transmission ended successfully. The Transmitter Error (TE) bit is set each time an DMA transmission ended with an error. For either event, an interrupt is generated for descriptor for which the Interrupt Enable (IE) was set. The interrupt is maskable with the Interrupt Enable (IE) bit in the control register.

The Transmitter AMBA error (TA) bit is set when an AMBA AHB error was encountered either when reading a descriptor or when writing data. Any active transmission was aborted and the DMA channel was disabled. It is recommended that the Telecommand Transmitter is reset after an AMBA AHB error. The interrupt is maskable with the Interrupt Enable (IE) bit in the control register.

156.5 Registers

The core is programmed through registers mapped into APB address space.

*Table 2629.*GRTCTX registers

APB address offset	Register
0x00	GRTCTX DMA Control register
0x04	GRTCTX DMA Status register
0x08	GRTCTX DMA Descriptor Pointer register
0x80	GRTCTX Control register
0x88	GRTCTX Configuration register
0x90	GRTCTX Physical Layer register
0x94	GRTCTX Coding Sub-Layer register
0x98	GRTCTX Start and Tail register
0x9C	GRTCTX All Frames register

156.5.1 GRTCTXDMA Control Register

*Table 2630.*0x00 -DCR - DMA control register

31 2	1	0
RESERVED	IE	EN
0	0	0
r	rw	rw

31: 2 RESERVED

1: Interrupt Enable (IE) - enable interrupts TA, TI, and TE

0: Enable (EN) - enable DMA transfers

156.5.2 GRTCTX DMA Status Register

*Table 2631.*0x04 - DSR - DMA status register

31 4	3	2	1	0
RESERVED	ACTIVE	TA	TI	TE
0	NR	0	0	0
r	r	wc	wc	wc

31: 4 RESERVED

3: Active (ACTIVE) - DMA access ongoing

2: Transmitter AMBA Error (TA) - DMA AMBA AHB error, cleared by writing a logical 1

1: Transmitter Interrupt (TI) - DMA interrupt, cleared by writing a logical 1

0: Transmitter Error (TE) - DMA transmitter error, cleared by writing a logical 1

156.5.3 GRTCTXDMA Descriptor Pointer Register

Table 2632. 0x08 - DDP - DMA descriptor pointer register

31 14	13 4	3 0
BASE	INDEX	RESERVED
NR	NR	0
rw	rw	r

31: 14 Descriptor base (BASE) - base address of descriptor table

13: 4 Descriptor index (INDEX) - index of active descriptor in descriptor table

3: 0 Reserved - fixed to “0000”

156.5.4 GRTCTX Control Register

Table 2633. 0x80 - CTRL - control register

31 3	2	1	0
RESERVED	RST	R	TxEN
0	0	0	0
r	rw	r	rw

31: 3 RESERVED

2: Reset (RST) - resets complete core

1: RESERVED

0: Transmitter Enable (TxEN) - enables Telecommand Transmitter (should be done after the complete configuration of the Telecommand Transmitter)

156.5.5 GRTCTX Configuration Register

Table 2634. 0x88 - CONF - configuration register

31 24	23 8	7 0
REVISION	FIFOSIZE	RESERVED
*	*	0
r	r	r

31: 24 (REVISION) - Revision number

23: 8 (FIFOSIZE) - FIFO size in bytes

7: 0 RESERVED

156.5.6 GRTCTX Physical Layer Register

Table 2635. 0x90 - PLR - physical layer register

31 20	19 16	15 11	10	9	8	7	6	5	4	3 2	1	0
DIVIDE	UNMODU-LATED	RESERVED	IDLE ALL	IDLE PRE	IDLE POST	RF AVAIL	RF POS	BIT POS	CLK RISE	CLK MODE	PLOP	NRZM
1	0	0	0	0	0	0	1	1	1	0b01	0	0
rw	rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31: 20 (DIVIDE) - Clock divider (value 0 not used)

19: 16 (UNMODULATE) - Number of unmodulated octet counts (CMM-1)

15: 11 RESERVED

10: (IDLEALL) - Idle Sequence when unmodulate (CMM-1) or non-active, else all-zero output

9: (IDLEPRE) - Optional Idle Sequence before CLTU (CMM-4)

8: (IDLEPOST) - Optional Idle Sequence after CLTU (CMM-4)

7: (RFAVAIL) - RF available when unmodulate (CMM-1) or non-active

6: (RFPOS) - Positive polarity for RF available output signal

5: (BITPOS) - Positive polarity for bit lock output signal

4: Rising clock edge (CLKRISE) - Rising clock edge coinciding with serial data bit change

3: 2 Clock mode (MODE) - 00 = never generated,
01 = only generated when modulated,
10 = also generated when unmodulated (CMM-1),
11 = always generated

1: Physical Layer Operations Procedure (PLOP) - PLOP-1 (when cleared) or PLOP-2 (when set)

0: Non-Return-to-Zero-Mark (MARK) - non-return-to-zero - mark encoding enable

156.5.7 GRTCTX Coding Sub-layer Register

Table 2636. 0x94 - CSL - coding sub-layer register

31 24	23 16	15 8	7	6 5	4	3	2	1	0
RESERVED	IDLE	FILLDATA	FILLER BIT	R	PSEUDO ALL	PSEUDO	BCH	START SEQ	TAIL SEQ
0	0x55	0x55	0	0	0	0	0	0	0
r	rw	rw	rw	r	rw	rw	rw	rw	rw

31: 24 RESERVED

23: 16 (IDLE) - Idle/Acquisition Sequence (reset 0x55)

15: 8 (FILLDATA) - Fill Data for BCH coding (reset 0x55)

7: (FILLERBIT) - Filler Bit for BCH coding (reset 0)

6: 5 RESERVED

4: (PSEUDOALL) - Pseudo-Randomize BCH fill data (only with BCH and PSEUDO)

3: (PSEUDO) - Pseudo-Randomize (only with BCH)

2: (BCH) - BCH encoding

1: (STARTSEQ) - Start Sequence Generation

0: (TAILSEQ) - Tail Sequence Generation

156.5.8 GRTCTX Start And Tail Register

Table 2637. 0x98 - STR - start and tail register

31 16	15 8	7 0
START	TAIL	TAILLAST
0xEB90	0xC5	0x79
rw	rw	rw

31: 16 (START) - Start Sequence value (reset 0xEB90)

15: 8 (TAIL) - Tail Sequence value (reset 0xC5) (applies to first 7 octets:)

7: 0 (TAILLAST) - Tail Sequence, last (reset 0x79) (applies to the last octet only):

156.5.9 GRTCTX All Frames Register

Table 2638. 0x9C - AFR - all frames register

31 26	25 16	15 2	1	0
RESERVED	SCID	RESERVED	PSS	FECF
0	0	0	0	0
r	rw	r	rw	rw

31: 26 RESERVED

25: 16 (SCID) - Spacecraft Identifier (unsed)

15: 2 RESERVED

1: (PSS) - ESA PSS compatible mode (unsed)

0: (FECF) - Insert CRC/FECF, overwriting the two last octets of a transfer frame (only with BCH)

156.6 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x082. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

156.7 Implementation

156.7.1 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

156.8 Configuration options

Table 2639 shows the configuration options of the core (VHDL generics).

*Table 2639.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR	0 - 16#FFF#	0
pmask	MASK field of the APB BAR	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by core	0 - NAHBIRQ-1	0
memtech	Memory technology	0 to NTECH	0
clktech	Clock buffer technology	0 to NTECH	0
buftype	Clock buffer type	TBD	0

156.9 Signal descriptions

Table 2640 shows the interface signals of the core (VHDL ports).

*Table 2640.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
TCI	TCACTIVE[]	Output	Bit Lock	-
	TCCLK[]		Bit clock	-
	TCDATA[]		Bit data	-
	CLCW-RFAVAILABLE[]		RF Available	-
	CLCW-BITLOCK[]		Bit Lock	-
TMO	TIMESTAMP	Input	Time Stamp	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-

* see GRLIB IP Library User's Manual

156.10 Signal definitions and reset values

The signals and their reset values are described in table 2641.

*Table 2641.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
tcactive[]	Output	Bit lock	-	-
tcdata[]	Output	Serial bit data	-	-
tcclk[]	Output	Serial bit data clock	-	-
rfavailable[]	Output	RF available	-	-

156.11 Timing

The timing waveforms and timing parameters are shown in figure 385 and are defined in table 2642.

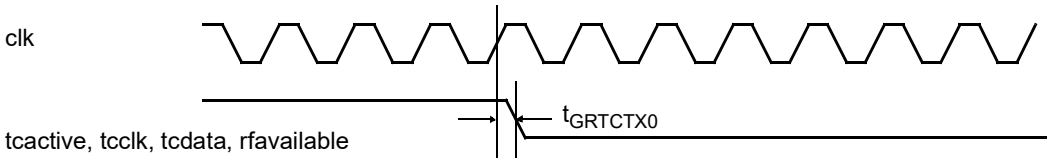


Figure 385. Timing waveforms

*Table 2642.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{GRTCTX0}$	clock to output	rising *clock* edge	TBD	TBD	ns

156.12 Library dependencies

Table 2643 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2643.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_TYPES	Signals, component	Component declaration

157 GRCTM - CCSDS Time Manager

157.1 Overview

The CCSDS Time Manager (GRCTM) provides basic time keeping functions such an Elapsed Time (ET) counter according to the Consultative Committee for Space Data Systems (CCSDS) Unsegmented Code specification, [CCSDS]. It comprises a Frequency Synthesizer (FS) by which a binary frequency is generated to drive the ET counter. The GRCTM provides support for setting the increment rate of the ET counter as well as of the FS counter.

The GRCTM provides datation services that sample the ET counter value on external events. It also provides generation of periodic pulses with cycle periods of less than one second. All services in the GRCTM core are accessible via an AMBA AHB slave interface.

The GRCTM provides a service for sampling the ET counter value on the occurrence of the time strobe generated by for example the Packet Telemetry Encoder (PTME), generating a Standard Spacecraft Time Source Packet according to the ESA Packet Telemetry Standard, [PSS]. The Time Source Packet can be read out via the AMBA AHB slave interface and is transmitted directly to a telemetry encoder via a serial interface.

The GRCTM can act as a master and/or a slave in a time distribution system. As a master only, the GRCTM distributes the ET to GRCTM slaves via a TimeWire (TW) interface. As a slave only, the GRCTM receives the ET via the TimeWire interface. When acting as a master and slave, the GRCTM receives the ET from a master GRCTM, but can also distribute the ET to other slaves.

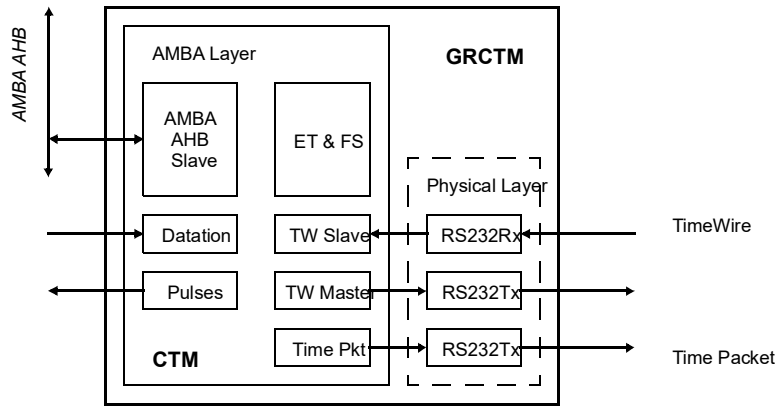


Figure 386. Block diagram

157.1.1 Foreseen usage of the core

On-board time maintenance and distribution is to be handled through a master CCSDS Time Manager (GRCTM) and one or more slave GRCTMs. Using a dedicated synchronisation line (TimeWire), the slave time manager can be synchronised with the master GRCTM. The slave GRCTM can further distribute the time to the payload. The GRCTM slaves will thus be slaved to the master GRCTM, but also act as masters for other modules. This isolates the master GRCTM from the payloads. The slave GRCTM provides four datation register into which the Elapsed Time (ET) counter can be latched on the occurrence of an external triggering event.

It is not possible to synchronise or set the ET counter in the master GRCTM. The ET counter can only be cleared by means of hardware or software reset.

157.1.2 Description of a general system using the core

The general approach to accurately maintaining on-board time is to have a central time reference measuring the elapsed time from an arbitrary epoch and to distribute regularly this time information to on-board applications by means of messages and synchronisation pulses. Another approach would be to have a centralised time system, where each application that needs to time stamp data could request the unit maintaining the central time reference to provide the relevant time information. Such an approach would have several inherent drawbacks, e.g. in systems with many users, the accuracy of a time stamp could be jeopardised due to long service latency and excessive bus traffic could degrade the overall performance of the data handling system.

The purpose of the GRCTM is to provide a building block for such time distribution services by providing the means for CCSDS compliant time keeping and a set of basic user time services. Most time distribution implementations have required support from the application processor to maintain synchronisation between the central and the local time references. Protocols and formats for distributing time information have differed between spacecraft and have sometimes only provided low resolution or poor accuracy. The purpose of the GRCTM is to provide an accurate time coherence throughout the spacecraft.

The correlation between the central time reference and ground has already been foreseen by providing a time strobe from the Packet Telemetry encoder. The time strobe has a deterministic relationship to the bit structure of the telemetry frame. This makes it possible to establish the time relation between the assertion of this time strobe on-board and the reception of the relevant frame on ground, taking into account the down link propagation delay. Each GRCTM instance maintains its own copy of the central elapsed time reference with which on-board applications can time stamp their data. This unbroken chain of time relationships on-board, and between the spacecraft and ground, provides a solution to the problem of knowing when an event took place on-board in any given space-time frame.

The GRCTM is foreseen to be used both as a central elapsed time reference in the spacecraft data management system, as well as the local elapsed time reference in an instrument or other subsystem. By using standardised AMBA interfaces, the integration of the GRCTM should be simple for most systems.

157.1.3 Functions not included

The GRCTM does not support alarm services.

The GRCTM does not support setting of an arbitrary epoch time.

157.2 Data formats

All Elapsed Time (ET) information handled by GRCTM is compliant with the CCSDS Unsegmented Code defined in [CCSDS] and repeated hereafter.

157.2.1 Reference documents

[CCSDS] Time Code Formats, CCSDS 301.0-B-4, www.ccsds.org

[PSS] Packet Telemetry Standard, ESA PSS-04-106, Issue 1, January 1988

157.2.2 CCSDS Unsegmented Code: Preamble Field (P-Field)

The time code preamble field (P-Field) may be either explicitly or implicitly conveyed. If it is implicitly conveyed (not present with T-Field), the code is not self-identified, and identification must be

obtained by other means. As presently defined, the explicit representation of the P-Field is limited to one octet whose format is described hereafter.

*Table 2644.*CCSDS Unsegmented Code P-Field definition

Bit	Value		Interpretation
0	0		Extension flag
1 - 3	“001”	1958 January 1 epoch (Level 1) [1]	Time code identification
	“010”	Agency-defined epoch (Level 2) [1]	
4 - 5	(number of octets of coarse time) - 1		Detail bits for information on the code
6 - 7	(number of octets of fine time)		

1 *For the Standard Spacecraft Time Source Packet defined in the ESA Packet Telemetry Standard, bits 1 to 3 must are set to 010_b.*

157.2.3 CCSDS Unsegmented Code: Time Field (T-Field)

For the unsegmented binary time codes described herein, the T-Field consists of a selected number of contiguous time elements, each element being one octet in length. An element represents the state of 8 consecutive bits of a binary counter, cascaded with the adjacent counters, which rolls over at a modulo of 256.

*Table 2645.*CCSDS Unsegmented Code T-Field definition

CCSDS Unsegmented Code											
Preamble Field	Time Field										
	Coarse time						Fine time				
-	2^{31}	2^{24}	2^{23}	2^{16}	2^{15}	2^{8}	2^{7}	2^{0}	2^{-1}	2^{-8}	2^{-9}	2^{-15}	2^{-16}	2^{-24}

The basic time unit is the second. The T-Field consists of 32 bits of coarse time (seconds) and 24 bits of fine time (sub seconds). The coarse time code elements are a count of the number of seconds elapsed from the epoch. The 32 bits of coarse time results in a maximum ambiguity period of approximately 136 years. Arbitrary epochs may be accommodated as a Level 2 code. The 24 bits of fine code elements result in a resolution of 2-24 second (about 60 nanoseconds). This code is not UTC-based and leap second corrections do not apply according to CCSDS.

157.2.4 Waveforms

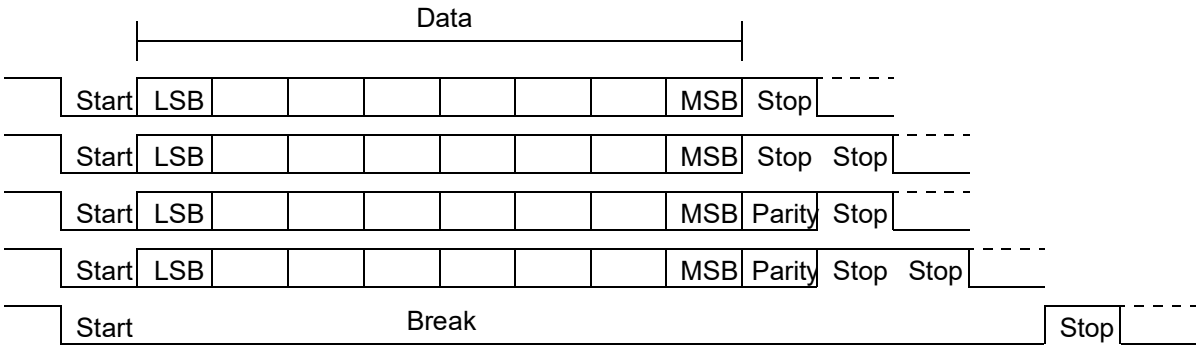


Figure 387. Bit asynchronous protocol

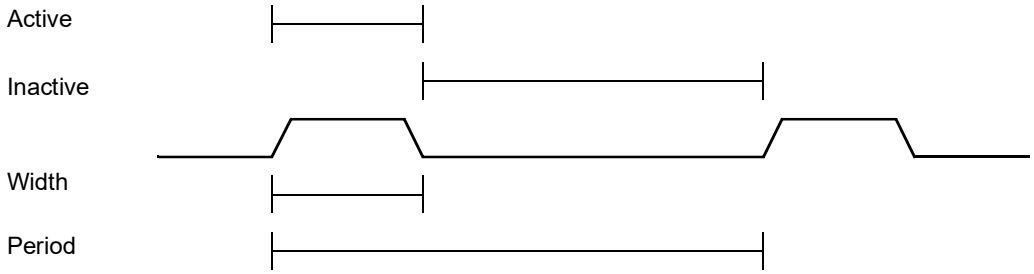


Figure 388. Pulse generation waveform

157.3 Operation

The CCSDS Time Manager (GRCTM) synthesizable core can be configured for various purposes. The different functions presented hereafter can be used to from a GRCTM to act as a master, slave, or master and slave.

157.3.1 Elapsed Time (ET)

The local Elapsed Time (ET) counter is based on a default 32 bit coarse time field and a 24 bit fine time field, complying to the CCSDS Unsegmented Code (CUC) T-Field. The width of the two time fields is fixed. The counter implementing the ET is incremented on the system clock only when enabled by the frequency synthesizer described below. The ET is incremented with a pre-calculated increment value, which matches the synthesised frequency. The local ET is output in the CUC format, P-Field and T-Field, to be used by an application embedding the GRCTM. The P-Field is static with the Time Code Identifier is set to 010b.

157.3.2 Frequency Synthesizer (FS)

The binary frequency required to determine the ET counter increment is derived from the system clock using a frequency synthesizer. The frequency synthesizer is incremented with a pre-calculated increment value, which matches the available system clock frequency. The FS simply generates a tick every time it wraps around, which makes the ET to step forward with the pre-calculated increment value. The output of the frequency synthesizer is used for enabling the increment of the local ET as described above. A 24 to 32 bit wide FS causes a systematic drift of less than 1 second/day.

157.3.3 TimeWire Interface (TW)

The GRCTM provides two TimeWire (TW) interfaces, one for the master function and one for the slave function. The TimeWire interface is used for distributing the Elapsed Time (ET) of a GRCTM master to one or more GRCTM slaves. The information carried in the synchronisation message comprises the Elapsed Time Field, which is 4 bytes, and a synchronisation pulse which is sent as a BREAK command. The synchronisation pulse is used for synchronising both the ET and the phase of the Frequency Synthesiser (FS) and is done once every second.

The GRCTM slave automatically synchronizes its ET with that of the GRCTM master without requiring any user support. A GRCTM that acts both as a master and a slave will be slaved to another master via a slave TimeWire interface, and will simultaneously distribute its own ET to other GRCTM slaves. The GRCTM slave will continue to work undisturbed in case a GRCTM master has failed. It is also possible to disable the synchronisation by means of a register further to avoid failure propagation. The TimeWire interface provides means for synchronising a GRCTM slave from a GRCTM that acts both as a master and a slave, which being synchronised in turn from another GRCTM master.

The message is sent just before the synchronisation instance. A BREAK command is sent just after the message to indicate the synchronisation pulse. The instance of the synchronisation actually depends on the reception of the BREAK command and the time it takes to generate an internal pulse

in the receiver on which the previously sent message is latched into the ET counter of the slave. The baseline is to send the synchronisation message and pulse to coincide with the wrap around of the sub-second bits in the ET. However, the time at which the message is sent out from the master is configurable by means of a generic (based on the fine part of the ET). An additional generic is provided for the fine tuning of the message start. On the slave side, a generic is provided to set the fine part of the ET at which the synchronisation pulse will occur. It is thus possible to synchronise the two units at any arbitrary point in time, provided it is done once a second.

To tolerate large skew and drift differences between the clocks driving the master and the slave GRCTM, a staged approach has been taken for the distribution of the synchronisation message and the synchronisation pulse. The first GRCTM master in a time chain synchronises its GRCTM slaves before it is time for these to act as masters and in turn send their synchronisation messages and pulses to their slaves. This is done to avoid that the first GRCTM master will synchronise the GRCTM slaves in such a way that no synchronisation messages are being sent out from these GRCTM due to clock drift. Since the synchronisation only occurs once a second, the first GRCTM master in a time chain has one second of time available to synchronise its GRCTM slaves before they synchronise their slaves in turn.

The TimeWire interface is based on a bit asynchronous interface (RS232/422) with the following programmable specification:

- 115200 baud (configured by generics/register bits)
- 1 start bit, 8 bit data, 1 or 2 stop bits (configured by generics/register bit)
- odd parity is generated in transmitter, but ignored in the receiver (configured by generics/register bit)
- no handshake
- message delimiting via BREAK command (13 bits when sent)
- synchronisation via BREAK command (13 bits when sent)

Table 2646. TimeWire transmission protocol

Byte Number	Elapsed Time Coarse Register (ETCR)	CCSDS Unsegmented Code Time Field Coarse Part	Comment
First	[31:24]	2^{31} 2^{24}	
Second	[23:16]	2^{23} 2^{16}	
Third	[15:8]	2^{15} 2^{8}	
Fourth	[7:0]	2^{7} 2^{0}	
Fifth	N/A	[RS232 Break Command]	Used as synchronisation pattern

Note that it is not possible for the TimeWire to carry sub-second phase information due to the usage of the above RS232/422 type of interface.

157.3.4 Datation

The GRCTM comprises three datation registers for the purpose of datation of user events relative the ET counter. The datation is triggered by three external edge sensitive inputs (programmable rising or falling edge).

A fourth datation register is provided for sampling the ET counter when generating a Standard Spacecraft Time Source Packet as described below.

Each of the three general datation services is automatically disabled after an occurrence and is not re-enabled until the corresponding fine time register is read. The format of all four datation registers is compliant to the CUC T-Field.

The ET counter can be accessed directly via the AMBA AHB interface. This can be used for direct datation from software.

157.3.5 Interrupts

The GRCTM provides individual interrupt lines for the incoming datation inputs, the time strobe input and the occurrence of the individual pulse outputs. The interrupt lines are asserted for at least two system clock cycles and can be connected to an external interrupt controller. The interrupts indicate that a new datation value can be read. The interrupts defined in table 2647 are generated.

*Table 2647.*Interrupts

Interrupt offset	Interrupt name	Description
1:st	DRL0	Datation Register 0 Latched
2:nd	DRL1	Datation Register 1 Latched
3:rd	DRL2	Datation Register 2 Latched
4:th	STL	Spacecraft Time Register Latched
5:th	PULSE0	Pulse 0 interrupt
6:th	PULSE1	Pulse 1 interrupt
7:th	PULSE2	Pulse 2 interrupt
8:th	PULSE3	Pulse 3 interrupt
9:th	PULSE4	Pulse 4 interrupt
10:th	PULSE5	Pulse 5 interrupt
11:th	PULSE6	Pulse 6 interrupt
12:th	PULSE7	Pulse 7 interrupt
13:th	ETSET	ET set (only when VHDL generic gSetET is 1)

157.3.6 Pulses

The GRCTM provides eight external outputs used for clock pulse distribution. The timing of each pulse output is individually derived from the Elapsed Time counter. It is possible to program for each pulse output individually the following parameters:

- periodicity pulse
- width of pulse
- polarity of pulse
- enable/disable pulse generation (reset status is disabled)

The pulse has two parts, the active and the inactive part. The active part always starts the pulse, followed by the inactive part. The polarity or logical level of the active part is programmable. The inactive part takes the logical inversion of the active pulse, and is the default output from the generator when the pulse is not issued or the overall generation is disabled. The leading edge of the active pulse part is aligned with the 1 second transition of the Elapsed Time counter.

The periodicity of the pulse corresponds to one of the ET bits that can be selected in the range 27 to 2-8 seconds, providing a range from 128 seconds to 3,91 ms, i.e. 0,0078 to 256 Hz frequency. See register definition for details.

The width of the active part of the pulse corresponds to one of the ET bits that can be selected in the range 26 to 2-9 seconds, providing a range from 64 seconds to 1,95 ms. See register definition for details.

It is possible to generate a pulse that has a duty cycle of 50%. It is also possible to generate a pulse for which the active part is as short as 2-9 seconds, and its period is as high as 27 seconds. The effective duty cycle can be as low as 2-9/27 for the longest period, up to 50% for the shortest period of 2-8 sec-

onds = 256 Hz. The duty cycle choice becomes more restricted as the frequency increases. Note that it is only possible to reduce the duty cycle in one direction: 50%/50%, 25%/75% ... 1%/99%. The active part of the pulse can thus never be more than 50% of the cycle. It should be noted that the active pulse width must be at most 50% of the pulse period. This is a requirement on the software usage.

The pulse outputs are guaranteed to be spike free. If the re-synchronisation of the GRCTM in slave mode occurs within 0,5 ms of the expected synchronisation instance, the ongoing pulse output width will be accurate to within 0,5 ms. Else, the pulse output will remain unchanged corresponding to up to four times the expected output width.

If a pulse output is disabled by means of writing to the corresponding register (PDRx) (i.e. writing a zero to the Pulse Enable bit (PE)), the pulse output will be immediately driven to the inversion of the Pulse Level bit (PL), which corresponds to the level of the inactive part of the pulse. It is thus possible to modify immediately the pulse output by disabling it using the PE bit and then changing the PL bit, since the output will always drive the inversion of the PL bit while disabled.

157.3.7 Standard Spacecraft Time Source Packet

As mentioned above, the GRCTM comprises one datation register for sampling the ET counter when generating a Standard Spacecraft Time Source Packet according to the ESA Packet Telemetry Standard, AD2, according to the following programmable bit asynchronous protocol specification:

- 115200 baud (configured by generics/register bit)
- 1 start bit, 8 bit data, 1 or 2 stop bits (configured by generics/register bit)
- odd parity is generated (configured by generics/register bit)
- no handshake (note that there is a BUSY signal for internal handshake, that can block the generation of additional packets)
- message delimiting via BREAK command (13 zero bits when sent)

The Spacecraft Time Coarse Register (STCR) and the Spacecraft Time Fine Register (STFR) are available for readout of the datation time from the software. The software cannot block or initiate a datation on these registers, since controlled from an external input pin. If multiple datation have

occurred since the registers were previously read, only the time at the first datation can be read from the registers.

Table 2648. Standard Spacecraft Time Source Packet

Octet number	Name	Value
0	Packet Header Octet 0	0x00
1	Packet Header Octet 1	0x00
2	Segment Flags & Sequence Count (0 to 5)	11_b & 14 bit counter
3	Sequence Count (6 to 13)	
4	Packet Length (0 to 7)	0x00
5	Packet Length (8 to 15)	0x00
6	Data Field & Sample Rate (0 to 3)	0000b & sampling rate
7	P-Field	0x2F
8	T-Field (2^{31} to 2^{24})	
9	T-Field (2^{23} to 2^{16})	
10	T-Field (2^{15} to 2^{8})	
11	T-Field (2^{7} to 2^{0})	
12	T-Field (2^{-1} to 2^{-8})	
13	T-Field (2^{-9} to 2^{-16})	
14	T-Field (2^{-17} to 2^{-24})	

Table 2649. Time sample rate

Bit	Rate (in frames)	Bit	Rate (in frames)
0000_b	1	0101_b	32
0001_b	2	0110_b	64
0010_b	4	0111_b	128
0011_b	8	1000_b	256
0100_b	16	others	undefined

157.3.8 AMBA AHB slave interface

All time services, including the Elapsed Time counter in the embedded GRCTM core, are clocked by the AMBA AHB clock HCLK. All input signals are assumed to be synchronous with the AMBA AHB interface clock HCLK. No input signal synchronisation is performed in the core. All outputs are synchronous with the AMBA AHB interface clock.

The AMBA AHB slave interface supports 32 bit wide data input and output. Since each access is a word access, the two least significant address bits are assumed always to be zero. Only address bits 23:0 are decoded. Note that address bits 31:24 are not decoded and should thus be handled by the AHB arbiter/decoder. The address input of the AHB slave interfaces is thus incompletely decoded. Misaligned addressing is not supported. One wait state is introduced for read and write accesses.

When the CCSDS field is narrower than the AMBA data width, zeros are padded to the right. Re-mapping between the opposing numbering conventions in the CCSDS and AMBA documentation is performed automatically. For read accesses, unmapped bits are always driven to zero.

The interface provides direct access to the T-Field of the ET counter.

The AMBA AHB interface has been reduced in function to support only what is required for the GRCTM. The following AMBA AHB features are constrained:

- Only supports HSIZE=WORD, HRESP_ERROR generated otherwise
- Only supports HMASTLOCK='0', HRESP_ERROR generated otherwise
- Only supports HBURST=SINGLE and INCR, HRESP_ERROR generated otherwise
- No HPROT decoding
- No HSPLIT generated
- No HRETRY generated
- HRESP_ERROR generated for unmapped addresses, and for write accesses to register without any writeable bits
- Only big-endianness is supported.
- Frequency synthesis and time increment configuration

The increment values for the ET and FS counters depend on the implemented width of each counter and the frequency of the available on the system clock.

157.3.9 Miscellaneous

The accuracy of the transmission or reception baud rate of the bit asynchronous serial interface is dependent on the selected system frequency and baud rate. The number of system clock periods used for sending or receiving a bit is directly proportional to the integer part of the division of the system frequency with the baud rate.

The BREAK command received on the bit asynchronous serial interface is a sequence of logical zeros that is at least one bit period longer than the normal byte frame, i.e. start bit, eight data bits, optional parity, one or two stop bits. When transmitted, it is always 13 bits.

157.3.10Numbering and naming conventions

Convention according to the CCSDS recommendations, applying to time structures:

- The most significant bit of an array is located to the left, carrying index number zero.
- An octet comprises eight bits.

*Table 2650.*CCSDS n-bit field definition

CCSDS n-bit field		
most significant		least significant
0	1 to n-2	n-1

Convention according to AMBA specification, applying to the APB/AHB interfaces:

- Signal names are in upper case, except for the following:
- A lower case 'n' in the name indicates that the signal is active low.
- Constant names are in upper case.
- The least significant bit of an array is located to the right, carrying index number zero.
- Big-endian support.

*Table 2651.*AMBA n-bit field definition

AMBA n-bit field		
most significant		least significant
n-1	n-2 down to 1	0

General convention, applying to all other signals and interfaces:

- Signal names are in mixed case.
- An upper case '_N' suffix in the name indicates that the signal is active low.

157.4 Registers

The core is programmed through registers mapped into AHB I/O address space. Only 32-bit single-accesses to the registers are supported.

*Table 2652.*GRCTM registers

AHB address offset	Register
0x00	Global Reset Register (GRR)
0x04	Global Control Register (GCR)
0x08	Global Status Register (GSR)
0x0C	N/A
0x10	N/A
0x14	Preamble Field Register (PFR)
0x18	Elapsed Time Coarse Register (ETCR)
0x1C	Elapsed Time Fine Register (ETFR)
0x20	Datation Coarse Register 0 (DCR0)
0x24	Datation Fine Register 0 (DFR0)
0x28	Datation Coarse Register 1 (DCR1)
0x2C	Datation Fine Register 1 (DFR1)

*Table 2652.*GRCTM registers

AHB address offset	Register
0x30	Datation Coarse Register 2 (DCR2)
0x34	Datation Fine Register 2 (DFR2)
0x38	Spacecraft Time Datation Coarse Register (STCR)
0x3C	Spacecraft Time Datation Fine Register (STFR)
0x40	Pulse Definition Register 0
0x44	Pulse Definition Register 1
0x48	Pulse Definition Register 2
0x4C	Pulse Definition Register 3
0x50	Pulse Definition Register 4
0x54	Pulse Definition Register 5
0x58	Pulse Definition Register 6
0x5C	Pulse Definition Register 7
0x60	Pending Interrupt Masked Status Register
0x64	Pending Interrupt Masked Register
0x68	Pending Interrupt Status Register
0x6C	Pending Interrupt Register
0x70	Interrupt Mask Register
0x74	Pending Interrupt Clear Register
0x78	N/A
0x7C	N/A
0x80	Elapsed Time Increment Register (ETIR)
0x84	Frequency Synthesizer Increment Register (FSIR)
0x88	Serial Configuration Register
0x8C	N/A
0x90	TimeWire Start Configuration Register
0x94	TimeWire Adjust Configuration Register
0x98	TimeWire Transmit Configuration Register
0x9C	TimeWire Receive Configuration Register
0xA0	Set Elapsed Time Coarse Register (SetETCR)
0xA4	Set Elapsed Time Fine Register (SetETFR)

157.4.1 Global Reset Register

Table 2653. 0x00 - GRR - Global Reset Register

31 24	23 1	0
SEB	RESERVED	SRST
0	0	0
w	r	rw

31: 24 SEB (Security Byte):

Write: '0x55'= the write will have effect (the register will be updated).
Any other value= the write will have no effect on the register.

Read: All zero.

23: 1 RESERVED

Write: Don't care.

Read: All zero.

0 System reset (SRST): **[1]**

Write: '1'= initiate reset,'0'= do nothing

Read: '1'= unsuccessful reset, '0'= successful reset

157.4.2 Global Control Register

Table 2654. 0x04 - GCR - Global Control Register

31 24	23 13	12	11	10	9	8	7	6 0
SEB	RESERVED	DRE2	DRE1	DRE0	EXT	SYNC	FREQDIS	RESERVED
0	0	0	0	0	0	0	0	0
w	r	rw	rw	rw	rw	rw	rw	r

31: 24 SEB (Security Byte):

Write: '0x55'= the write will have effect (the register will be updated).
Any other value= the write will have no effect on the register.

Read: All zero.

23: 13 RESERVED

Write: Don't care.

Read: All zero.

12 Datation Register2 Edge (DRE2)

Write/Read: '0'= falling, '1'= rising

11 Datation Register1 Edge (DRE1)

Write/Read: '0'= falling, '1'= rising

10 Datation Register0 Edge (DRE0)

Write/Read: '0'= falling, '1'= rising

9 External synchronisation (EXT) (through external synchronisation interface)

Write/Read: '0'= disabled, '1'= enable.

8 Synchronise slave (SYNC) (through TimeWire)

Write/Read: '0'= disabled, '1'= enabled

7 Disable frequency synthesizer from driving Elapsed Time counter (FREQDIS)

Write/Read: '1'= disabled, '0'= enabled

6: 0 RESERVED

Write: Don't care.

Read: All zero.

157.4.3 Global Status Register

Table 2655. 0x08 - GSR - Global Status Register [8]

31 5	4	3	2	1	0
RESERVED	SARM	STL	DRL2	DRL1	DRL0
0	0	0	0	0	0
r	r	r	r	r	r

31: 5 RESERVED
- Write: Don't care.
- Read: All zero.

4 Set Elapsed Time counter Arm (SARM): [9]
- Write: Don't care.
- Read: '1' = Ready to set ET, '0' = reset value, or already ET has been set

3 Spacecraft Time Register Latched (STL): [3]
- Write: Don't care.
- Read: '1' = Latched with new value, '0' = old value

2 Datation Register 2 Latched (DRL2): [4]
- Write: Don't care.
- Read: '1' = Latched with new value, '0' = old value

1 Datation Register 1 Latched (DRL1): [4]
- Write: Don't care.
- Read: '1' = Latched with new value, '0' = old value

0 Datation Register 0 Latched (DRL0): [4]
- Write: Don't care.
- Read: '1' = Latched with new value, '0' = old value

157.4.4 Preamble Field Register

Table 2656. 0x14 - PFR - Preamble Field Register [8]

31 8	7 0
RESERVED	P-FIELD
0	0x2E
r	r

31: 8 RESERVED
- Write: Don't care.
- Read: All zero.

7: 0 Preamble Field (P-Field):
- Write: Don't care.
- Read: Static P-Field

157.4.5 Elapsed Time Coarse Register

*Table 2657.*0x18 - ETCR - Elapsed Time Coarse Register [8]

31 0
T-FIELD, COARSE
0
r

31: 0 T-Field, coarse part [5]
- Write: Don't care.
- Read: T-Field, coarse part

157.4.6 Elapsed Time Fine Register

Table 2658. 0x1C - ETFR - Elapsed Time Fine Register [8]

31 8	7 0
T-FIELD, FINE	RESERVED
0	0
r	r

31: 8 T-Field, fine part [5]
- Write: Don't care.
- Read: T-Field, fine part

7: 0 RESERVED
- Write: Don't care.
- Read: All zero.

157.4.7 Datation Time Coarse Register 0

Table 2659. 0x20 - DCR0 - Datation Time Coarse Register 0 [8]

31 0
T-FIELD, COARSE
0
r

31: 0 T-Field, coarse part [6]
- Write: Don't care.
- Read: T-Field, coarse part

157.4.8 Datation Time Fine Register 0

Table 2660. 0x24 - DFR0 - Datation Time Fine Register 0 [8]

31 8	7 0
T-FIELD, FINE	RESERVED
0	0
r	r

31: 8 T-Field, fine part [6]
- Write: Don't care.
- Read: T-Field, fine part

7: 0 RESERVED
- Write: Don't care.
- Read: All zero.

157.4.9 Datation Time Coarse Register 1

Table 2661. 0x28 - DCR1 - Datation Time Coarse Register 1 [8]

31 0
T-FIELD, COARSE
0
r

31: 0 T-Field, coarse part [6]
- Write: Don't care.
- Read: T-Field, coarse part

157.4.10Datation Time Fine Register 1

*Table 2662.*0x2C - DFR1 - Datation Time Fine Register 1 [8]

31 8	7 0
T-FIELD, FINE	RESERVED
0	0
r	r

31: 8 T-Field, fine part [6]
Write: Don't care.
Read: T-Field, fine part

7: 0 RESERVED
Write: Don't care.
Read: All zero.

157.4.11Datation Time Coarse Register 2

*Table 2663.*0x30 - DCR2 - Datation Time Coarse Register 2 [8]

31 0
T-FIELD, COARSE
0
r

31: 0 T-Field, coarse part [6]
Write: Don't care.
Read: T-Field, coarse part

157.4.12Datation Time Fine Register 2

*Table 2664.*0x34 - DFR2 - Datation Time Fine Register 2 [8]

31 8	7 0
T-FIELD, FINE	RESERVED
0	0
r	r

31: 8 T-Field, fine part [6]
Write: Don't care.
Read: T-Field, fine part

7: 0 RESERVED
Write: Don't care.
Read: All zero.

157.4.13Spacecraft Time Datation Coarse Register

*Table 2665.*0x38 - STCR - Spacecraft Time Datation Coarse Register [8]

31 0
T-FIELD, COARSE
0
r

31: 0 T-Field, coarse part [7]
Write: Don't care.
Read: T-Field, coarse part

157.4.14Spacecraft Time Datation Fine Register

*Table 2666.*0x3C - STFR - Spacecraft Time Datation Fine Register [8]

31 8	7 0
T-FIELD, FINE	RESERVED
0	0
r	r

31: 8 T-Field, fine part [7]
- Write: Don't care.
- Read: T-Field, fine part

7: 0 RESERVED
- Write: Don't care.
- Read: All zero.

157.4.15Pulse Definition Register 0 to 7

*Table 2667.*0x40 - 0x5C - PDR0 to PDR7 - Pulse Definition Register 0 to 7

31 24	23 20	19 16	15 11	10	9 2	1	0
RESERVED	PP	PW	RESERVED	PL	RESERVED	PE	R
0	0	0	0	1	0	0	0
r	rw	rw	r	rw	r	rw	r

31: 24 RESERVED
- Write: Don't care.
- Read: All zero.

23: 20 Pulse Period (PP):
- Write/Read:
  - '0000' = 2^{7} seconds
  - '0001' = 2^{6} seconds
  - '0010' = 2^{5} seconds
  - ...
  - '1110' = 2^{-7} seconds
  - '1111' = 2^{-8} seconds
  - Period = $2^{(7-\mathbf{PP})}$
  - Frequency = $2^{-(7-\mathbf{PP})}$

19: 16 Pulse Width (PW):
- Write/Read:
  - '0000' = 2^{6} seconds
  - '0001' = 2^{5} seconds
  - '0010' = 2^{4} seconds
  - ...

*Table 2667.*0x40 - 0x5C - PDR0 to PDR7 - Pulse Definition Register 0 to 7

	'1110'	=	2^{-8} seconds
	'1111'	=	2^{-9} seconds
	Width	=	$2^{(6-PW)}$

15: 11	RESERVED
	Write: Don't care.
	Read: All zero.
10	Pulse Level (PL): Defines logical level of active part of pulse output.
	Write/Read: '0'= Low, '1'= High
9: 2	RESERVED
	Write: Don't care.
	Read: All zero.
1	Pulse Enable (PE):
	Write/Read: '0'= disabled, '1'= enabled
0	RESERVED
	Write: Don't care.
	Read: All zero.

157.4.16Elapsed Time Increment Register

*Table 2668.*0x80 - ETIR - Elapsed Time Increment Register

31	7	0
RESERVED	ETINC	
0	*	
r	rw*	

7: 0 ETINC

Write/Read: Increment, in number ET Fine Time LSB.

(Presence of ETIR can be discovered by read/write access, resulting in an error if not implemented.)

157.4.17Frequency Synthesizer Increment Register

*Table 2669.*0x84 - FSIR - Frequency Synthesizer Increment Register

31	0
FSINC	
*	
rw*	

31: 0 FSINC

Write/Read: Increment

(The number of implemented bits can be discovered by writing all ones and the reading them back.)

(Presence of ETIR can be discovered by read/write access, resulting in an error if not implemented.)

157.4.18Serial Configuration Register

*Table 2670.*0x88 - SCR - Serial Configuration Register

31 30	29	28	27 16	15 0
R	ODD	TWO	RESERVED	BAUDTHRESHOLD
0	*	*	0	*
r	rw	rw	r	rw

31: 30 RESERVED

28 ODD: Send odd parity and ignore received parity when set.

28 TWO: Send and receive two stop bits when set.

27: 16 RESERVED

15: 0 BAUDTHRESHOLD: Set baud rate: system frequency / baud rate

157.4.19TimeWire Start Configuration Register

*Table 2671.*0x90 - TWSC - TimeWire Start Configuration Register

31	8	7	0
START		RESERVED	
*		0	
rw		r	

31: 8 START: Defines T-Field fine part value when to start sending TimeWire message.

7: 0 RESERVED

157.4.20TimeWire Adjust Configuration Register

*Table 2672.*0x94 - TWAC - TimeWire Adjust Configuration Register

31 8	7 0
RESERVED	ADJUST
0	*
r	rw

31: 1 RESERVED

7: 0 ADJUST: Delay in number of systems clock when to start sending TimeWire message.

157.4.21TimeWire Transmit Configuration Register

*Table 2673.*0x98 - TWTC - TimeWire Transmit Configuration Register

31 1	0
RESERVED	TRANSMIT
0	*
r	rw

31: 1 RESERVED

0 TRANSMIT: T-Field fine part value other than 0 assumed for sent TimeWire message.

157.4.22TimeWire Receive Configuration Register

*Table 2674.*0x9C - TWRC - TimeWire Receive Configuration Register

31 8	7 0
RECEIVE	RESERVED
*	0
rw	w

31: 8 RECEIVE: Defines T-Field fine part value corresponding to received TimeWire message time point.

7: 0 RESERVED

157.4.23Set Elapsed Time Coarse Register

*Table 2675.*0xA0 - SETCR - Set Elapsed Time Coarse Register [9]

31 0
T-FIELD, COARSE
0
rw

31: 0 T-Field, coarse part [9]

Write: T-Field, coarse part

Read: T-Field, coarse part

157.4.24Set Elapsed Time Fine Register

*Table 2676.*0xA4 - SETFR - Set Elapsed Time Fine Register [9]

31 8	7 0
T-FIELD, FINE	RESERVED
0	0
rw	r

31: 8 T-Field, fine part [9]
- Write: T-Field, fine part
- Read: T-Field, fine part

7: 0 RESERVED
- Write: Don't care.
- Read: All zero.

Legend:

[1] The global system reset caused by the SRST-bit in the GRR-register results in the following actions:
- Initiated by writing a '1", gives '0' on read-back when the reset was successful.
- No need to write a '0' to remove the reset.
- Unconditionally, means no need to check/disable something in order for this reset-function to correctly execute.
- Could of course lead to data-corruption coming/going from/to the reset core.

Behaviour:
- Resets the complete core (all logic, buffers & register values)
(except for the ET and FS counters which continue running undisturbed)
- Behaviour is similar to a power-up.
- This reset shall not cause any spurious interrupts

{Note that the above actions require that the HRESET signal is fed back inverted to HRESETn}

[2] The channel reset results in the following actions:
Not implemented in Global Configuration Register.

[3] This bit is sticky which means that it remains asserted until the corresponding STFR register is read at which time the bit is cleared. The corresponding registers should be read in the STCR – STFR order.

[4] This bit is sticky which means that it remains asserted until the corresponding Defraud register is read at which point the bit is cleared. The corresponding registers should be read in the DCRx – DFRx order.

[5] When ETCR is read, the ETFR register is latched and are not released until ETFR has been read. The registers should be read in the ETCR – ETFR order.

[6] The coarse and fine time part of the register pair is latched on an external event and is released on reading the corresponding fine time register. No new event is accepted until the corresponding fine time register has been read.

[7] The coarse and fine time part of the register pair is latched on an external event. No new event is accepted until the corresponding fine time register has been read. This does not prevent datations to occur and Standard Spacecraft Time Source Packet to be generated.

[8] An AMBA AHB ERROR response is generated if a write access is attempted to a register which does not have any writeable bits.

[9] The GSR.SARM bit is set when the SetETFR register is written to. It is cleared when an active high signal is detected on the CTMIN.SETET input, at which point the SetETCR and SetETFR contents are written to the ET counter (ETCR and ETFR) and the frequency synthesizer is reset. Only available when gSetET VHDL generic is set to 1.

157.4.25Interrupt registers

The interrupt registers give complete freedom to the software, by providing means to mask interrupts, clear interrupts, force interrupts and read interrupt status.

When an interrupt occurs the corresponding bit in the Pending Interrupt Register is set. The normal sequence to initialize and handle a module interrupt is:

- Set up the software interrupt-handler to accept an interrupt from the module.
- Read the Pending Interrupt Register to clear any spurious interrupts.
- Initialize the Interrupt Mask Register, unmasking each bit that should generate the module interrupt.
- When an interrupt occurs, read the Pending Interrupt Status Register in the software interrupt-handler to determine the causes of the interrupt.
- Handle the interrupt, taking into account all causes of the interrupt.
- Clear the handled interrupt using Pending Interrupt Clear Register.

Masking interrupts: After reset, all interrupt bits are masked, since the Interrupt Mask Register is zero. To enable generation of a module interrupt for an interrupt bit, set the corresponding bit in the Interrupt Mask Register.

Clearing interrupts: All bits of the Pending Interrupt Register are cleared when it is read or when the Pending Interrupt Masked Register is read. Reading the Pending Interrupt Masked Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register. Selected bits can be cleared by writing ones to the bits that shall be cleared to the Pending Interrupt Clear Register.

Forcing interrupts: When the Pending Interrupt Register is written, the resulting value is the original contents of the register logically OR-ed with the write data. This means that writing the register can force (set) an interrupt bit, but never clear it.

Reading interrupt status: Reading the Pending Interrupt Status Register yields the same data as a read of the Pending Interrupt Register, but without clearing the contents.

Reading interrupt status of unmasked bits: Reading the Pending Interrupt Masked Status Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register, but without clearing the contents.

The interrupt registers comprise the following:

- Pending Interrupt Masked Status Register[PIMSR]R
- Pending Interrupt Masked Register[PIMR]R
- Pending Interrupt Status Register[PISR]R
- Pending Interrupt Register[PIR]R/W
- Interrupt Mask Register[IMR]R/W
- Pending Interrupt Clear Register[PICR]W

Table 2677. 0x60 - 0x74 - IR - Interrupt registers

31 13	12	11		4	3	2	1	0
R	ETSET	PULSE7	...	PULSE0	STL	DRL2	DRL1	DRL0
0	0	0	0	0	0	0	0	0
r	rw	rw	rw	rw	rw	rw	rw	rw

12: ETSET ET has been set from SetETCR and SetETFR registers (only when VHDL generic gSetET is 1)
11: PULSE7 Pulse 7 interrupt
10: PULSE6 Pulse 6 interrupt
9: PULSE5 Pulse 5 interrupt
8: PULSE4 Pulse 4 interrupt
7: PULSE3 Pulse 3 interrupt
6: PULSE2 Pulse 2 interrupt
5: PULSE1 Pulse 1 interrupt
4: PULSE0 Pulse 0 interrupt
3: STL Spacecraft Time Register Latched

2:	DRL2	Datation Register 2 Latched
1:	DRL1	Datation Register 1 Latched
0:	DRL0	Datation Register 0 Latched

All bits in all interrupt registers are reset to 0b after reset.

157.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x033. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

157.6 Configuration options

Table 2678 shows the configuration options of the core (VHDL generics).

*Table 2678.*Configuration options

Generic	Function	Description	Allowed range	Default
GRLIB AMBA plug&play settings				
hindex	AHB slave index		Integer	0
hirq	AHB slave interrupt		Integer	0
singleirq	Single interrupt	Enable interrupt registers	Integer	0
ioaddr	IO area address		0 - 16#FFF#	0
iomask	IO area mask		0 - 16#FFF#	16#FFF#
syncrst	synchronous reset		0 - 1	0
Features settings				
gProgrammable	Programmable	ET and FS increment, and TimeWire	0 - 1	0
gExternal	External support	Support for external synchronization input	0 - 1	0
gMaster	Master CTM support		0 - 1	0
gSlave	Slave CTM support		0 - 1	0
gDatation	Datation support		0 - 1	0
gPulse	Pulse support		0 - 1	0
gTimePacket	Time Packet support		0 - 1	0
gSetET	Set ET support	Register from which ET is set on next. input	0 - 1	0
Frequency synthesizer and Elapsed Time counter settings				
gFrequency	Frequency Synthesizer	Defines the accuracy of the synthesized reference time, the wider the synthesizer the less drift is induced.	2 - 32	30
gETIncrement	Increment of ET counter	Defines with what value the Elapsed Time counter is to be incremented each time when the Frequency Synthesizer wraps around. The ET increment needs to match the synthesized frequency.	Integer	4
gFSIncrement	Increment of FS counter	Defines increment value of the Frequency Synthesizer which is added to the counter every system clock cycle. It defines the frequency of the synthesized reference time. Should match ET increment.	Integer	135107990
TimeWire settings				
gTWStart	ETF at msg start	ET Fine at start of message [1]	Integer	16#FFE330#
gTWAdjust	Adjust phase of msg	System clock based tuning of message start	Integer	5
gTWTransmit	ETF at transmission	ET Fine at synchronisation (master) [1]	Integer	16#000000#
gTWRecieve	ETF at reception	ET Fine at synchronisation (slave) [1] [2]	Integer	16#FFE000#
gDebug	Debug when set	Only used for TW settings adjustments.	0 - 1	0
Asynchronous bit serial interface settings (TimeWire and Time Packet)				
gSystemClock	System frequency	System clock frequency [Hz]	Integer	33333333
gBaud	Baud rate	[Baud]	Integer	115200
gOddParity	Odd parity	Odd parity generated, but not checked	0 - 1	0
gTwoStopBits	Number of stop bits	0=one stop bit, 1=two stop bits	0 - 1	0

Legend:

[1] These generics are defined as 32 bit ET fine time values, thus 16#FFE000# corresponds to all the 24 implemented bits being all-ones.

[2] For proper mitigation of spikes on the Pulses[0:7] outputs, only the leftmost bits should be set.

157.7 Signal descriptions

Table 2679 shows the interface signals of the core (VHDL ports).

*Table 2679.*Signal descriptions

Signal name	Field	Type	Function	Description	Active
HRESETn	N/A	Input	Reset	Resets the ET & FS in the VHDL core. The signal is assumed synchronous with rising HCLK edge.	Low
CRESETn	N/A	Input	Reset	Resets all logic but the ET & FS in the VHDL core. The signal is assumed synchronous with rising HCLK edge.	Low
HCLK	N/A	Input	Clock		-

Table 2679. Signal descriptions

Signal name	Field	Type	Function	Description	Active
CTMIN	TWSLAVE	Input	TimeWire slave	TimeWire input	-
	DATATION		Datation input	The inputs are sampled on rising HCLK edge.	-
	TIMEMODE		Time rate select	Selects the rate of the time strobe periodicity	-
	TIMESTROBE		Time strobe input		-
	TIMEBUSY_N		Time packet busy		Low
	EXTERNALTIME		Elapsed Time	ET coarse [31:0] & ET fine [-1:-24]	-
	EXTERNALSYNC		Synchronization		High
	SETET		Set ET	Pulse to set ET from reg.	High
CTMOUT	TWMASTER	Output	TimeWire master	TimeWire output	
	PULSES		Pulse outputs	The outputs are driven on rising HCLK edge.	-
	TIMEPKT		Time packet data		-
	ELAPSEDTIME		Elapsed Time	ET coarse [31:0] & ET fine [-1:-24]	-
	ELAPSEDNEXT		Next Elapsed Time	ET coarse [31:0] & ET fine [-1:-24]	-
	ELAPSEDEVENT		ET increment		High
	ELAPSEDSYNC		Synchronisation		High
AHBIN	*	Input	AMB slave input signals		-
AHBOUT	*	Output	AHB slave output signals		-
	HIRQ(hirq+12)		Interrupts	ETSET output	Location on HIRQ bus depends on *hirq* generic. If *hirq*=0, no interrupt will be generated. If *singleirq*=1 only one common interrupt will be generate using *hirq*.
	HIRQ(hirq+11)			PULSES(7) output	
	HIRQ(hirq+10)			PULSES(6) output	
	HIRQ(hirq+9)			PULSES(5) output	
	HIRQ(hirq+8)			PULSES(4) output	
	HIRQ(hirq+7)			PULSES(3) output	
	HIRQ(hirq+6)			PULSES(2) output	
	HIRQ(hirq+5)			PULSES(1) output	
	HIRQ(hirq+4)			PULSES(0) output	
	HIRQ(hirq+3)			STL	
	HIRQ(hirq+2)			DRL2	
	HIRQ(hirq+1)			DRL1	
	HIRQ(hirq+0)			DRL0	

* see GRLIB IP Library User's Manual

157.8 Signal definitions and reset values

The signals and their reset values are described in table 2680.

*Table 2680.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
twmaster	Output	UART transmit data line	-	Logical 1
twslave	Input	UART receive data line	-	-
pulses	Output	Pulse output	-	Logical 0
datation	Input	Datation input	-	-
timemode	Input	Time rate select	-	-
timestrobe	Input	Time strobe input	-	-
timepkt	Output	Time packet data	-	Logical 0
timebusy_n	Input	Time packet busy	-	Low

157.9 Timing

The timing waveforms and timing parameters are shown in figure 389 and are defined in table 2681.

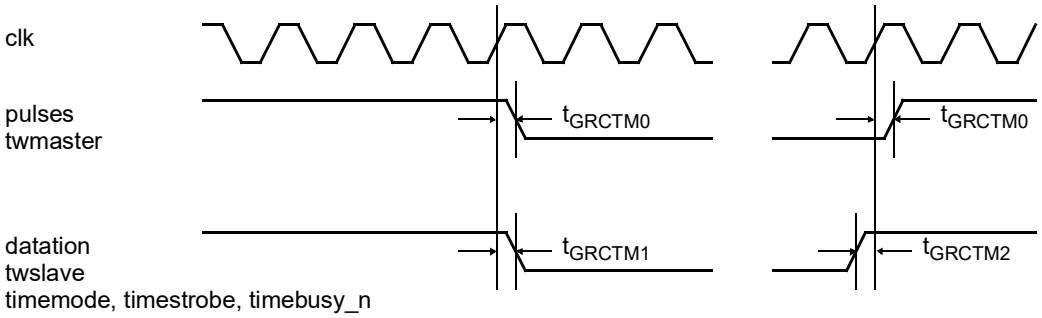


Figure 389. Timing waveforms

*Table 2681.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{GRCTM0}	clock to output delay	rising *clk* edge	TBD	TBD	ns
t_{GRCTM1}	input to clock hold	rising *clk* edge	-	-	ns
t_{GRCTM2}	input to clock setup	rising *clk* edge	-	-	ns

Note: The inputs are re-synchronized internally. The signals do not have to meet any setup or hold requirements.

157.10 Library dependencies

Table 2682 shows libraries used when instantiating the core (VHDL libraries).

*Table 2682.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_Types	Signals, component	Signals and component declaration

157.11 Instantiation

This example shows how the core can be instantiated.

```
   library  IEEE;
   use      IEEE.Std_Logic_1164.all;
   library  GRLIB;
   use      GRLIB.AMBA.all;
   library  TMTC;
   use      TMTC.TMTC_Types.all;

...

   component GRCTM is
   generic(
      hindex:        in    Integer                := 0;
      hirq:          in    Integer                := 0;
      ioaddr:        in    Integer                := 0;
      iomask:        in    Integer                := 16#fff#;
      syncrst:       in    Integer                := 0;
      gProgrammable: in    Integer                := 0;
      gExternal:     in    Natural range 0 to 1   := 0;  -- External sync
      gMaster:       in    Natural range 0 to 1   := 1;  -- Master CTM support
      gSlave:        in    Natural range 0 to 1   := 1;  -- Slave CTM support
      gDatation:     in    Natural range 0 to 1   := 1;  -- Datation support
      gPulse:        in    Natural range 0 to 1   := 1;  -- Pulse support
      gTimePacket:   in    Natural range 0 to 1   := 1;  -- Time Packet support

      gFrequency:    in    Positive               := 30; -- Frequency Synthesize
      gETIncrement:  in    Natural := 4;                 -- ET increment
      gFSIncrement:  in    Natural := 135107990;         -- FS increment

      gTWStart:      in    Natural := 16#FFE330#;        -- ETF at start of msg
      gTWAdjust:     in    Natural := 5;                 -- Adjust phase of msg
      gTWTransmit:   in    Natural := 16#000000#;        -- ETF at transmision
      gTWRecieve:    in    Natural := 16#FFE000#;        -- ETF at reception

      gSystemClock:  in    Natural := 33333333;          -- System frequency[Hz]
      gBaud:         in    Natural := 115200;            -- Baud rate
      gOddParity:    in    Natural range 0 to 1   := 0;  -- Odd parity
      gTwoStopBits:  in    Natural range 0 to 1   := 0;  -- Two stop bits
      gDebug:        in    Natural range 0 to 1   := 0); -- Debug mode when set
   port(
      -- AMBA AHB system signals
      HCLK:          in    Std_ULogic;                   -- System clock
      HRESETn:       in    Std_ULogic;                   -- Synchronised reset
      CRESETn:       in    Std_ULogic;                   -- Synchronised reset

      -- AMBA AHB slave interface
      AHBIn:         in    AHB_Slv_In_Type;              -- AHB slave input
      AHBOut:        out   AHB_Slv_Out_Type;             -- AHB slave output

      -- Time interfaces
      CTMIn:         in    GRCTM_In_Type;
      CTMOut:        out   GRCTM_Out_Type);
   end component GRCTM;
```

157.12 Configuration tuning

The gFrequency generic defines the width of the Frequency Synthesiser (FS). The greater the width, the smaller the drift induced. The gFSIncrement generic defines with what value the FS counter should be incremented to obtain a synthesized frequency that matches the least significant bit of the Elapsed Time (ET) counter, normally being 2**24 Hz. It is also possible to synthesize a frequency less than 2**24 Hz, which will require the gETIncrement generic to have a higher value than the default 1.

The gETIncrement generic defines with what value the ET counter should be incremented. The specified value is added to the current ET counter value. The addition is done to the least significant bit in the fine part of the ET counter, i.e. gETIncrement is multiplied with 2**-24 before the addition. The gETIncrement is normally set to 1, since the obtained synthesised frequency is normally 2**24 Hz.

For lower frequencies, the gETIncrement generic must be larger than 1. Note that the synthesized frequency must always be a power of two.

The following paragraphs define synchronization via the optional TimeWire master and slave interfaces.

The gTWStart generic defines at what time the transmission of the TimeWire message should start on the TWMaster output. The gTWStart value corresponds to the ET counter fine part assuming 24 bit resolution. E.g. 16#FFE330# corresponds to bit 2**-1 to 2**-24. The gTWAdjust generic defines the number of HCLK periods that should pass between the gTWStart time has occurred and the TimeWire message should be sent. This allows for fine grained adjustment of the starting point for the TimeWire message.

The gTWTransmit generic defines the ET fine part value that is transmitted virtually to the slave. Its only consequence is to decide whether the ET coarse part to be sent in the TimeWire message should be the same as the current ET in the master or be incremented by one second. The gTWRecieve generic defines the ET fine part value that should be loaded into the ET at synchronisation. Normally this will be 0 for slave only applications of the GRCTM.

When the gDebug generic is 1, the ET counter reset will be 0x000000 for the coarse part and 0xFF0000 for the fine part (assuming CUC 32 & 24 bit resolution). This achieves an ET synchronisation early in a simulation without the need to wait for a second of simulation time.

157.12.1Master configuration

The master configuration is used for the source of the time chain and supports the following features:

- Frequency Synthesizer (FS) and Elapsed Time (ET) counters
- Master TimeWire interface
- Datation by means of direct read out of the ET counter via AMBA AHB interface

The following register are available in this configuration: GRR, GCR, GSR, ETCR, ETFR.

Table 2678 shows the master configuration.

*Table 2683.*Master configuration

Generic	Function	Default
Features settings		
gMaster	Master CTM support	1
gSlave	Slave CTM support	0
gDatation	Datation support	0
gPulse	Pulse support	0
gTimePacket	Time Packet support	0
Frequency synthesizer and Elapsed Time counter settings		
gFrequency	Frequency Synthesizer	30
gETIncrement	Increment of ET counter (In this example a resolution is 2**-22 due to the slow system frequency.)	4
gFSIncrement	Increment of FS counter	135107990
TimeWire settings		
gTWStart	ETF at msg start	16#FFC3300#
gTWAdjust	Adjust phase of msg	5
gTWTransmit	ETF at transmission	16#FFE000#
gTWRecieve	ETF at reception	16#000000#
gDebug	Debug when set	0
Asynchronous bit serial interface settings (TimeWire and Time Packet)		
gSystemClock	System frequency	33333333
gBaud	Baud rate	115200
gOddParity	Odd parity	0
gTwoStopBits	Number of stop bits	0

157.12.2Master/Slave configuration

The master/slave configuration is used for an intermediate unit in the time chain, and supports all features.

All registers are available in this configuration.

Table 2678 shows the master/slave configuration.

*Table 2684.*Master/Slave configuration

Generic	Function	Default
Features settings		
gMaster	Master CTM support	1
gSlave	Slave CTM support	1
gDatation	Datation support	1
gPulse	Pulse support	1
gTimePacket	Time Packet support	1
Frequency synthesizer and Elapsed Time counter settings		
gFrequency	Frequency Synthesizer	30
gETIncrement	Increment of ET counter (In this example a resolution is 2**-22 due to the slow system frequency.)	4
gFSIncrement	Increment of FS counter	135107990
TimeWire settings		
gTWStart	ETF at msg start	16#FFE330#
gTWAdjust	Adjust phase of msg	5
gTWTransmit	ETF at transmission	16#0000000#
gTWRecieve	ETF at reception	16#FFE000#
gDebug	Debug when set	0
Asynchronous bit serial interface settings (TimeWire and Time Packet)		
gSystemClock	System frequency	33333333
gBaud	Baud rate	115200
gOddParity	Odd parity	0
gTwoStopBits	Number of stop bits	0

157.12.3Slave configuration

The slave configuration is used for a sink at the end of the time chain, e.g. in the payload, and supports the following features:

- Frequency Synthesizer (FS) and Elapsed Time (ET) counters
- Slave TimeWire interface
- Datation by means of direct read out of the ET counter via AMBA AHB interface

Note that the slave configuration can also support other features as required. Only those necessary for proper operation have been listed above.

The following register are available in this configuration: GRR, GCR, GSR, ETCR, ETFR.

Table 2678 shows the slave configuration.

*Table 2685.*Slave configuration

Generic	Function	Default
Features settings		
gMaster	Master CTM support	0
gSlave	Slave CTM support	1
gDatation	Datation support	0
gPulse	Pulse support	0
gTimePacket	Time Packet support	0
Frequency synthesizer and Elapsed Time counter settings		
gFrequency	Frequency Synthesizer	30
gETIncrement	Increment of ET counter (In this example a resolution is 2**-22 due to the slow system frequency.)	4
gFSIncrement	Increment of FS counter	135107990
TimeWire settings		
gTWStart	ETF at msg start	16#000000#
gTWAdjust	Adjust phase of msg	0
gTWTransmit	ETF at transmission	16#000000#
gTWRecieve	ETF at reception	16#000000#
gDebug	Debug when set	0
Asynchronous bit serial interface settings (TimeWire and Time Packet)		
gSystemClock	System frequency	33333333
gBaud	Baud rate	115200
gOddParity	Odd parity	0
gTwoStopBits	Number of stop bits	0

158 SPWCUC - SpaceWire - CCSDS Unsegmented Code Transfer Protocol

158.1 Overview

This interface implements the SpaceWire - CCSDS Unsegmented Code Transfer Protocol (CUCTP), providing automatic SpaceWire Time-Code transmission and reception, and automatic CUCTP packet reception. It also provides support for CUCTP packet transmission.

The SpaceWire - CCSDS Unsegmented Code Transfer Protocol interface (SPWCUC) is assumed to operate in an AMBA bus system where AMBA APB bus is present. The AMBA APB bus is used for configuration, control and status handling. The interface is tightly coupled with the CCSDS Time Manager (GRCTM) and the SpaceWire codec with AHB host Interface and RMAP target (GRSPW2).

158.2 Protocol

The SpaceWire - CCSDS Unsegmented Code Transfer Protocol (CUCTP) provides synchronization between Elapsed Time (ET) counters in the local node and remote nodes, by means of SpaceWire Time-Codes and SpaceWire packets. The time format conforms to the CCSDS Unsegmented Code format.

Table 2686. SpaceWire - CCSDS Unsegmented Code Transfer Protocol

Destination Logical Address	Protocol Identifier	CCSDS Unsegmented Code											CRC	EOP
		P-Field		T-Field										
		1st	2nd	Coarse Time						Fine Time				
				2^{31} 2^{24}	2^{23} 2^{16}	2^{15} 2^{8}	2^{7} 2^{0}	2^{-1} 2^{-8}	2^{-9} 2^{-16}	2^{-17} 2^{-24}				*weight*
		0 7	8 15	0 7	8 15	16 23	24 31	32 39	40 47	48 55				*index*
8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits			8 bits	*no. of bits*

SpaceWire Time-Codes are continuously transmitted from a master node to all slave nodes. The transmission of the Time-Codes is synchronized with the local ET counter in the master node. The six bits of the Time-Code time information correspond to six bits of the local ET counter (its exact mapping being programmable by means of register access). The ET bits with lower weights than the size bits mapped to the Time-Code time information bits are all zero at time of Time-Code transmission.

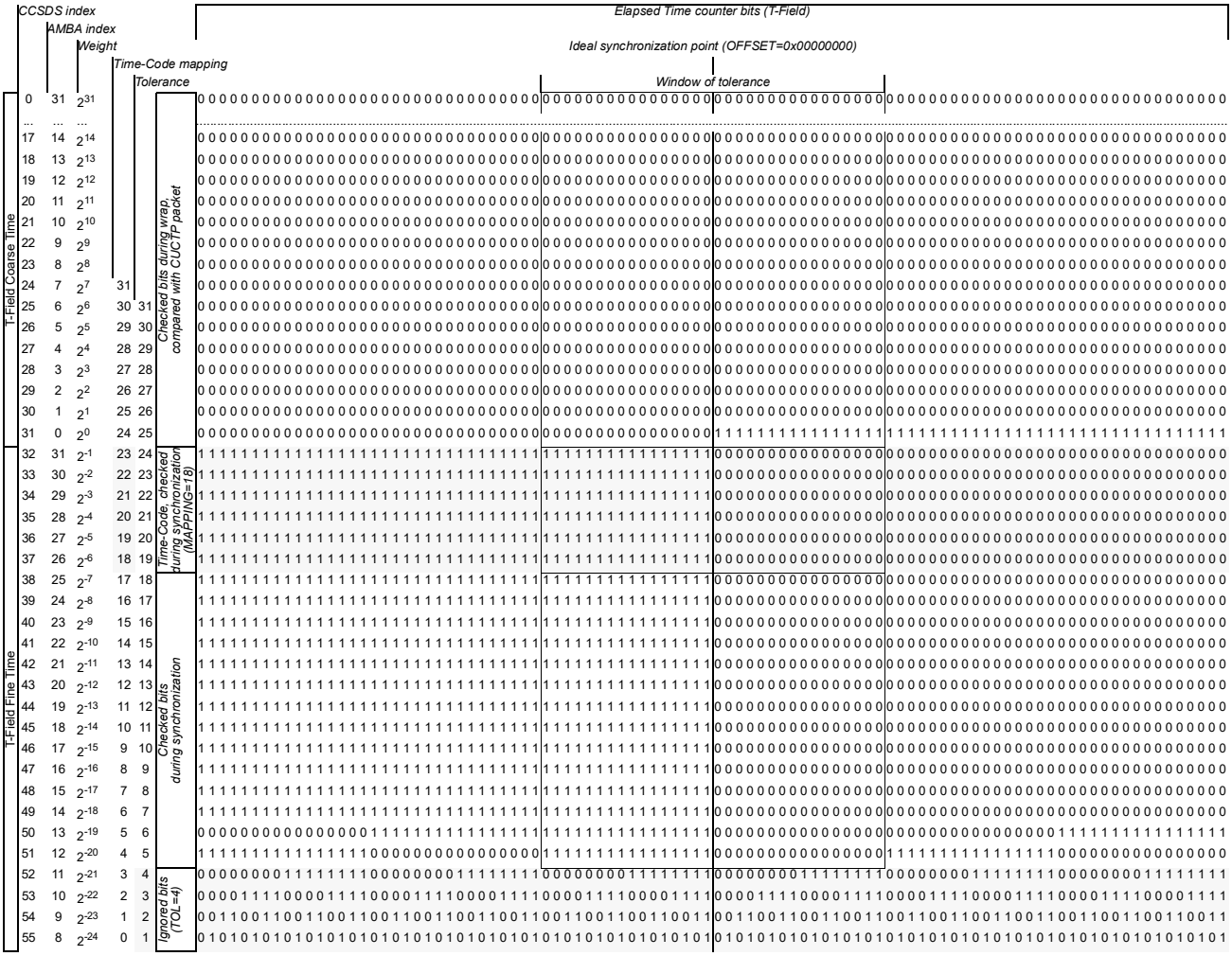
When a Time-Code is received in a slave node, the Time-Code time information is first verified to be an increment of the previously received time-information. The event of the Time-Code reception is assumed to occur synchronously with the local ET counter in the slave node. It is possible to allow a window of tolerance surrounding the exact time at which the reception event was expected. If the event was outside the window of tolerance a tolerance error has occurred and the no further synchronization is performed. Note that the check means that the ET counter bits with weights lower than the six ET bits mapped to the Time-Code time information are checked to have an expected value. This expected value is programmable by means of register access and should be seen as an offset from the all zero value (c.f. Time-Code transmission timing above). The check of the ET counter bits with the lower weights is done with respect to the expected value and the expected value -1 (see further down for the definition of -1). Thus, the ideal point at which a Time-Code can be received is just at the transition to the expected value. The window of tolerance means that a programmable number of ET bits with the lowest weights are ignored in the above comparisons. Note also that the window of tolerance is symmetrical around the expected value and that the -1 subtraction corresponds to the ET bit with the lowest weight that is not ignored due to the window of tolerance.

If the Time-Code reception event was within the window of tolerance, the time information is compared with the corresponding bits in the local ET counter. If there is a match then the two ET counters are considered to be synchronized. If there is a mismatch, then a synchronization error has occurred. Note that during synchronization only the six bits of the ET mapped to the Time-Code time information bits are considered for synchronization. If no Time-Code is received within the window, then the local ET is considered to still be in synchronization but it is freewheeling on Time-Code level.

Additionally, whenever the Time-Code time information wraps from 0x3F to 0x00 it is possible to synchronize the ET bits that have a higher weight than the bits mapped to the Time-Code time information bits. This is performed whenever a new CUCTP packet has been received preceding the reception of the Time-Code with the wrapping time information. If no such packet has been received, then the synchronization will be as described above, but with an increment of the ET bits with the higher weight. If a new and valid CUCTP packet has been received, then in addition to checking the six ET bits, the ET bits with a higher weight are compared with the corresponding ET bits received in the CUCTP packet. If there is a mismatch, then a wrapping error has occurred. If no packet has been received, then the local ET is considered to still be in synchronization but it is freewheeling on packet level.

To summarize, ET bits mapped to the Time-Code time information bits and ET bits with lower weight are checked for every Time-Code received; whilst ET bits with higher weight are checked when ever time information is wrapping. ET bits with lower weight can be offset from the all zero value. ET bits with the lowest weight can be ignored to form a window of tolerance.

Table 2687. Example of synchronization



158.3 Functionality

The interface implements the following functions:

- transmission of SpaceWire Time-Codes
- reception of SpaceWire Time-Codes
- reception of SpaceWire - CCSDS Unsegmented Code Transfer Protocol (CUCTP) packets
- verification of SpaceWire Time-Codes to be received within window of tolerance
- synchronization of SpaceWire Time-Codes time information with local Elapsed Time counter in CCSDS Time Manager (GRCTM) using its external synchronization interface
- synchronization of SpaceWire - CCSDS Unsegmented Code Transfer Protocol (CUCTP) packets with local Elapsed Time counter in CCSDS Time Manager (GRCTM) using its external synchronization interface, occurring on the wrapping of SpaceWire Time-Codes time information
- support for transmission of SpaceWire - CCSDS Unsegmented Code Transfer Protocol (CUCTP) packets

158.3.1 SpaceWire Time-Code transmission

SpaceWire Time-Code transmission is normally performed by a single master node in a system.

SpaceWire Time-Code transmission can be enabled by means of register access. SpaceWire Time-Codes can be transmitted simultaneously on one to four possible outputs, selectable by means of register access.

The six bits of the Time-Code time information are mapped to six bits of the local ET counter, by means of register access. The least significant time information bit [index 0] can be mapped to one of the 32 ET counter bits with the lowest weights.

When Time-Code time information is transmitted, an interrupt is generated (TickTx). If the Time-Code time information is wrapping from 0x3F to 0x00, an interrupt is generated (TickTxWrap).

158.3.2 SpaceWire Time-Code reception

SpaceWire Time-Code reception can be enabled by means of register access. SpaceWire Time-Codes can be received from one of four possible inputs, selectable by means of register access.

When SpaceWire Time-Code reception is enabled, the received Time-Code time information is checked to be an increment of the previously received Time-Code time information. If not, a Time-Code reception error interrupt is generated (TickRxError). If the received Time-Code time information is not an increment, then the received Time-Code time information is stored for the next comparison but is not used for further processing as described hereafter. If the received Time-Code time information is equal to the previously received Time-Code time information, then no action is taken and no interrupts are generated.

The Time-Code control flag can be checked to be “00”, selectable by means of register access. If checking is enabled, control flag differing from “00” will generate a Time-Code reception error interrupt (TickRxError).

If the expected Time-Code time information (and optionally control flags) is received, an interrupt is generated (TickRx). If the Time-Code time information is wrapping from 0x3F to 0x00, an interrupt is generated (TickRxWrap). Both events qualify the Time-Code time information for further processing as described hereafter.

158.3.3 CCSDS Unsegmented Code Transfer Protocol (CUCTP) packet reception

SpaceWire - CCSDS Unsegmented Code Transfer Protocol packets can be automatically received when enabled by means of register access. General SpaceWire communication is performed by the

GRSPW2 SpaceWire codec, which outputs received characters unfiltered to the SPWCUC interface. The SPWCUC interface implements the reception and checking of the CUCTP packet format.

Destination Logical Address (DLA) as per ECSS-E-ST-50-51C. Packets with non-matching DLA are discarded. The DLA is programmable in the range 0x00 to 0xFF by means of register access. The DLA can also be masked with a programmable mask pattern, by means of register access.

The Protocol Identifier as per ECSS-E-ST-50-51C. Packets with non-matching Protocol Identifier are discarded. The Protocol Identifier is programmable in the range 0x00 to 0xFF, as per ECSS-E-ST-50-51C, by means of register access.

Fixed and programmable bits of the P-Field are checked with expected values. The Detail bits for information on the code are fixed to “1111”. The Time code identification is optionally compared to a programmable value, accessible through register access. Reserved bits for future use are checked to be fixed to “000000”. If any of the bits of the received P-Field are incorrect, the packet is discarded and an error interrupt is generated (PktError).

The packet CRC is the same as for RMAP as per ECSS-E-ST-50-52C.

Packets with incorrect CRC are discarded and an error interrupt is generated (PktError).

Packet without an End-Of-Packet (EOP) or an Error-End-of-Packet (EEP) are discarded and an error interrupt is generated (PktError). Too long or too short packets are discarded and an error interrupt is generated (PktError).

A correctly received packet is considered for synchronization, and an interrupt is generated (PktRx). Note that the fully, partially or incorrectly received packet is accessible through register access.

158.3.4 Verification of Time-Codes to be received within tolerance

SpaceWire Time-Code reception can be enabled by means of register access.

The reception time offset is programmable by means of register access. The offset is always counted from the all zero value corresponding to the time at which the Time-Code transmission started. The programmable offset is 32 bit wide and corresponds to the ET bits with the lowest weight.

The window of tolerance is programmable by means of register access. Up to 31 least significant bits can be ignored during Time-Code event comparison with the local ET counter. Note that the window of tolerance is a binary coded window.

If a received Time-Code event is outside the window of tolerance an error interrupt is generated (ToleranceError) and no further synchronization check are attempted for this specific Time-Code. The status will indicate that the local ET is out of synchronization (InSync, FreeSync and FreePkt are cleared).

The window of tolerance should not have a larger width than half the offset. Note that bits in the offset can be masked and ignored by the window of tolerance and will then not be used in the comparison.

If no Time-Code is received within the window of tolerance, then the local ET is considered to still be in synchronization (InSync set) but it is freewheeling on Time-Code level (FreeSync set).

158.3.5 Synchronization via Time-Codes

Synchronization by means of Time-Code reception can be enabled by means of register access, and occurs on reception Time-Codes the window of tolerance as described above.

If the check passes, the status indicates that the local ET is in synchronization (InSync set) and an interrupt is generated (Sync).

If the check fails, an error interrupt is generated (SyncError). The status indicates that the local ET is out of synchronization (InSync, FreeSync and FreePkt are cleared).

If the received Time-Code time information was wrapping and no new CUCTP packet was previously received, then the status indicates that the local ET is freewheeling on packet level (FreePkt set).

158.3.6 Synchronization via CUCTP packets

Synchronization by means of CUCTP packets can be enabled by means of register access, and occurs on reception Time-Codes the window of tolerance as described above when the time information is wrapping. Only correctly received and checked CUCTP packet are considered for synchronization, which have bit 15 in the P-Field cleared (meaning synchronization).

If the check passes, the status indicates that the local ET is in synchronization (InSync set) and an interrupt is generated (Wrap).

If the check fails, an error interrupt is generated (WrapError). The status indicates that the local ET is out of synchronization (InSync, FreeSync and FreePkt are cleared).

158.3.7 Initialization via CUCTP packets

Initialization by means of CUCTP packets can be enabled by means of register access, and occurs on reception Time-Codes the window of tolerance as described above when the time information is wrapping. Only correctly received and checked CUCTP packet are considered for synchronization, which have bit 15 in the P-Field set (meaning initialization).

No other checks are performed. The status indicates that the local ET is in synchronization (InSync set) and an interrupt is generated (Init).

158.3.8 CCSDS Unsegmented Code Transfer Protocol (CUCTP) packet transmission support

CUCTP packet transmission is normally performed by a single master node in a system.

When a Time-Code has been transmitted with time information that is wrapping from 0x3F to 0x00, an interrupt is generated (TickTxWrap) which can be used as a starting point for software controlled generation and transmission of CUCTP packets. The next T-Field to be sent in the CUCTP packet can be read out by means of register access.

158.4 Data formats

All Elapsed Time (ET) information is compliant with the CCSDS Unsegmented Code defined in [CCSDS] and repeated hereafter.

158.4.1 Numbering and naming conventions

Convention according to the CCSDS recommendations, applying to time structures:

- The most significant bit of an array is located to the left, carrying index number zero.
- An octet comprises eight bits.

Table 2688. CCSDS n-bit field definition

CCSDS n-bit field		
most significant		least significant
0	1 to n-2	n-1

Convention according to AMBA specification:

- The least significant bit of an array is located to the right, carrying index number zero.
- Big-endian support.

Table 2689. AMBA n-bit field definition

AMBA n-bit field		
most significant		least significant
n-1	n-2 down to 1	0

158.4.2 Reference documents

[CCSDS] Time Code Formats, CCSDS 301.0-B-4, www.ccsds.org

[SPW] Space engineering: SpaceWire - Links, nodes, routers and networks, ECSS-E-ST-50-12C

[PID] Space engineering: SpaceWire protocol identification, ECSS-E-ST-50-51C

[RMAP] Space engineering: SpaceWire - Remote memory access protocol, ECSS-E-ST-50-52C

158.4.3 CCSDS Unsegmented Code: Preamble Field (P-Field)

The time code preamble field (P-Field) may be either explicitly or implicitly conveyed. If it is implicitly conveyed (not present with T-Field), the code is not self-identified, and identification must be obtained by other means. As presently defined, the explicit representation of the P-Field is limited to one octet whose format is described hereafter.

Table 2690. CCSDS Unsegmented Code P-Field definition

Bit	Value		Interpretation
0	"1"		Extension flag, P-Field extended with 2nd octet
1 - 3	"001"	1958 January 1 epoch (Level 1)	Time code identification
	"010"	Agency-defined epoch (Level 2)	
4 - 5	"11"	(number of octets of coarse time) - 1	Detail bits for information on the code
6 - 7	"11"	(number of octets of fine time)	
8	"0"		Extension flag, P-Field not extended with 3rd octet
9-14	"000000"	Don't care	Reserved for future use
15	"1"=Initialize,"0"=Synchronize		Initialization

Note: Revision 0 of the core used bit 8 for Initialize/Synchronize selection. Bit 15 is used from Revision 1 and forward, since bit 8 has the meaning of an extension flag for a third octet.

158.4.4 CCSDS Unsegmented Code: Time Field (T-Field)

For the unsegmented binary time codes described herein, the T-Field consists of a selected number of contiguous time elements, each element being one octet in length. An element represents the state of 8 consecutive bits of a binary counter, cascaded with the adjacent counters, which rolls over at a modulo of 256.

Table 2691. CCSDS Unsegmented Code T-Field definition

CCSDS Unsegmented Code									
Preamble Field	Time Field								
	Coarse time						Fine time		
-	2^{31} 2^{24}	2^{23} 2^{16}	2^{15} 2^{8}	2^{7} 2^{0}			2^{-1} 2^{-8}	2^{-9} 2^{-15}	2^{-16} 2^{-24}
0:15	0			31			32		55

The basic time unit is the second. The T-Field consists of 32 bits of coarse time (seconds) and 24 bits of fine time (sub seconds). The coarse time code elements are a count of the number of seconds elapsed from the epoch. The 32 bits of coarse time results in a maximum ambiguity period of approximately 136 years. Arbitrary epochs may be accommodated as a Level 2 code. The 24 bits of fine code elements result in a resolution of 2-24 second (about 60 nanoseconds). This code is not UTC-based and leap second corrections do not apply according to CCSDS.

158.5 Registers

The core is programmed through registers mapped into APB address space.

*Table 2692.*Registers

APB address offset	Register
0x00	Configuration Register
0x04	Status Register
0x08	Control Register
0x10	Destination Logical Address and Mask Register
0x14	Protocol Identifier Register
0x18	Offset Register
0x20	T-Field Coarse Time Packet Register
0x24	T-Field Fine Time Packet Register
0x28	P-Field Packet and CRC Packet Register
0x30	Elapsed Time Coarse Register
0x34	Elapsed Time Fine Register
0x38	Next Elapsed Time Coarse Register
0x3C	Next Elapsed Time Fine Register
0x60	Pending Interrupt Masked Status Register
0x64	Pending Interrupt Masked Register
0x68	Pending Interrupt Status Register
0x6C	Pending Interrupt Register
0x70	Interrupt Mask Register
0x74	Pending Interrupt Clear Register

158.5.1 Various T-Field Mappings

Table 2693. Various T-Field mappings

| T-Field Coarse Time Registers (AMBA index numbering) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | T-Field Fine Time Registers (AMBA index numbering) | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---

158.5.3 Configuration Register

*Table 2695.*0x00 - CONF - Configuration Register

31 28	27 26	25 24	23 21	20 16	15 13	12 8	7	6 4	3 2	1	0
SELOUT	R	SELIN	R	MAPPING	R	TOL	R	TID	R	CTF	CP
0	0	0	0	*	0	*	0	0b001	0	0	0
rw	r	rw	r	rw	r	rw	r	rw	r	rw	rw

31: 28	SELOUT	Select output(s) for SpaceWire time-code transmission, index 3 downto 0.
27: 26	RESERVED	
25: 24	SELIN	Select input for SpaceWire time-code and packet reception, one of 0 through 3.
23: 21	RESERVED	
20: 16	MAPPING	Defines mapping of Time-Code time information versus T-Field: T-Field definition [0 to 55], where 0 is MSB and 55 is LSB, 0 corresponds to 2**31 seconds, and 55 corresponds to 2**-24 seconds. MAPPING= 0 corresponds to Time-Code[5] mapped to T-Field[50] and Time-Code[0] mapped to T-Field[55], MAPPING= 1 corresponds to Time-Code[5] mapped to T-Field[49] and Time-Code[0] mapped to T-Field[54], ... MAPPING=30 corresponds to Time-Code[5] mapped to T-Field[20] and Time-Code[0] mapped to T-Field[25]. MAPPING=31 corresponds to Time-Code[5] mapped to T-Field[19] and Time-Code[0] mapped to T-Field[24].
15: 13	RESERVED	
12: 8	TOL	Defines SpaceWire Time-Code reception tolerance: T-Field definition [0 to 55], where 0 is MSB and 55 is LSB, 0 corresponds to 2**31 seconds, and 55 corresponds to 2**-24 seconds. TOL= 0 corresponds to zero tolerance, TOL= 1 corresponds to T-Field[55] being ignored, TOL= 2 corresponds to T-Field[54:55] being ignored, ... TOL=30 corresponds to T-Field[26:55] being ignored, TOL=31 corresponds to T-Field[25:55] being ignored.
7	RESERVED	
6: 4	TID	Defines CUC P-Field Time Code Identification: “001” 1958 January 1 epoch (Level 1) “010” Agency-defined epoch (Level 2)
3: 2	RESERVED	
1	CTF	Check SpaceWire Time-Code control flags to be all zero when set.
0	CP	Check CUC P-Field Time Code Identification to match TID field when set.

Power-up default: depends on configuration

158.5.4 Status Register

Table 2696. 0x04 - STAT - Status Register

31 11	10	9	8	7 6	5 0
RESERVED	FREEPKT	FREESYNC	INSYNC	TIMEFLAG	TIMEINFO
0	0	0	0	0	0
r	r	r	r	r	r

31: 11	RESERVED	
10	FREEPKT	Time freewheeling on packet level
9	FREESYNC	Time freewheeling on time-code level
8	INSYNC	Synchronous time reception of packets and time-codes
7: 6	TIMEFLAG	Received SpaceWire Time-Code Control Flags
5: 0	TIMEINFO	Received SpaceWire Time-Code Time Information

158.5.5 Control Register

Table 2697. 0x08 - CTRL - Control Register

31 6	5	4	3	2	1	0
RESERVED	PKTRXEN	PKTINITEN	PKTSYNCEN	RXEN	TXEN	RESET
0	0	0	0	0	0	0
r	rw	rw	rw	rw	rw	rw

31: 6	RESERVED	
5	PKTRXEN	Enable SpaceWire CUC packet reception
4	PKTINITEN	Enable SpaceWire CUC packet initialization
3	PKTSYNCEN	Enable SpaceWire CUC packet synchronization
2	RXEN	Enable SpaceWire Time-Code reception
1	TXEN	Enable SpaceWire Time-Code transmission
0	RESET	Reset core

Power-up default: 0x00000000

158.5.6 Destination Logical Address and Mask Register

Table 2698. 0x10 - DLA - Destination Logical Address and Mask Register

31 16	15 8	7 0
RESERVED	MASK	DLA
0	0	*
r	rw	rw

31: 16	RESERVED	
15: 8	MASK	Destination Logical Address Mask, ignore bit when set. Zero at reset.
7: 0	DLA	Destination Logical Address

Power-up default: configuration dependent

158.5.7 Protocol Identifier Register

*Table 2699.*0x14 - PIR - Protocol Identifier Register

31	8	7	0
RESERVED		PID	
0		*	
r		rw	

31: 8 RESERVED

7: 0 PID Protocol Identifier

Power-up default: configuration dependent

158.5.8 Offset Register

*Table 2700.*0x18 - OFFS - Offset Register

31	24	23	0
OFFSET			
0			
rw			

2^{7} 2^{0} 2^{-1} 2^{-24}

24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55

31: 0 OFFSET Packet reception offset: 0 corresponds to LSB of T-Field.

Power-up default: 0x00000000

158.5.9 T-Field Coarse Time Packet Register

*Table 2701.*0x20 - TCTP - T-Field Coarse Time Packet Register

31	0
T-Field, Coarse Time	
0	
rw	

2^{31} 2^{0}

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

31: 0 COARSE T-Field of received packet

158.5.10T-Field Fine Time Packet Register

*Table 2702.*0x24 - TFTP - T-Field Fine Time Packet Register

31	8	7	0
T-Field, Fine Time		RESERVED	
0		0	
rw		r	

2^{-1} 2^{-24}

32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55

31: 8 FINE T-Field of received packet

7: 0 RESERVED

158.5.11P-Field Packet and CRC Packet Register

*Table 2703.*0x28 - PPCP - P-Field Packet and CRC Packet Register

31	30	29	28 24	23 16	15 0
NEW	INIT	FORCE	R	CRC	P-Field
0	0	0	0	0	0
rw	rw	rw	r	rw	rw
				b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

31	NEW	New packet received
30	INIT	Initialization bit set in received packet
29	FORCE	Force packet initialization at once, one shot (mostly for debug)
28: 24	RESERVED	
23: 16	CRC	CRC of received packet
15: 0	P-Field	P-Field of received packet

158.5.12Elapsed Coarse Time Register

*Table 2704.*0x30 - ECT - Elapsed Coarse Time Register

31 0
T-Field, Coarse Time
*
r
2^{31} 2^{0}

31: 0	COARSE	T-Field of current time (latches Fine Time when read)

158.5.13Elapsed Fine Time Register

*Table 2705.*0x34 - EFT - Elapsed Fine Time Register

31 8	7 0
T-Field, Fine Time	RESERVED
0	0
r	r
2^{-1} 2^{-24}	

31: 8	FINE	T-Field of current time
7: 0	RESERVED	

158.5.14Next Elapsed Coarse Time Packet Register

*Table 2706.*0x38 - NECT - Next Elapsed Coarse Time Packet Register

31 0
T-Field, Coarse Time
*
r
2^{31} 2^{0}

31: 0	COARSE	T-Field of next packet to be sent

158.5.15Next Elapsed Fine Time Packet Register

*Table 2707.*0x3C - NEFT - Next Elapsed Fine Time Packet Register

31	8	7	0
T-Field, Fine Time		RESERVED	
*		0	
r		r	
2^{-1}	2^{-24}		

31: 8	FINE	T-Field of next packet to be sent
7: 0	RESERVED	

158.5.16

Interrupt registers

The interrupt registers give complete freedom to the software, by providing means to mask interrupts, clear interrupts, force interrupts and read interrupt status.

When an interrupt occurs the corresponding bit in the Pending Interrupt Register is set. The normal sequence to initialize and handle a module interrupt is:

- Set up the software interrupt-handler to accept an interrupt from the module.
- Read the Pending Interrupt Register to clear any spurious interrupts.
- Initialize the Interrupt Mask Register, unmasking each bit that should generate the interrupt.
- When an interrupt occurs, read the Pending Interrupt Status Register in the software interrupt-handler to determine the causes of the interrupt.
- Handle the interrupt, taking into account all causes of the interrupt.
- Clear the handled interrupt using Pending Interrupt Clear Register.

Masking interrupts: After reset, all interrupt bits are masked, since the Interrupt Mask Register is zero. To enable generation of a module interrupt for an interrupt bit, set the corresponding bit in the Interrupt Mask Register.

Clearing interrupts: All bits of the Pending Interrupt Register are cleared when it is read or when the Pending Interrupt Masked Register is read. Reading the Pending Interrupt Masked Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register. Selected bits can be cleared by writing ones to the bits that shall be cleared to the Pending Interrupt Clear Register.

Forcing interrupts: When the Pending Interrupt Register is written, the resulting value is the original contents of the register logically OR-ed with the write data. This means that writing the register can force (set) an interrupt bit, but never clear it.

Reading interrupt status: Reading the Pending Interrupt Status Register yields the same data as a read of the Pending Interrupt Register, but without clearing the contents.

Reading interrupt status of unmasked bits: Reading the Pending Interrupt Masked Status Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register, but without clearing the contents.

The interrupt registers comprise the following:

*Table 2708.*Interrupt registers

Description	Name	Mode
Pending Interrupt Masked Status Register	SPWCUCPIMSR	r
Pending Interrupt Masked Register	SPWCUCPIMR	r
Pending Interrupt Status Register	SPWCUCPISR	r
Pending Interrupt Register	SPWCUCPIR	w
Interrupt Mask Register	SPWCUCIMR	w
Pending Interrupt Clear Register	SPWCUCPICR	w

*Table 2709.*Interrupt registers template

31 13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	TickTX	TickTx Wrap	TickRx	TickRx Wrap	TickRx Error	Toler-ance Error	Sync	Sync Error	Wrap	Wrap Error	Pkt Rx	Pkt Error	Pkt Init
	0	0	0	0	0	0	0	0	0	0	0	0	0
	*	*	*	*	*	*	*	*	*	*	*	*	*

31: 13	RESERVED	
12	TickTx	Time-code transmission, incrementing time information*
11	TickTxWrap	Time-code transmission, wrapping time information*
10	TickRx	Time-code reception, incrementing time information*
9	TickRxWrap	Time-code reception, wrapping time information*
8	TickRxError	Error in received time-code time information*
7	ToleranceError	Time-code reception timing outside tolerance with respect to local Elapsed Time*
6	Sync	Time-information synchronized correctly with respect to local Elapsed Time*
5	SyncError	Error in time-information synchronization with respect to local Elapsed Time*
4	Wrap	Packet synchronized correctly with respect to local Elapsed Time*
3	WrapError	Error in packet synchronization with respect to local Elapsed Time*
2	PktRx	Packet received correctly*
1	PktError	Error in received packet (e.g. CRC, P-Field, EEP, etc.)*
0	PktInit	Initialization of local Elapsed Time through received packet or forced access*
	All bits in all interrupt registers are reset to 0b after reset	

*See table 2708.

158.6 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x089. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

158.7 Implementation

158.7.1 Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

158.8 Configuration options

Table 2710 shows the configuration options of the core (VHDL generics).

*Table 2710.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the core.	0 - NAHBIRQ-1	0
dlareset	Destination Logical Address reset value	0-255	254
pidreset	Protocol Identifier reset value	0-255	254
mapreset	Index of least significant bit for mapping in SpaceWire Time-Code Time information. LSB is marked as 0.	0-31	18
tolreset	Number of least significant bits that tolerate errors. LSB is marked as 1, whereas 0 allows no tolerance.	0-31	8

158.9 Signal descriptions

Table 2711 shows the interface signals of the core (VHDL ports).

*Table 2711.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
CUCI	TickInDone	Input	SpaceWire Time-Code input processed	High
	TickOutRaw		SpaceWire Time-Code output event	High
	TimeOut[7:0]		SpaceWire Time-Code output	-
	RxDAv		SpaceWire character event	High
	RxDataOut[8:0]		SpaceWire character	-
	ElapsedTime[0:55]		Elapsed Time	
	ElapsedNext[0:55]		Next Elapsed Time	
	ElapsedEvent		Elapsed Time counter event	High
	ElapsedSync		Elapsed Time counter synchronization event	High
CUCO	TickInRaw	Output	SpaceWire Time-Code input request	High
	TimeIn[7:0]		SpaceWire Time-Code input	-
	ExternalTime[0:55]		External Elapsed Time input	-
	ExternalSync		External Elapsed Time input synchronization	High

* see GRLIB IP Library User's Manual

158.10 Signal definitions and reset values

The core has no external signals.

158.11 Timing

The core has no external timing.

158.12 Library dependencies

Table 2712 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2712.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	SPACEWIRECUC	Signals, component	Component declarations, signals.

159 GRPW - PacketWire Interface

The *PacketWire to AMBA AHB Interface* (GRPW) comprises a bi-directional PacketWire link and an AMBA AHB master interface. The purpose of the interface is to allow read and write accesses on an AMBA AHB bus to be initiated from the PacketWire interface. The protocol allows single or multiple reads per command, each command specifying a read or write access, the number of word transfers and the starting address. For a write access, word oriented data is transmitted to the interface, and for read accesses word oriented data is received from the interface.

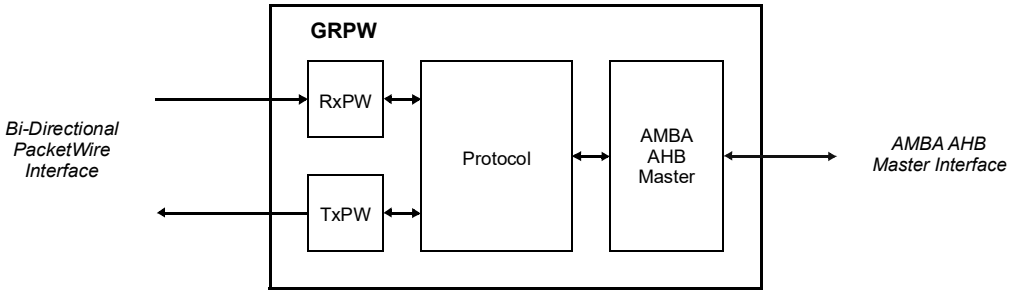


Figure 390. Block diagram

In a typical application, the PacketWire interface would be used as a remote control interface of a System-On-a-Chip device based around the AMBA AHB bus. An example could be a Packet Telemetry and Telecommand device which can be controlled remotely from a processor board via a PacketWire link. The link would provide both the capability to read and write registers, and to make block transfers to and from the target device and its memories.

This interface is based on the de facto standard PacketWire interface used by the *European Space Agency* (ESA). At the time of writing there were no relevant documents available from the *European Cooperation for Space Standardization* (ECSS).

159.1 Operation

159.1.1 Protocol

The communication protocol is based on the protocol used in the LEON processor. Commands are sent to the interface as messages over the bi-directional PacketWire interface. The protocol allows read and write accesses, as shown in table 2713. For each command, the number of 32-bit words to be transferred are specified, ranging from 1 to 64 words. For each command access, a 32-bit starting byte address is specified.

All transfers are assumed to be word aligned, effectively ignoring the two least significant bits of the address, assuming them to be both zero. There are no restrictions on the address, allowing a wrap around at the end of the address space during a transfer. The start address can thus be set to any position in the address space. The address is automatically incremented by 4 after each word access during

a transfer. The address and data bit numbering in table 2713 correspond to the AMBA AHB bit numbering conventions.

*Table 2713.*Protocol on PacketWire side

	Control	**Address**				**Data**												
						first word				**second word**				**...**	**last word**			
Cmd	Write																	
Octet	0	1	2	3	4	5	6	7	8	9	10	11	12	...	n-4	n-3	n-2	n-1
Send	11_b Length-1	31:24	23:16	15:8	7:0	31:24	23:16	15:8	7:0	31:24	23:16	15:8	7:0	...	31:24	23:16	15:8	7:0
Byte						0	1	2	3	4	5	6	7	...	n-4	n-3	n-2	n-1
Receive																		
Cmd	Read																	
Octet	0	1	2	3	4	5	6	7	8	9	10	11	12	...	n-4	n-3	n-2	n-1
Send	10_b Length-1	31:24	23:16	15:8	7:0													
Byte						0	1	2	3	4	5	6	7	...	n-4	n-3	n-2	n-1
Receive						31:24	23:16	15:8	7:0	31:24	23:16	15:8	7:0	...	31:24	23:16	15:8	7:0

159.1.2 Bi-directional PacketWire interface

The bi-directional PacketWire interface comprises two PacketWire links, one in each direction, the PacketWire input link and the PacketWire output link. Each link comprises three ports for transmitting the message delimiter, the bit clock and the serial bit data. Each link also comprises an additional port for busy signalling, indicating when the receiver is ready to receive the next octet.

The interface accepts and generates the waveform format shown in figure 391.

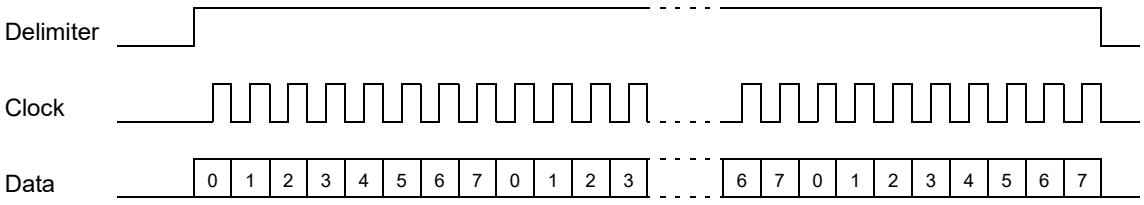


Figure 391. Synchronous bit serial waveform

The PacketWire protocol follows the CCSDS transmission convention, the most significant bit being sent first, both for octet transfers (control), and for word transfer (address or data). Transmitted data should consist of multiples of eight bits otherwise the last bits will be lost. The input message delimiter port is used to delimit messages (commands). It should be asserted while a message is being input, and deasserted in between. In addition, the message delimiter port should define the octet boundaries in the data stream, the first octet explicitly and the following octets each subsequent eight bit clock cycles.

The maximum receiving input baud rate is defined as twice the frequency of the system clock input (f_{HCLK}). The maximum receiving throughput is limited by the AMBA AHB system into which this core is integrated. There is no lower limit for the input baud rate in the receiver. Note however that there are constraints on the input baud rate related to the automatic baud rate detection, as described hereafter.

The output baud rate is automatically adjusted to the incoming baud rate, provided that the incoming baud rate is less than half the frequency of the system clock input (f_{HCLK}). The lower limit for the input baud rate detection is $f_{HCLK}/512$. If the input baud rate is less than this limit, the output baud rate will equal $f_{HCLK}/512$. The input baud rate is determined by measuring the width of the logical one phase of the input bit clock.

The handshaking between the PacketWire links and the interface is implemented with busy ports, one in each direction. When a message is sent, the busy signal on the PacketWire input link will be asserted as soon as the first data bit is detected, it will then be deasserted as soon as the interface is ready to receive the next octet. This gives the transmitter ample time to stop transmitting after the completion of the first octet and wait for the busy signal deassertion before starting the transmission of the next octet. The handshaking is continued through out the message. At the end of message, the busy signal will be asserted until the completion of the message. For a write command, the busy signal will be deasserted after the completion of the AMBA AHB write access of the last word. For a read command, the busy signal will be deasserted after the completion of the AMBA AHB read access of the last word and its transmission on the PacketWire output link. It is therefore not possible for the external transmitter to send a new command until the previous has been completed.

Illegal commands are prevented from being executed. An illegal command is defined as a control octet for which the two most significant bits are neither 11_b nor 10_b. No accesses to the AMBA AHB bus will be performed and no response will be generated on the PacketWire link for a read command. A new command will be accepted as soon as the input message delimiter has been deasserted and a new command is transmitted.

A command can be aborted by prematurely deasserting the message delimiter on the PacketWire input link. This can be done at any point of the message, e.g. during the control octet, during the address or during the data transfer for write accesses. An aborted message will immediately terminate the access on the AMBA AHB bus. Note that it is not possible to predict whether or not the last word write access to the AMBA AHB bus has been completed or not in the case the message is aborted.

It is not possible to determine whether or not an access has been successfully completed on the AMBA AHB bus. For read accesses, a data response will be generated on the PacketWire output link independently of whether the AMBA AHB access was terminated with an *OKAY* or *ERROR*.

In the case an AMBA AHB access is not terminated because of indefinite *RETRY* or *SPLIT* responses, the command will not be completed and the busy port on the PacketWire input link will not be deasserted. This locked state can be observed by monitoring the response on the PacketWire input link, for which the busy signal will not be deasserted. For read accesses, this locked state can also determine if no data is received on the PacketWire output link. To overcome this locked stated, the message delimiter should be firstly deasserted on the PacketWire input linked. The message delimiter should then be asserted and a new control octet should then be transmitted, even though the busy port is asserted on the PacketWire input link. This action will abort any AMBA AHB accesses and restore the state of the interface. The newly started message should then be completed using the handshake method previously described. Note that it is not possible to determine at what time instant the abort will occur, possibly ruining the on going access. This is however acceptable considering being a recovery from a locked state.

159.1.3 AMBA AHB master interface

The AMBA AHB master interface has been reduced in functionality to support only what is required for the core. The following AMBA AHB features are constrained:

- only generates ***HSIZE*** = *HSIZE_WORD*
- only generates ***HLOCK*** = 0_b
- only generates ***HPROT*** = 0000_b
- only generates ***HBURST*** = *HBURST_SINGLE*
- never generates ***HTRANS*** = *HTRANS_BUSY*
- both ***HRESP*** = *HRESP_OKAY* and ***HRESP*** = *HRESP_ERROR* are treated as a successfully completed access
- both ***HRESP*** = *HRESP_RETRY* and ***HRESP*** = *HRESP_SPLIT* will result in a rescheduling the previous access until terminated with *HRESP_OKAY* or *HRESP_ERROR*

- only big-endianness is supported

The interface can act as a default AHB master generating idle accesses when required. It only implements a single word access at a time, without bursts, to reduce complexity for retry and split handling, and not requiring the 1024 byte boundary imposed by the AMBA specification on burst transfers to be taken into account.

159.1.4 Advanced Microcontroller Bus Architecture

Convention according to the Advanced Microcontroller Bus Architecture (AMBA) Specification, applying to the AHB and APB interfaces:

- Signal and port names are in upper case, except for the following:
- A lower case '*n*' in the name indicates that the signal or port is active low.
- Constant names are in upper case.
- The *least* significant bit of an array is located to the *right*, carrying index number zero.

*Table 2714.*AMBA n-bit field definition

AMBA n-bit field		
most significant		least significant
n-1	n-2 down to 1	0

159.1.5 Consultative Committee for Space Data Systems

Convention according to the Consultative Committee for Space Data Systems (CCSDS) recommendations, applying to all relevant structures:

- The *most* significant bit of an array is located to the *left*, carrying index number zero, and is transmitted first.
- An octet comprises eight bits.

General convention, applying to signals, ports and interfaces:

- Signal or port names are in mixed case.
- An upper case '_*N*' suffix in the name indicates that the signal or port is active low.

*Table 2715.*CCSDS n-bit field definition

CCSDS n-bit field		
most significant		least significant
0	1 to n-2	n-1

159.2 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x032. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

159.3 Implementation

159.3.1 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual).

159.4 Configuration options

Table 2716 shows the configuration options of the core (VHDL generics).

*Table 2716.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
syncreset	Synchronous reset when set, else asynchronous	0 - 1	0

159.5 Signal descriptions

Table 2717 shows the interface signals of the core (VHDL ports).

*Table 2717.*Signal descriptions

Signal name	Field	Type	Function	Comment	Active
HRESETn	N/A	Input	Reset		Low
HCLK	N/A	Input	Clock		-
PWI	VALID	Input	Delimiter	This input port is the message delimiter for the input interface. It should be deasserted between messages	High
	CLOCK		Bit clock	This input port is the PacketWire bit clock. The receiver registers are clocked on the rising ***PWI.Clk*** edge.	Rising
	DATA		Data	This input port is the serial data input for the interface. Data are sampled on the rising ***PWI.Clk*** edge when ***PWI.Valid*** is asserted.	-
	BUSY_N		Not ready for octet	This input port indicates whether the receiver is ready to receive one octet. The input is considered as asynchronous.	Low
PWO	VALID	Output	Delimiter	This output port is the packet delimiter for the output interface. It is deasserted between packets. The output is clocked out on the rising ***HCLK*** edge.	High
	CLOCK		Bit clock	This output port is the PackeWire output bit clock. The output is clocked out on the rising ***HCLK*** edge.	Rising
	DATA		Data	This output port is the serial data output for the interface. The output is clocked out on the rising ***HCLK*** edge.	-
	BUSY_N		Not ready for octet	This port indicates whether the receiver is ready to receive one octet. The output is clocked out on the rising ***HCLK*** edge.	Low
AHBI	*	Input	AMB master input signals		-
AHBO	*	Output	AHB master output signals		-

* see GRLIB IP Library User's Manual

159.6 Signal definitions and reset values

The signals and their reset values are described in table 2718.

*Table 2718.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
pwi_valid	Input	Delimiter	High	-
pwi_clock	Input	Bit clock	Rising	-
pwi_data	Input	Data	-	-
pwo_busy_n	Output	Not ready for octet	Low	Logical 0
pwo_valid	Output	Delimiter	High	Logical 0
pwo_clock	Output	Bit clock	Rising	Logical 0
pwo_data	Output	Data	-	Logical 0
pwi_busy_n	Input	Not ready for octet	Low	-

159.7 Timing

The timing waveforms and timing parameters are shown in figure 392 and are defined in table 2719.

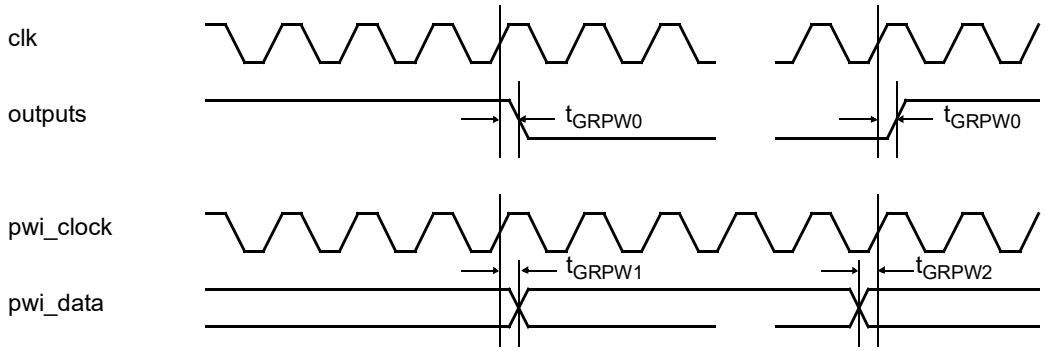


Figure 392. Timing waveforms

*Table 2719.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
t_{GRPW0}	clock to output delay	rising *clk* edge	TBD	TBD	ns
t_{GRPW1}	input to clock hold	rising *pwi_clock* edge	TBD	-	ns
t_{GRPW2}	input to clock setup	rising *pwi_clock* edge	TBD	-	ns
t_{GRPW3}	pwi_valid to pwi_clock edge	rising *pwi_clock* edge	TBD	-	ns
t_{GRPW3}	pwi_valid de-asserted period	-	TBD*t_{CLK}		periods

Note: The *pwi_busy_n* input is re-synchronized inside the core. The signal does not have to meet any setup or hold requirements.

159.8 Library dependencies

Table 2720 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2720.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_Types	Signals, component	Component declaration

159.9 Instantiation

The core is an almost fully synchronous design based on a single system clock strategy. The asynchronous part is related to the PacketWire (PW) input interface, for which the receiving shift register is implemented as a separate clock domain. All signals going between clock domains are clocked twice before being used to reduce the risk for metastability.

All registers in the core are reset asynchronously. The reset input can be asserted asynchronously, but requires synchronous deassertion to avoid any recovery time violations.

The ***PWI.Valid*** input should be deasserted for at least 4 ***HCLK*** clock periods between messages. The ***PWI.Data*** input is clocked into a receiving shift register on the rising ***PWI.Clk*** edge. The ***PWI.Clk*** input should have a 50% duty cycle.

The ***PWI.Busy_N*** should be asserted as soon as possible by the receiver, allowing the transmitter to halt the transmission between octets. The input is synchronised using two registers clocked on the rising ***HCLK*** edge.

This example shows how the core can be instantiated.

```
library  IEEE;
use      IEEE.Std_Logic_1164.all;
library  GRLIB;
use      GRLIB.AMBA.all;
library  TMTC;
use      TMTC.TMTC_Types.all;
..
   component GRPW is
      generic(
         hindex:           in    Integer := 0);
      port(
         -- AMBA AHB System Signals
         HCLK:             in    Std_ULogic; -- system clock
         HRESETn:          in    Std_ULogic; -- synchronised reset
         -- AMBA AHB Master Interface
         AHBOut:           out   AHB_Mst_Out_Type;
         AHBIn:            in    AHB_Mst_In_Type;
         -- PacketWire interface
         PWI:              in    GRPW_In_Type;
         PWO:              out   GRPW_Out_Type);
   end component GRPW;
```

160 GRPWRX - PacketWire Receiver

160.1 Overview

The PacketWire Receiver implements a receiver function with Direct Memory Access (DMA) support. Packets (or blocks of data, normally CCSDS Space Packets) are automatically stored to memory, for which the user configures a descriptor table with descriptors that point to each individual packet or one or more packets stored in a fixed length fields (framing mode).

The core provides the following external and internal interfaces:

- Packet Wire interface (serial bit data, bit clock, packet delimiter, abort, ready, busy)
- AMBA AHB master interface, with sideband signals as per [GRLIB]
- AMBA APB slave interface, with sideband signals as per [GRLIB]

The operation of the receiver is highly programmable by means of control registers.

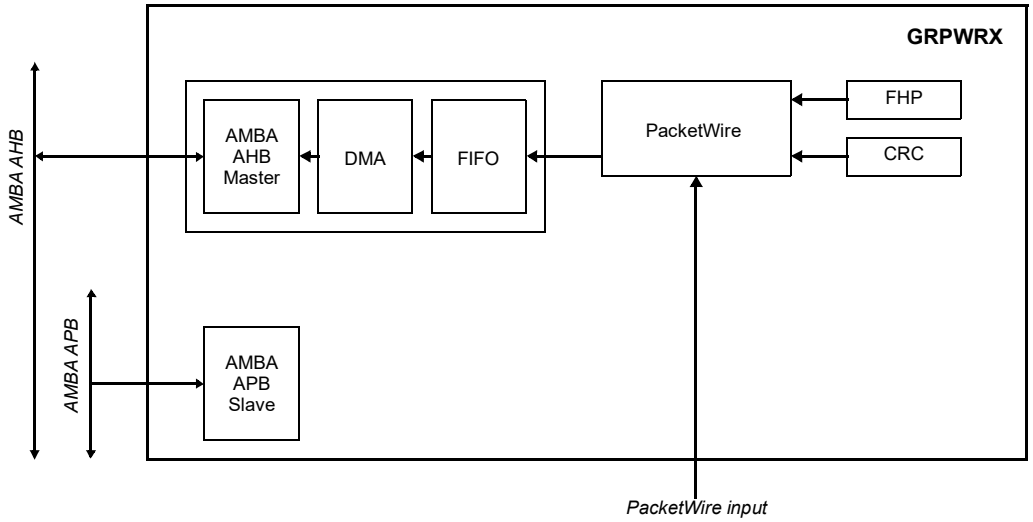


Figure 393. Block diagram

160.2 PacketWire interface

A PacketWire link comprises four ports for transmitting the message delimiter, the bit clock, the serial bit data and an abort signal. A link also comprises additional ports for busy signalling, indicating when the receiver is ready to receive the next octet, and for ready signalling, indicating that the receiver is ready to receive a complete packet. The waveform format shown in figure 394.

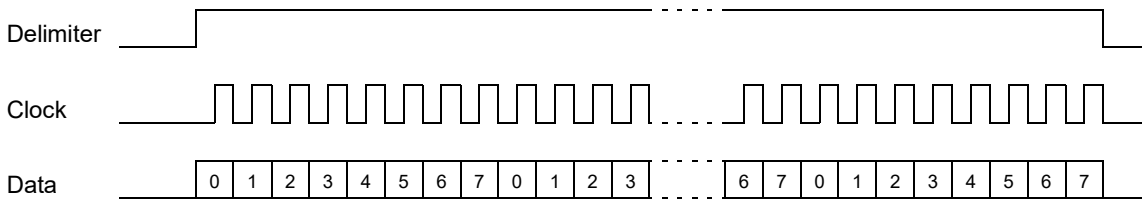


Figure 394. Synchronous bit serial waveform

The PacketWire protocol follows the CCSDS transmission convention, the most significant bit being sent first, both for octet transfers (control), and for word transfer (address or data). Transmitted data

should consist of multiples of eight bits otherwise the last bits will be lost. The message delimiter port is used to delimit messages (commands). It should be asserted while a message is being input, and deasserted in between. In addition, the message delimiter should define the octet boundaries in the data stream, the first octet explicitly and the following octets each subsequent eight bit clock cycles. The delimiter should be de-asserted for at least eight bit periods between messages.

The handshaking between the PacketWire link and the interface is implemented with a busy port. When a message is sent, the busy signal on the PacketWire link will be asserted as soon as the first data bit is detected, it will then be deasserted as soon as the interface is ready to receive the next octet. This gives the transmitter ample time to stop transmitting after the completion of the first octet and wait for the busy signal deassertion before starting the transmission of the next octet. The handshaking is continued through out the message. At the end of message, the busy signal will be asserted until the completion of the message.

160.3 Operation

160.3.1 Introduction

The DMA interface provides a means for the user to receive blocks of data of arbitrary length (maximum 65535 bytes), normally these are packet structures such as CCSDS Space Packets. It also supports reception of one or more blocks of data into a fixed length field such as a CCSDS Telemetry Transfer Frame Data Field (framing mode).

160.3.2 Descriptor setup

The DMA interface is used for receiving data. The reception is done using descriptors located in memory. A single descriptor is shown in tables 2721 through 2722. The address field of the descriptor should point to the start of where the received data is to be stored. The address need not be word-aligned. If the interrupt enable (IE) bit is set, an interrupt will be generated when the transfer has completed (this requires that the interrupt enable bit in the control register is also set). The interrupt will be generated regardless of whether the transfer was successful or not. The wrap (WR) bit is also a control bit that should be set before reception and it will be explained later in this section.

*Table 2721.*GRPWRX descriptor word 0 (address offset 0x0)

31 16	15 9	8	7	6 4	3	2	1	0
LEN	RESERVED	CERR	OV	RESERVED	FHP	WR	IE	EN

31: 16	(LEN) - Length in bytes (note that length is limited to 2048 bytes for framing mode) In packet mode, the LEN field is written by the hardware after the reception. In framing mode, the LEN field is written by the software before reception.
15: 9	RESERVED
8:	Cyclic Redundancy Code Error (CERR) - (read only) Set to one when a CRC error was detected in a packet (speculative, only useful if CRC is present in received packet)
7:	Overrun (OV) - (read only) Overrun detected during transmission.
6: 3	RESERVED
3:	First Header Pointer (FHP) - First Header Pointer to be stored (2 bytes)
2:	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 16. The pointer automatically wraps to zero when the 16 kB boundary of the descriptor table is reached.
1:	Interrupt Enable (IE) - an interrupt will be generated when data for this descriptor has been received provided that the receive interrupt enable bit in the control register is set. The interrupt is generated regardless if the data was transferred successfully or if it terminated with an error.
0:	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.

Table 2722. GRPWRX descriptor word 1 (address offset 0x4)

31	0
ADDRESS	

31: 0 Address (ADDRESS) - Pointer to the buffer area to where data will be stored.

To enable a descriptor the enable (EN) bit should be set and after this is done, the descriptor should not be touched until the enable bit has been cleared by the core.

160.3.3 Packet mode

In packet mode, each descriptor corresponds to one received packet. The maximum length of a packet can be 65535 bytes. There is no check for too long packets. Reception of any too long packet will result in indeterministic behavior. The length of the received packet is automatically written into descriptor word 0.

160.3.4 Framing mode

In framing mode, each pair of descriptors correspond to one fixed length field as the CCSDS Telemetry Transfer Frame Data Field. The first descriptor defines the length (fixed for a field) and position in memory where the data is to be stored. The second descriptor in a pair defines the fixed length (2 bytes) and position of the memory where the First Header Pointer (FHP) calculated for the data received in a field belonging to the previous descriptor is to be stored. The First Header Pointer is calculated according to CCSDS: if the first packet starts at the beginning of the field then it is all zeros, if no packet starts in the field then it is all ones, any other location of the start of the first packet in a field is its count from the start of the field minus one. The First Header Pointer write-back is enabled by setting the FHP bit in the descriptor word 0. Normally the start location of First Header Pointer is two bytes in front of the field when CCSDS Telemetry Transfer Frames are used.

160.3.5 Starting transmission

Enabling a descriptor is not enough to start transmission. A pointer to the memory area holding the descriptors must first be set in the core. This is done in the descriptor pointer register. The address must be aligned to a 16 kByte boundary. Bits 31 to 14 hold the base address of descriptor area while bits 13 to 4 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the core, the pointer field is incremented by 16 to point at the next descriptor. The pointer will automatically wrap back to zero when the next 16 kByte boundary has been reached. The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 16 kByte boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when reception is active.

The final step to activate the reception is to set the enable bit in the DMA control register. This tells the core that there are more active descriptors in the descriptor table. This bit should always be set when new descriptors are enabled, even if transmission is already active. The descriptors must always be enabled before the transmission enable bit is set.

160.3.6 Descriptor handling after transmission

When the reception of a packet (or field in framing mode) has finished, status is written to the first word in the corresponding descriptor, while the second word is left untouched. The other bits in the first descriptor word are set to zero after reception. The enable bit should be used as the indicator when a descriptor can be used again, which is when it has been cleared by the core.

If the Cyclic Redundancy Code (CRC) bit is set, a CRC calculated over all but the two last octets, will be checked and the results stored in the descriptor. The CRC is defined in CCSDS 132.0-B-1. This is not supported in framing mode.

There are multiple bits in the DMA status register that hold status information.

The Receiver Interrupt (RI) bit is set each time a DMA reception ended successfully. The Receiver Error (RE) bit is set each time an DMA reception ended with an error. For either event, an interrupt is generated for transfers for which the Interrupt Enable (IE) was set in the descriptor. The interrupt is maskable with the Interrupt Enable (IE) bit in the control register.

The Receiver AMBA error (RA) bit is set when an AMBA AHB error was encountered either when reading a descriptor or when writing data. Any active reception was aborted and the DMA channel was disabled. It is recommended that the receiver is reset after an AMBA AHB error. The interrupt is maskable with the Interrupt Enable (IE) bit in the control register.

160.4 Registers

The core is programmed through registers mapped into APB address space.

Table 2723. GRPWRX registers

APB address offset	Register
0x00	GRPWRX DMA Control register
0x04	GRPWRX DMA Status register
0x08	GRPWRX DMA Descriptor Pointer register
0x80	GRPWRX Control register
0x84	GRPWRX Status register
0x88	GRPWRX Configuration register
0x8C	GRPWRX Physical Layer register

160.4.1 DMA Control Register

Table 2724. 0x00 - DCR - DMA control register

31 2	1	0
RESERVED	IE	EN
0	0	0
r	rw	rw

31: 2 RESERVED

1: Interrupt Enable (IE) - enable interrupts RA, RI, and RE

0: Enable (EN) - enable DMA transfers

160.4.2 DMA Status Register

Table 2725. 0x04 - DSR - DMA status register

31 4	3	2	1	0
RESERVED	ACTIVE	RA	RI	RE
0	NR	0	0	0
r	r	wc	wc	wc

31: 4 RESERVED

3: Active (ACTIVE) - DMA access ongoing

2: Receiver AMBA Error (RA) - DMA AMBA AHB error, cleared by writing a logical 1

1: Receiver Interrupt (RI) - DMA interrupt, cleared by writing a logical 1

0: Receiver Error (RE) - DMA receiver error, cleared by writing a logical 1

160.4.3 DMA Descriptor Pointer Register

Table 2726. 0x08 - DDP - DMA descriptor pointer register

31 14	13 4	3 0
BASE	INDEX	RESERVED
NR	NR	0
rw	rw	r

31: 14 Descriptor base (BASE) - base address of descriptor table

13: 4 Descriptor index (INDEX) - index of active descriptor in descriptor table

3: 0 Reserved - fixed to “0000”

160.4.4 Control Register

Table 2727. 0x80 - CTRL - control register

31 3	2	1	0
RESERVED	RST	RES	RxEN
0	0	1	0
r	r	r	r

31: 3 RESERVED

2: Reset (RST) - resets complete core

1: RESERVED

0: Receiver Enable (RxEN) - enables receiver (should be done after the complete configuration of the receiver)

160.4.5 Status Register

Table 2728. 0x84 - STAT - Status register

31 … 3	2	1	0
RESERVED	VALID	BUSY	READY
0	0	1	0
r	r	r	r

31: 3 RESERVED
2: Packet valid delimiter (VALID) - External valid signal
1: Busy with octet (BUSY) - External busy signal
0: Ready for packet (READY) - External ready signal

160.4.6 Configuration Register

Table 2729. 0x88 - CONF - configuration register

31 … 24	23 … 8	7 … 1	0
REVISION	FIFOSIZE	RESERVED	MODE
*	*	0	0
r	r	r	rw

31: 24 (REVISION) - Revision number (read-only)
23: 8 (FIFOSIZE) - FIFO size in bytes (read-only)
23: 1 RESERVED
0: (MODE) - Enable framing mode when set, else packet mode when cleared

160.4.7 Physical Layer Register

Table 2730. 0x8C - PLR - physical layer register

31 … 20	19 … 8	7	6	5	4	3 … 0
HALFBAUD	RESERVED	BUSY POS	READY POS	VALID POS	CLK RISE	RESERVED
0	0	0	1	1	1	0
r	r	rw	rw	rw	rw	r

31: 20 (HALFBAUD) - Received clock rate division factor with respect to the system clock - 1. Corresponds to the high phase of the incoming PacketWire bit clock. (read only)
19: 8 RESERVED
7: (BUSYPOS) - Positive polarity of busy input signal
6: (READYPOS) - Positive polarity of ready input signal
5: (VALIDPOS) - Positive polarity of valid output signal
4: (CLKRISE) - Rising clock edge in the middle of the serial data bit
3: 0 RESERVED

160.5 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x08E. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

160.6 Implementation

160.6.1 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual).

160.7 Configuration options

Table 2731 shows the configuration options of the core (VHDL generics).

*Table 2731.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR	0 - 16#FFF#	0
pmask	MASK field of the APB BAR	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by core	0 - NAHBIRQ-1	0
memtech	Memory technology	0 to NTECH	0
clktech	Clock buffer technology	0 to NTECH	0
buftype	Clock buffer type	TBD	0
burstlength	Sets the AHB burst length used by the core	-	16

160.8 Signal descriptions

Table 2732 shows the interface signals of the core (VHDL ports).

*Table 2732.*Signal descriptions

Signal name	Field	Type	Function		Active
RSTN	N/A	Input	Reset		Low
CLK	N/A	Input	Clock		-
APBI	*	Input	APB slave input signals		-
APBO	*	Output	APB slave output signals		-
AHBI	*	Input	AMB master input signals		-
AHBO	*	Output	AHB master output signals		-
PWO	BUSY_N	Output	Not ready for octet	Port indicates whether the receiver is ready to receive one octet. The port is considered asynchronous.	Programmable
	READY		Ready for packet	Port indicates whether the receiver is ready to receive one packet. The port is considered asynchronous.	Programmable
PWI	VALID	Input	Delimiter	Port is the packet delimiter for the output interface. It is deasserted between packets. The output is clocked out on the rising CLK edge.	Programmable
	CLK		Bit clock	Port is the PacketWire output bit clock. The output is clocked out on the rising CLK edge.	Programmable
	DATA		Data	Port is the serial data output for the interface. The output is clocked out on the rising CLK edge.	-
	ABORT		Abort	Port is clocked out on the rising CLK edge.	High

* see GRLIB IP Library User's Manual

160.9 Signal definitions and reset values

The signals and their reset values are described in table 2733.

*Table 2733.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
pwi_valid	Input	Delimiter	High	-
pwi_clock	Input	Bit clock	Rising	-
pwi_data	Input	Data	-	-
pwi_aboart	Input	Abort (fixed output)	High	-
pwo_busy_n	Output	Not ready for octet	Low	Logical 1
pwo_ready	Output	Ready for packet	High	Logical 0

160.10 Timing

The timing waveforms and timing parameters are shown in figure 395 and are defined in table 2734.

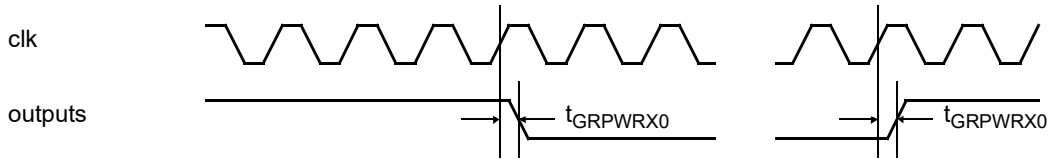


Figure 395. Timing waveforms

*Table 2734.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{GRPWRX0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns

Note: The inputs are re-synchronized inside the core. The signals do not have to meet any setup or hold requirements.

160.11 Library dependencies

Table 2735 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2735.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_TYPES	Signals, component	Component declaration

161 GRPWTX - PacketWire Transmitter

161.1 Overview

The PacketWire Transmitter implements a transmit function with Direct Memory Access (DMA) suppport. Packets (or blocks of data, normally CCSDS Space Packets) are automatically fetched from memory, for which the user configures a descriptor table with descriptors that point to each individual packet.

The core provides the following external and internal interfaces:

- Packet Wire interface (serial bit data, bit clock, packet delimiter, abort, ready, busy)
- AMBA AHB master interface, with sideband signals as per [GRLIB]
- AMBA APB slave interface, with sideband signals as per [GRLIB]

The operation of the transmitter is highly programmable by means of control registers.

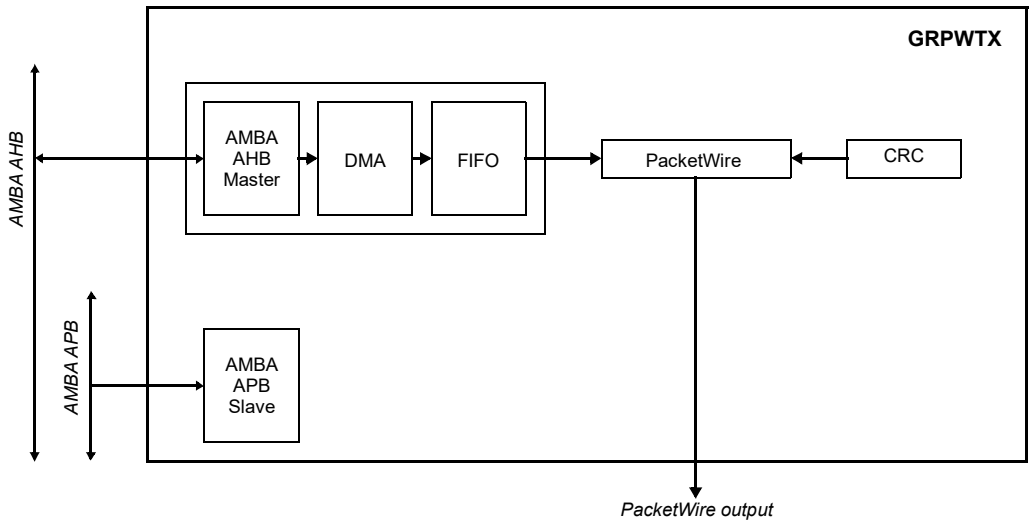


Figure 396. Block diagram

161.2 PacketWire interface

A PacketWire link comprises four ports for transmitting the message delimiter, the bit clock, the serial bit data and an abort signal. A link also comprises additional ports for busy signalling, indicating when the receiver is ready to receive the next octet, and for ready signalling, indicating that the receiver is ready to receive a complete packet. The waveform format shown in figure 397.

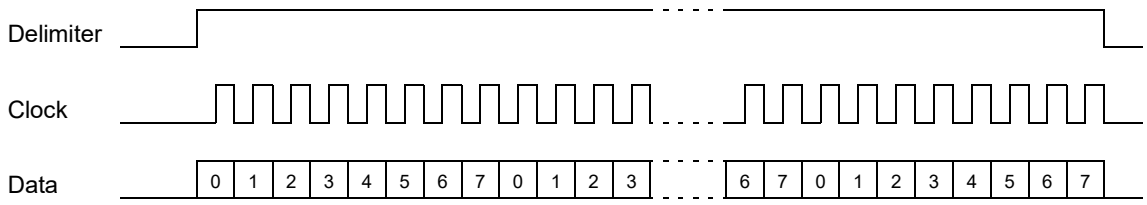


Figure 397. Synchronous bit serial waveform

The PacketWire protocol follows the CCSDS transmission convention, the most significant bit being sent first, both for octet transfers (control), and for word transfer (address or data). Transmitted data

should consist of multiples of eight bits otherwise the last bits will be lost. The message delimiter port is used to delimit messages (commands). It should be asserted while a message is being input, and deasserted in between. In addition, the message delimiter port should define the octet boundaries in the data stream, the first octet explicitly and the following octets each subsequent eight bit clock cycles.

The handshaking between the PacketWire link and the interface is implemented with a busy port. When a message is sent, the busy signal on the PacketWire link will be asserted as soon as the first data bit is detected, it will then be deasserted as soon as the interface is ready to receive the next octet. This gives the transmitter ample time to stop transmitting after the completion of the first octet and wait for the busy signal deassertion before starting the transmission of the next octet. The handshaking is continued through out the message. At the end of message, the busy signal will be asserted until the completion of the message.

161.3 Operation

161.3.1 Introduction

The DMA interface provides a means for the user to send blocks of data of arbitrary length, normally these are packet structures such as CCSDS Space Packets.

161.3.2 Descriptor setup

The DMA interface is used for sending data on the uplink. The transmission is done using descriptors located in memory. A single descriptor is shown in tables 2736 through 2737. The address field of the descriptor should point to the start of the data to be sent. The address need not be word-aligned. If the interrupt enable (IE) bit is set, an interrupt will be generated when the transfer has completed (this requires that the interrupt enable bit in the control register is also set). The interrupt will be generated regardless of whether the transfer was successful or not. The wrap (WR) bit is also a control bit that should be set before transmission and it will be explained later in this section.

*Table 2736.*GRPWTX descriptor word 0 (address offset 0x0)

31 16	15 8	7	6 4	3	2	1	0
LEN	RESERVED	UR	RESERVED	CRC	WR	IE	EN

31: 16	(LEN) - length in bytes
15: 8	RESERVED
7:	Underrun (UR) - Underrun detected during transmission.
6: 4	RESERVED
3:	Cyclic Redundancy Code (CRC) - Insert CRC, overwriting the two last octets of a data block
2:	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 16. The pointer automatically wraps to zero when the 16 kB boundary of the descriptor table is reached.
1:	Interrupt Enable (IE) - an interrupt will be generated when the data from this descriptor has been sent provided that the transmitter interrupt enable bit in the control register is set. The interrupt is generated regardless if the data was transferred successfully or if it terminated with an error.
0:	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.

*Table 2737.*GRPWTX descriptor word 1 (address offset 0x4)

31 0
ADDRESS

31: 0	Address (ADDRESS) - Pointer to the buffer area to where data will be fetched.

To enable a descriptor the enable (EN) bit should be set and after this is done, the descriptor should not be touched until the enable bit has been cleared by the core.

161.3.3 Starting transmission

Enabling a descriptor is not enough to start transmission. A pointer to the memory area holding the descriptors must first be set in the core. This is done in the descriptor pointer register. The address must be aligned to a 16 kByte boundary. Bits 31 to 14 hold the base address of descriptor area while bits 13 to 4 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the core, the pointer field is incremented by 16 to point at the next descriptor. The pointer will automatically wrap back to zero when the next 16 kByte boundary has been reached. The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 16 kByte boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when transmission is active.

If the Cyclic Redundancy Code (CRC) bit is set, a CRC calculated over all but the two last octets, will be inserted overwriting the two last octets of a data block. The CRC is defined in CCSDS 132.0-B-1.

The final step to activate the transmission is to set the enable bit in the DMA control register. This tells the core that there are more active descriptors in the descriptor table. This bit should always be set when new descriptors are enabled, even if transmission is already active. The descriptors must always be enabled before the transmission enable bit is set.

161.3.4 Descriptor handling after transmission

When the transmission has finished, status is written to the first word in the corresponding descriptor. The other bits in the first descriptor word are set to zero after transmission, while the second word is left untouched. The enable bit should be used as the indicator when a descriptor can be used again, which is when it has been cleared by the core.

There are multiple bits in the DMA status register that hold status information.

The Transmitter Interrupt (TI) bit is set each time a DMA transmission ended successfully. The Transmitter Error (TE) bit is set each time an DMA transmission ended with an error. For either event, an interrupt is generated for which the Interrupt Enable (IE) was set in the descriptor. The interrupt is maskable with the Interrupt Enable (IE) bit in the control register.

The Transmitter AMBA error (TA) bit is set when an AMBA AHB error was encountered either when reading a descriptor or data. Any active transmission was aborted and the DMA channel was disabled. It is recommended that the transmitter is reset after an AMBA AHB error. The interrupt is maskable with the Interrupt Enable (IE) bit in the control register.

161.4 Registers

The core is programmed through registers mapped into APB address space.

Table 2738. GRPWTX registers

APB address offset	Register
0x00	GRPWTX DMA Control register
0x04	GRPWTX DMA Status register
0x08	GRPWTX DMA Descriptor Pointer register
0x80	GRPWTX Control register
0x84	GRPWTX Status register
0x88	GRPWTX Configuration register
0x8C	GRPWTX Physical Layer register

161.4.1 DMA Control Register

Table 2739. 0x00 - DCR - DMA control register

31 2	1	0
RESERVED	IE	EN
0	0	0
r	rw	rw

31: 2 RESERVED

1: Interrupt Enable (IE) - enable interrupts TA, TI, and TE

0: Enable (EN) - enable DMA transfers

161.4.2 DMA Status Register

Table 2740. 0x04 - DSR - DMA status register

31 4	3	2	1	0
RESERVED	ACTIVE	TA	TI	TE

31: 4 RESERVED

3: Active (ACTIVE) - DMA access ongoing

2: Transmitter AMBA Error (TA) - DMA AMBA AHB error, cleared by writing a logical 1

1: Transmitter Interrupt (TI) - DMA interrupt, cleared by writing a logical 1

0: Transmitter Error (TE) - DMA transmitter error, cleared by writing a logical 1

161.4.3 DMA Descriptor Pointer Register

Table 2741. 0x08 - DDP - DMA descriptor pointer register

31 14	13 4	3 0
BASE	INDEX	RESERVED
NR	NR	0
rw	rw	r

31: 14 Descriptor base (BASE) - base address of descriptor table

13: 4 Descriptor index (INDEX) - index of active descriptor in descriptor table

3: 0 Reserved - fixed to “0000”

161.4.4 Control Register

Table 2742. 0x80 - CTRL - control register

31 3	2	1	0
RESERVED	RST	R	TxEN
0	0	0	0
r	rw	r	rw

31: 3 RESERVED

2: Reset (RST) - resets complete core

1: RESERVED

0: Transmitter Enable (TxEN) - enables transmitter (should be done after the complete configuration of the transmitter)

161.4.5 Status Register

Table 2743. 0x84 - STAT - Status register (read-only)

31 … 2	1	0
RESERVED	BUSY	READY
0	1	0
r	r	r

31: 2 RESERVED
1: Busy with octet (BUSY) - External busy signal
0: Ready for packet (READY) - External ready signal

161.4.6 Configuration Register

Table 2744. 0x88 - CONF - configuration register (read-only)

31 … 24	23 … 8	7 … 0
REVISION	FIFOSIZE	RESERVED
*	*	0
r	r	r

31: 24 (REVISION) - Revision number (read-only)
23: 8 (FIFOSIZE) - FIFO size in bytes (read-only)
7: 0 RESERVED

161.4.7 Physical Layer Register

Table 2745. 0x8C - PLR - physical layer register

31 … 20	19 … 8	7	6	5	4	3	2 … 0
HALFBAUD	RESERVED	BUSY POS	READY POS	VALID POS	CLK RISE	CLK MODE	RESERVED
1	0	0	1	1	1	0	0
rw	r	rw	rw	rw	rw	rw	r

31: 20 (HALFBAUD) - System clock division factor (indicates the width of the high and low phases of the outgoing PacketWire bit clock in number of system clock periods -1)
19: 8 RESERVED
7: (BUSYPOS) - Positive polarity of busy input signal
6: (READYPOS) - Positive polarity of ready input signal
5: (VALIDPOS) - Positive polarity of valid output signal
4: (CLKRISE) - Rising clock edge in the middle of the serial data bit
3: (CLKMODE) - 0=when valid (default), 1=always (experimental)
2: 0 RESERVED

161.5 Vendor and device identifier

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x08D. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

161.6 Implementation

161.6.1 Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual).

161.7 Configuration options

Table 2746 shows the configuration options of the core (VHDL generics).

*Table 2746.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR	0 - 16#FFF#	0
pmask	MASK field of the APB BAR	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by core	0 - NAHBIRQ-1	0
memtech	Memory technology	0 to NTECH	0
burstlength	Sets the AHB burst length used by the core	-	16

161.8 Signal descriptions

Table 2747 shows the interface signals of the core (VHDL ports).

*Table 2747.*Signal descriptions

Signal name	Field	Type	Function		Active
RSTN	N/A	Input	Reset		Low
CLK	N/A	Input	Clock		-
APBI	*	Input	APB slave input signals		-
APBO	*	Output	APB slave output signals		-
AHBI	*	Input	AMB master input signals		-
AHBO	*	Output	AHB master output signals		-
PWI	BUSY_N	Input	Not ready for octet	Port indicates whether the receiver is ready to receive one octet. The port is considered asynchronous.	Programmable
	READY		Ready for packet	Port indicates whether the receiver is ready to receive one packet. The port is considered asynchronous.	Programmable
PWO	VALID	Output	Delimiter	Port is the packet delimiter for the output interface. It is deasserted between packets. The output is clocked out on the rising CLK edge.	Programmable
	CLK		Bit clock	Port is the PacketWire output bit clock. The output is clocked out on the rising CLK edge.	Programmable
	DATA		Data	Port is the serial data output for the interface. The output is clocked out on the rising CLK edge.	-
	ABORT		Abort	Port is clocked out on the rising CLK edge.	High

* see GRLIB IP Library User's Manual

161.9 Signal definitions and reset values

The signals and their reset values are described in table 2748.

*Table 2748.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
pwo_valid	Output	Delimiter	High	Logical 0
pwo_clock	Output	Bit clock	Rising	Logical 0
pwo_data	Output	Data	-	Logical 0
pwo_abort	Output	Abort (fixed output)	High	Logical 0
pwi_busy_n	Input	Not ready for octet	Low	-
pwi_ready	Input	Ready for packet	High	-

161.10 Timing

The timing waveforms and timing parameters are shown in figure 398 and are defined in table 2749.

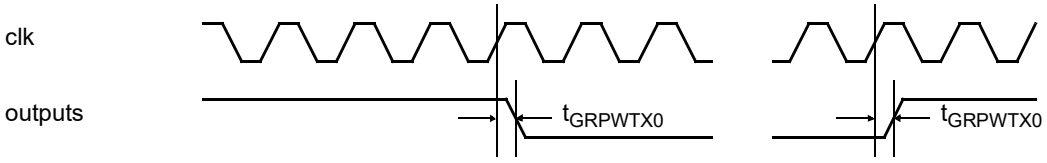


Figure 398. Timing waveforms

*Table 2749.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{GRPWTX0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns

Note: The inputs are re-synchronized inside the core. The signals do not have to meet any setup or hold requirements.

161.11 Library dependencies

Table 2750 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2750.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_TYPES	Signals, component	Component declaration

162 PW2APB - PacketWire receiver to AMBA APB Interface

162.1 Overview

The PacketWire to AMBA APB Interface implements the PacketWire protocol used by the Packet Telemetry Encoder (PTME) IP core and the Virtual Channel Assembler (VCA) device.

The core provides the following external and internal interfaces:

- Packet Wire interface (serial bit data, bit clock, packet delimiter, abort, ready, busy)
- AMBA APB slave interface, with sideband signals as per [GRLIB]

The core incorporates status and monitoring functions accessible via the AMBA APB slave interface. This includes:

- Valid and abort signalling from PacketWire interface
- Data overrun

Data is received on the PacketWire interface and read via the AMBA APB slave interface. It is possible to receive and read out one octet at a time.The packet delimiter and abort signals are observable via the control register, together with busy, ready and overrun signals. The baud rate is detected automatically and read via a configuration register.

162.2 PacketWire interface

A PacketWire link comprises four ports for transmitting the message delimiter, the bit clock, the serial bit data and an abort signal. A link also comprises additional ports for busy signalling, indicating when the receiver is ready to receive the next octet, and for ready signalling, indicating that the receiver is ready to receive a complete packet. The waveform format shown in figure 399.

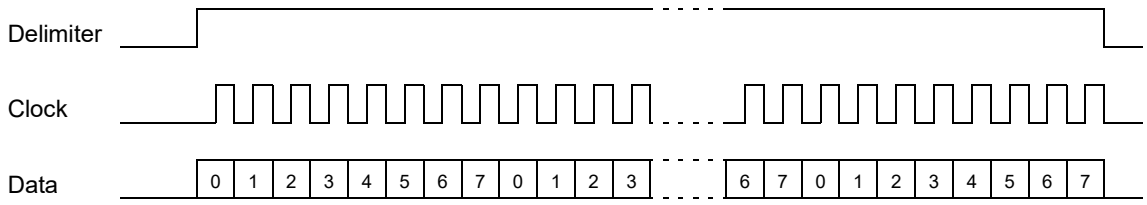


Figure 399. Synchronous bit serial waveform

The PacketWire protocol follows the CCSDS transmission convention, the most significant bit being sent first, both for octet transfers (control), and for word transfer (address or data). Transmitted data should consist of multiples of eight bits otherwise the last bits will be lost. The message delimiter port is used to delimit messages (commands). It should be asserted while a message is being input, and deasserted in between. In addition, the message delimiter port should define the octet boundaries in the data stream, the first octet explicitly and the following octets each subsequent eight bit clock cycles.

The maximum receiving input baud rate is defined as twice the frequency of the system clock input (f_{CLK}). The maximum receiving throughput is limited by the AMBA system into which this core is integrated. There is no lower limit for the input baud rate in the receiver.

The handshaking between the PacketWire link and the interface is implemented with a busy port. When a message is sent, the busy signal on the PacketWire link will be asserted as soon as the first data bit is detected, it will then be deasserted as soon as the interface is ready to receive the next octet. This gives the transmitter ample time to stop transmitting after the completion of the first octet and wait for the busy signal deassertion before starting the transmission of the next octet. The handshaking is continued through out the message.

To ensure that the receiver recognizes that the delimiter has been de-asserted, an application specific minimum time for the de-assertions should be imposed. Alternatively, ready signalling can be implemented in software, where the ready signal is asserted when the receiver software is ready to receive a packet, and is then immediately de-asserted when the first octet of a packet has been received. The transmitter software should thus only check the ready signal before starting to send a packet. This is illustrated in figure 400.

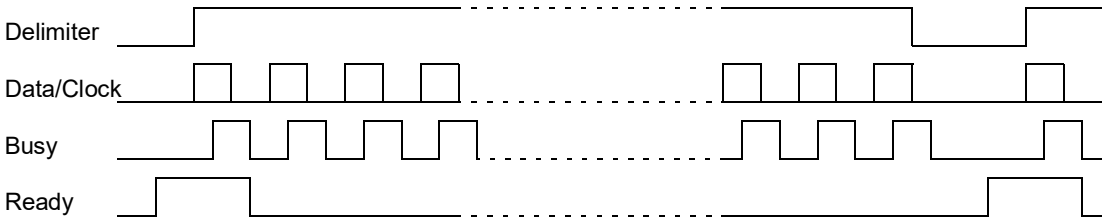


Figure 400. Handshake with ready signalling

To ensure that an abort is properly signalled, the abort signal should be de-asserted at the time the delimiter is asserted in the beginning of a packet, else the abort will be ignored. The abort signal should then be asserted while the delimiter is still asserted, to signal an abort to the receiver.

162.3 Registers

The core is programmed through registers mapped into APB address space.

Table 2751. PW2APB registers

APB address offset	Register
0x00	Control Register
0x04	Configuration Register
0x08	Data Reception Register

162.3.1 Control Register

*Table 2752.*0x00 - CTRL - Control Register

31 8	7	6	5	4	3	2	1 0
RESERVED	RESET	OV	READY	NEW	VALID	ABORT	SIZE
0	0	0	0	0	0	0	0
r	r	r	r	r	r	r	r

31: 8 RESERVED
- Write: Don't care.
- Read: All zero.

7 RESET
- Write: Write logical one to reset core
- Read: All zero

6 OV
- Write: Don't care.
- Read: Data input overrun, cleared on read.

(Note that a received octet that results in an overrun does not overwrite the OCTET field.)

(If there is a new octet available, as indicated by the NEW bit, it should be read out to free the receiver data reception register.)

5 READY
- Read/Write: Interface ready to receive a packet (only affects output signal)

4 NEW
- Write: Don't care.
- Read: Interface has received a new octet, ready for read-out (cleared when receiver data reception register has been read)

3 VALID
- Write: Don't care.
- Read: Packet delimiter when asserted (reflects input signal value)

2 ABORT
- Write: Don't care.
- Read: Abort current packet when asserted (only set when a valid abort takes place)

(If there is a new octet available, as indicated by the NEW bit, it should be read to free the receiver data reception register.)

1: 0 SIZE — Reception size and order (left to right):
- Read/: `00 = 8 bit: 7:0`

162.3.2 Configuration Register

*Table 2753.*0x04 - CONF -Configuration Register

31 8	7 0
RESERVED	HALFBAUD
0	0
r	r

31: 8 RESERVED
- Write: Don't care.
- Read: All zero.

7: 0 HALFBAUD — System clock division factor (indicates the width of the high phase of the incoming PacketWire bit clock in number of system clock periods -1)
- Read: `0x00 = divide by 1`

`0xFF = divide by 256`

162.3.3 Data Reception Register

*Table 2754.*0x08 - DRR - Data Reception Register

31 8	7 0
RESERVED	OCTET
0	0
r	rw

31: 8 RESERVED

Write: Don't care.

Read: All zero.

7: 0 OCTET

Write: Last octet correctly received. Note that a received octet that results in an overrun does not overwrite the OCTET field.

Read: All zero.

162.4 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x03C. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

162.5 Configuration options

Table 2755 shows the configuration options of the core (VHDL generics).

*Table 2755.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFC#
syncrst	Only synchronous reset	0, 1	1

162.6 Signal descriptions

Table 2756 shows the interface signals of the core (VHDL ports).

Table 2756. Signal descriptions

Signal name	Field	Type	Function		Active
RSTN	N/A	Input	Reset		Low
CLK	N/A	Input	Clock		-
APBI	*	Input	APB slave input signals		-
APBO	*	Output	APB slave output signals		-
PWO	BUSY_N	Input	Not ready for octet	This input port indicates whether the receiver is ready to receive one octet. The input is considered as asynchronous.	Low
	READY		Ready for packet	This input port indicates whether the receiver is ready to receive one packet. The input is considered as asynchronous.	High
PWI	VALID	Output	Delimiter	This output port is the packet delimiter for the output interface. It is deasserted between packets. The output is clocked out on the rising CLK edge.	High
	CLK		Bit clock	This output port is the PacketWire output bit clock. The output is clocked out on the rising CLK edge.	Rising
	DATA		Data	This output port is the serial data output for the interface. The output is clocked out on the rising CLK edge.	-
	ABORT		Abort	The output is clocked out on the rising CLK edge.	High

* see GRLIB IP Library User's Manual

162.7 Signal definitions and reset values

The signals and their reset values are described in table 2757.

Table 2757. Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
pwo_valid	Output	Delimiter	High	Logical 0
pwo_clock	Output	Bit clock	Rising	Logical 0
pwo_data	Output	Data	-	Logical 0
pwo_aboart	Output	Abort	High	Logical 0
pwi_busy_n	Input	Not ready for octet	Low	-
pwi_ready	Input	Ready for packet	High	-

162.8 Timing

The timing waveforms and timing parameters are shown in figure 401 and are defined in table 2758.

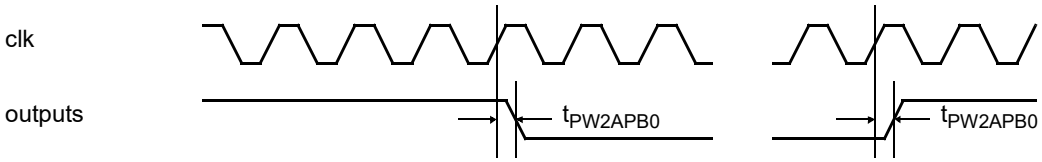


Figure 401. Timing waveforms

*Table 2758.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{PW2APB0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns

Note: The inputs are re-synchronized inside the core. The signals do not have to meet any setup or hold requirements.

162.9 Library dependencies

Table 2759 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2759.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_Types	Signals, component	Component declarations, signals.

163 APB2PW - AMBA APB to PacketWire Transmitter Interface

163.1 Overview

The AMBA APB to PacketWire Interface implements the PacketWire protocol used by the Packet Telemetry Encoder (PTME) IP core and the Virtual Channel Assembler (VCA) device.

The core provides the following external and internal interfaces:

- Packet Wire interface (serial bit data, bit clock, packet delimiter, abort, ready, busy)
- AMBA APB slave interface, with sideband signals as per [GRLIB]

The core incorporates status and monitoring functions accessible via the AMBA APB slave interface. This includes:

- Busy and ready signalling from PacketWire interface

Data are transferred to the PacketWire interface by writing to the AMBA APB slave interface. It is possible to transfer one, two or four bytes at a time, following the AMBA big-endian convention regarding send order. Data are output serially on the PacketWire interface.The packet delimiter and abort signals are controlled, together with the data size, through the control register. The progress of the interface can be monitored via the AMBA APB slave interface, through the control register. The baud rate is set via a configuration register.

163.2 PacketWire interface

A PacketWire link comprises four ports for transmitting the message delimiter, the bit clock, the serial bit data and an abort signal. A link also comprises additional ports for busy signalling, indicating when the receiver is ready to receive the next octet, and for ready signalling, indicating that the receiver is ready to receive a complete packet. The waveform format shown in figure 402.

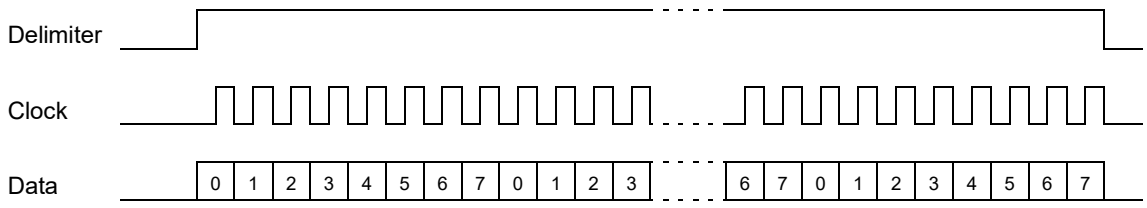


Figure 402. Synchronous bit serial waveform

The PacketWire protocol follows the CCSDS transmission convention, the most significant bit being sent first, both for octet transfers (control), and for word transfer (address or data). Transmitted data should consist of multiples of eight bits otherwise the last bits will be lost. The message delimiter port is used to delimit messages (commands). It should be asserted while a message is being input, and deasserted in between. In addition, the message delimiter port should define the octet boundaries in the data stream, the first octet explicitly and the following octets each subsequent eight bit clock cycles.

The handshaking between the PacketWire link and the interface is implemented with a busy port. When a message is sent, the busy signal on the PacketWire link will be asserted as soon as the first data bit is detected, it will then be deasserted as soon as the interface is ready to receive the next octet. This gives the transmitter ample time to stop transmitting after the completion of the first octet and wait for the busy signal deassertion before starting the transmission of the next octet. The handshaking is continued through out the message.

To ensure that the receiver recognizes that the delimiter has been de-asserted, an application specific minimum time for the de-assertions should be imposed. Alternatively, ready signalling can be implemented in software, where the ready signal is asserted when the receiver software is ready to receive a

packet, and is then immediately de-asserted when the first octet of a packet has been received. The transmitter software should thus only check the ready signal before starting to send a packet. This is illustrated in figure 403.

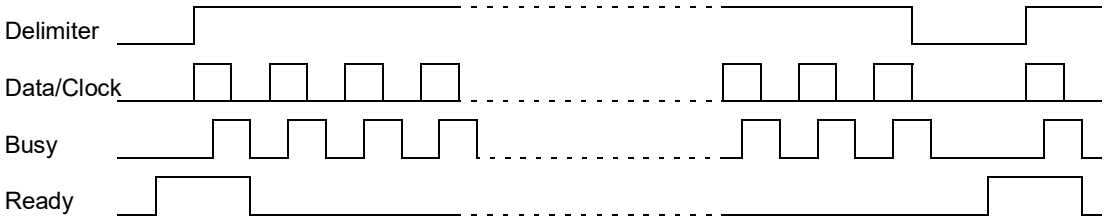


Figure 403. Handshake with ready signalling

To ensure that an abort is properly signalled, the abort signal should be de-asserted at the time the delimiter is asserted in the beginning of a packet, else the abort will be ignored. The abort signal should then be asserted while the delimiter is still asserted, to signal an abort to the receiver.

163.3 Registers

The core is programmed through registers mapped into APB address space.

Table 2760. APB2PW registers

APB address offset	Register
0x000	Control Register
0x004	Configuration Register
0x008	Data Transmission Register

163.3.1 Control Register

*Table 2761.*0x000 - CTRL - Control Register

31 8	7	6	5	4	3	2	1 0
RESERVED	RESET	R	READY	BUSY	VALID	ABORT	SIZE

31: 8 RESERVED
- Write: Don't care.
- Read: All zero.

7 RESET
- Write: Write logical one to reset core
- Read: All zero

6 RESERVED
- Write: Don't care.
- Read: All zero.

5 READY
- Write: Don't care.
- Read: Interface ready to receive a packet

4 BUSY
- Write: Don't care.
- Read: Interface busy with octet(s) when set, else ready for data input

3 VALID
- Read/Write: Packet delimiter when asserted (only affects output signal)

2 ABORT
- Read/Write: Abort current packet when asserted (only affects output signal)

1: 0 SIZE — Transfer size and order (left to right):
- Read/write: 00 = 8 bit: 7:0
- 01 = 16 bit: 15:8, 7:0
- 10 = 24 bit: 23:16, 15:8, 7:0
- 11 = 32 bit: 31:24, 23:16, 15:8, 7:0

Power-up default: 0x00000000

163.3.2 Configuration Register

*Table 2762.*0x004 - CONF - Configuration Register

31 8	7 0
RESERVED	HALFBAUD

31: 8 RESERVED
- Write: Don't care.
- Read: All zero.

7: 0 HALFBAUD — System clock division factor (indicates the width of the high and low phases of the outgoing PacketWire bit clock in number of system clock periods -1)
- Read/write: 0x00 = divide by 1
- 0xFF = divide by 256

Power-up default: 0x00000000

163.3.3 Data Transmission Register

*Table 2763.*0x008 - TX - Data Transmission Register

31 24	23 16	15 8	7 0
FIRST OCTET	SECOND OCTET	THIRD OCTET	LAST OCTET

31: 24 FIRST OCTET
- Write: First octet to be transmitted, if SIZE=11
- Read: All zero.

23: 16 SECOND OCTET
- Write: Second octet to be transmitted
- Read: All zero.

15: 8 THIRD OCTET
- Write: First octet to be transmitted
- Read: All zero.

7: 0 LAST OCTET
- Write: Last octet to be transmitted, any SIZE value
- Read: All zero.

Power-up default: 0x00000000

163.4 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x03B. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

163.5 Configuration options

Table 2764 shows the configuration options of the core (VHDL generics).

*Table 2764.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFC#
syncrst	Only synchronous reset	0, 1	1

163.6 Signal descriptions

Table 2765 shows the interface signals of the core (VHDL ports).

Table 2765. Signal descriptions

Signal name	Field	Type	Function		Active
RSTN	N/A	Input	Reset		Low
CLK	N/A	Input	Clock		-
APBI	*	Input	APB slave input signals		-
APBO	*	Output	APB slave output signals		-
PWI	BUSY_N	Input	Not ready for octet	This input port indicates whether the receiver is ready to receive one octet. The input is considered as asynchronous.	Low
	READY		Ready for packet	This input port indicates whether the receiver is ready to receive one packet. The input is considered as asynchronous.	High
PWO	VALID	Output	Delimiter	This output port is the packet delimiter for the output interface. It is deasserted between packets. The output is clocked out on the rising CLK edge.	High
	CLK		Bit clock	This output port is the PacketWire output bit clock. The output is clocked out on the rising CLK edge.	Rising
	DATA		Data	This output port is the serial data output for the interface. The output is clocked out on the rising CLK edge.	-
	ABORT		Abort	The output is clocked out on the rising CLK edge.	High

* see GRLIB IP Library User's Manual

163.7 Signal definitions and reset values

The signals and their reset values are described in table 2766.

Table 2766. Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
pwi_valid	Input	Delimiter	High	-
pwi_clock	Input	Bit clock	Rising	-
pwi_data	Input	Data	-	-
pwi_aboart	Input	Abort	High	-
pwo_busy_n	Output	Not ready for octet	Low	Logical 0
pwo_ready	Output	Ready for packet	High	Logical 0

163.8 Timing

The timing waveforms and timing parameters are shown in figure 404 and are defined in table 2767.

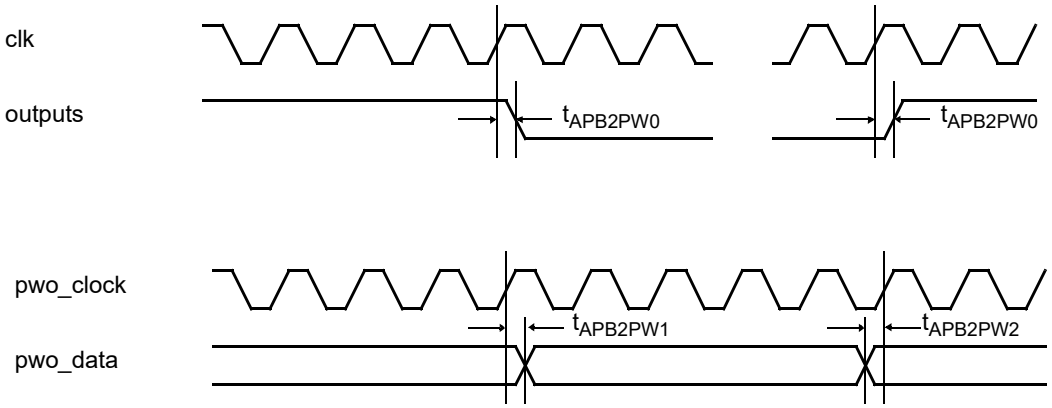


Figure 404. Timing waveforms

*Table 2767.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{APB2PW0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns
$t_{APB2PW1}$	input to clock hold	rising *pwo_clock* edge	TBD	-	ns
$t_{APB2PW2}$	input to clock setup	rising *pwo_clock* edge	TBD	-	ns
$t_{APB2PW3}$	pwo_valid to pwo_clock edge	rising *pwo_clock* edge	TBD	-	ns
$t_{APB2PW4}$	pwo_valid de-asserted period	-	TBD*t_{CLK}		periods

Note: The inputs are re-synchronized inside the core. The signals do not have to meet any setup or hold requirements.

163.9 Library dependencies

Table 2768 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2768.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_Types	Signals, component	Component declarations, signals.

164 AHB2PP - AMBA AHB to Packet Parallel Interface

164.1 Overview

The AMBA AHB to Packet Parallel Interface implements the PacketParallel protocol used by the Packet Telemetry Encoder (PTME) IP core and the Virtual Channel Assembler (VCA) device.

The core implements the following functions:

- Packet Parallel protocol
- General Purpose Input Output port

The core provides the following external and internal interfaces:

- Packet Parallel interface (octet data, packet delimiter, write strobe, abort, ready, busy)
- AMBA AHB slave interface, with sideband signals as per [GRLIB]
- AMBA APB slave interface, with sideband signals as per [GRLIB]

The core incorporates status and monitoring functions accessible via the AMBA APB slave interface. This includes:

- Busy and ready signalling from Packet Parallel interface
- Read back of output data
- Interrupts on ready for new word, or ready for new packet

Data is transferred to the Packet Parallel interface by writing to the AMBA AHB slave interface, located in the AHB I/O area. Writing is only possible when the Packet Parallel packet valid delimiter is asserted, else the access results in an AMBA access error. It is possible to transfer one, two or four bytes at a time, following the AMBA big-endian convention regarding send order. The last written data can be read back via the AMBA AHB slave interface. Data are output as octets on the Packet Parallel interface.

In the case the data from a previous write access has not been fully transferred over the Packet Parallel interface, a new write access will result in an AMBA retry response. The progress of the interface can be monitored via the AMBA APB slave interface. An interrupt is generated when the data from the last write access has been transferred. An interrupt is also generated when the Packet Parallel ready indicator is asserted.

164.2 Interrupts

Two interrupts are implemented by the Packet Parallel interface:

Index:Name:Description:

0 NOT BUSYReady for a new data (word, half-word or byte)

1 READYReady for new packet

The interrupts are configured by means of the *pirq* VHDL generic.

164.3 Registers

The core is programmed through registers mapped into APB address space.

Table 2769. AHB2PP registers

APB address offset	Register
0x000	Configuration Register
0x004	Status Register
0x008	Control Register
0x010	Data Input Register
0x014	Data Output Register
0x018	Data Direction Register

164.3.1 Configuration Register (R/W)

Table 2770. 0x000 - CONF - Configuration Register

31 … 3	2 … 0
RESERVED	WS
0	0
r	rw

3-0: WS Number of additional Wait States

All bits are cleared to 0 at reset.

The width of the write strobe can be extended by mean of the WS field. The nominal asserted width is one system clock period (corresponding to WS=0). The asserted period can be extended up to a total asserted width of 16 system clock periods.

The minimum gap between octet write accesses when the strobe is de-asserted is one system clock period when WS={0, 3}, two when WS={4, 7}, three when WS={8, 11}, and four when WS={12, 15}.

164.3.2 Status Register)

Table 2771. 0x004 - STAT - Status register

31 … 3	2	1	0
RESERVED	BUSY	PP Busy	PP Ready
0	0	0	0
r	r	r	r

2: BUSY AHB2PP interface busy with data transfer
1: PP Busy Packet Parallel busy input
0: PP Ready Packet Parallel ready input

All bits are cleared to 0 at reset.

164.3.3 Control

*Table 2772.*0x008 - CTRL - Control Register

31	4	3	2	1	0
RESERVED		PP Abort	PP Valid	RST	EN
0		0	0	0	0
r		rw	rw	rw*	rw

3: PP Abort Packet Parallel abort output
2: PP Valid Packet Parallel valid output
1: RESET Reset complete core when 1
0: ENABLE Enable Packet Parallel interface when 1, else enable GPIO function

All bits are cleared to 0 at reset. Note that RESET is read back as 0b.

164.3.4 Data Input

*Table 2773.*0x010 - DIN - Data Input Register

31	8	7	0
RESERVED		DIN	
0		0	
r		r	

7-0: DIN Input data *ppi.data[7:0]*

All bits are cleared to 0 at reset.

164.3.5 Data Output

*Table 2774.*0x014 - DOUT - Data Output Register

31	8	7	0
RESERVED		DOUT	
0		0	
r		rw	

7-0: DOUT Output data *ppo.data[7:0]*

All bits are cleared to 0 at reset.

Note that the GPIO functionality can only be used when the Packet Parallel interface is disabled via the Control Register above.

164.3.6 Data Direction

*Table 2775.*0x018 - DIR - Data Direction Register

31	8	7	0
RESERVED		DDIR	
0		0	
r		r	

7-0: DDIR Direction: *ppo.enable[7:0]*
0b = input = high impedance,

1b = output = driven

All bits are cleared to 0 at reset.

Note that the GPIO functionality can only be used when the Packet Parallel interface is disabled via the Control Register above.

164.4 AHB I/O area

Data to be transferred to the Packet Parallel interface is written to the AMBA AHB slave interface which implements a AHB I/O area. See [GRLIB] for details.

Note that the address is not decoded by the core. Address decoding is only done by the AMBA AHB controller, for which the I/O area location and size is configured by means of the *ioaddr* and *iomask* VHDL generics.

It is possible to transfer one, two or four bytes at a time, following the AMBA big-endian convention regarding send order. The last written data can be read back via the AMBA AHB slave interface. Data are output as octets on the Packet Parallel interface.

Table 2776. AHB I/O area - data word definition

31 24	23 16	15 8	7 0
DATA [31:24]	DATA [23:16]	DATA [15:8]	DATA [7:0]

Table 2777. AHB I/O area - send order

Transfer size	Address offset	DATA [31:24]	DATA [23:16]	DATA [15:8]	DATA [7:0]	Comment
Word	0	first	second	third	last	Four bytes sent
Halfword	0	first	last	-	-	Two bytes sent
	2	-	-	first	last	Two bytes sent
Byte	0	first	-	-	-	One byte sent
	1	-	first	-	-	One byte sent
	2	-	-	first	-	One byte sent
	3	-	-	-	first	One byte sent

164.5 Vendor and device identifiers

The module has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x039. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

164.6 Configuration options

Table 2778 shows the configuration options of the core (VHDL generics).

*Table 2778.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index.	1 - NAHBSLV-1	0
ioaddr	Addr field of the AHB IO bar.	0 - 16#FFF#	0
iomask	Mask field of the AHB IO bar.	0 - 16#FFF#	16#F00#
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFC#
pirq	Interrupt line used by the AHB2PP.	0 - NAHBIRQ-1	0
syncrst	Only synchronous reset	0, 1	1
oepol	Output enable polarity	0, 1	1

164.7 Signal descriptions

Table 2779 shows the interface signals of the core (VHDL ports).

*Table 2779.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB slave input signals	-
AHBO	*	Output	AHB slave output signals	-
PPI	busy_n	Input	Packet Parallel busy signal	-
	ready		Packet Parallel ready signal	
	data[7:0]		Packet Parallel data (GPIO only)	
PPO	abort	Output	Packet Parallel abort signal	-
	valid_n		Packet Parallel packet delimiter signal	-
	wr_n		Packet Parallel octet write strobe	
	data[7:0]		Packet Parallel octet data	
	enable[7:0]		Enable/drive octet data output	

* see GRLIB IP Library User's Manual

164.8 Library dependencies

Table 2780 shows the libraries used when instantiating the core (VHDL libraries).

*Table 2780.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
TMTC	TMTC_Types	Signals, component	Component declarations, signals.

165 GRRM - Reconfiguration Module

165.1 Overview

The core processes different alarms and provides reconfiguration commands as outputs. When alarm conditions are matched the reconfiguration sequences are fetched from the external memory, these sequences are processed and output commands are provided specific to an alarm. The core operates in an AMBA bus system where both the AMBA AHB bus and the APB bus are present. The AMBA APB bus is used for configuration, control and status handling and the AMBA AHB Master is used to generate read and write access on the AMBA AHB bus.

165.2 Operation

According to the alarm activated and the current state of the core an alarm pattern is formed, the alarm pattern provides the address from which the reconfiguration sequences should be fetched. The AMBA AHB bus is used for retrieving the reconfiguration sequences in memory external to the core. The reconfiguration sequences consist of internal commands and external commands. The internal commands are processed by internal command processor which performs actions like masking the alarm that got activated and changing the state according to the internal commands. The external commands are formed into packets and sent to the command pulse distribution unit using PacketWire interface. The alarms can be logged to external memory along with the time instance at which the alarm is triggered. During initialization the core can configure itself or other systems available in the AHB bus.

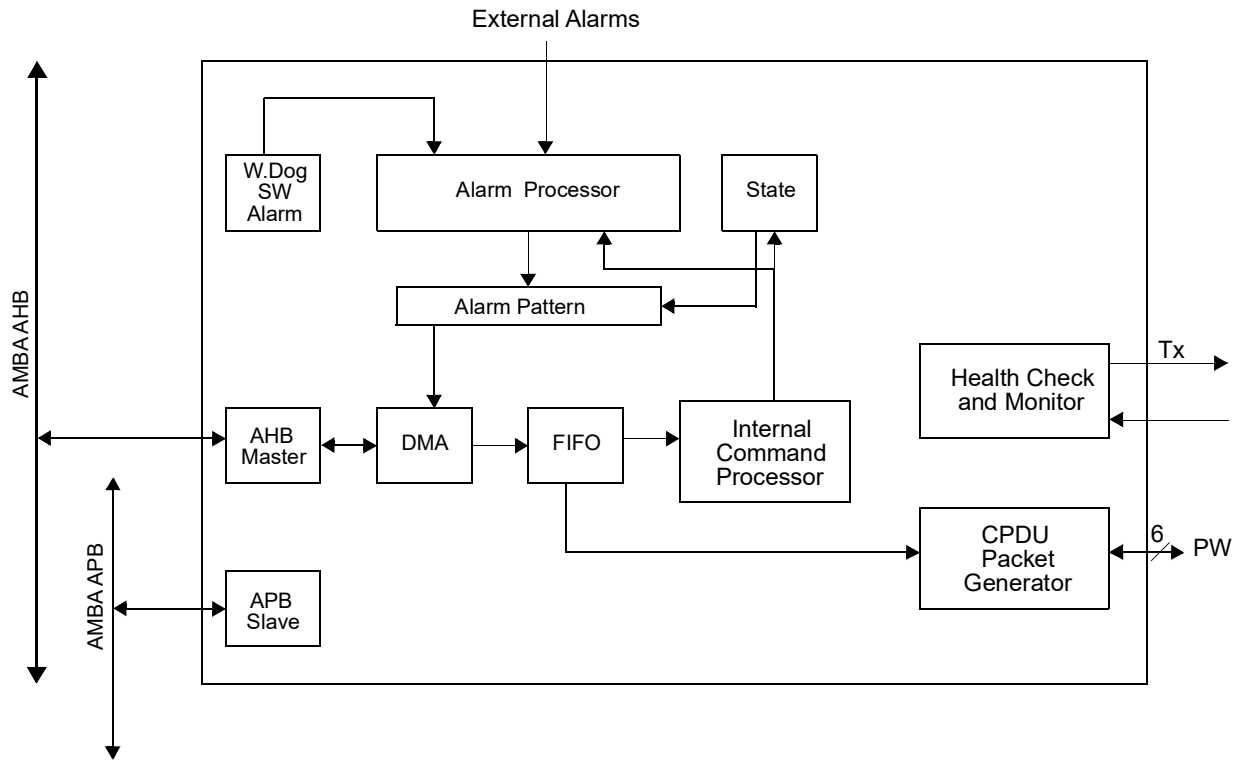


Figure 405. Block diagram

165.3 Alarms

External alarms (external discrete signal inputs), watchdog timeout and software alarms (alarm triggered by processors or any other unit capable of writing the APB register space) are the alarm inputs. Using watchdog and software alarms the core can monitor the processor module and using external

alarms the core can monitor high-level system alarms for example low battery indicator. All alarms are individually maskable using the alarm mask AM register.

165.3.1 External alarms

The external alarm inputs are processed and an alarm event is generated when the input matches the conditions specified in the alarm monitoring unit. For external alarms the monitoring unit consist of mask AM, polarity AP and delay AD registers, these registers must be configured according to the requirement for a particular external alarm. An external input should pass the above mentioned condition and produces the alarm event which is registered as a pending event in the alarm pending AP register.

165.3.2 Watchdog timeout

Watchdog timeout: it is a down-counter that must be cyclically reloaded upon by the processor. If it is not reloaded an alarm event is triggered which is registered as a pending event in the alarm pending AP register. The corresponding mask AM register must be enabled.

165.3.3 Software alarms

The processor can trigger a software alarm event by configuring the register SWNP. The SWNP.SWPM0 register bits can be configured with a value corresponds to a particular software alarm. The value 0b000 is reserved and all other values trigger an alarm respectively (seven software alarm). The SWNP.EN bit must also be set to trigger the alarm.

The software alarms can also be used to check the functionality of the core by allocating one of the alarms to be self check alarm, the processor can trigger that alarm and expect the reconfiguration sequences to clear that alarm pending event. The reconfiguration sequence must also be configured according to the need.

The software alarm 0b001 has the highest priority and 0b111 has the lowest priority.

165.4 Alarm Pattern

Alarm pending AP register memorizes any alarm that has triggered an alarm event. Each alarm becomes a pending event. The reconfiguration sequences are selected according to the highest priority pending alarm event and the current state of the state machine. The highest priority is given to external alarms, followed by watchdog alarms and finally software alarms.

Alarm pattern is formed by concatenation of the priority vector (LSB) and the state machine output (MSB). This vector is used as a pointer to the first address of the reconfiguration sequence to be executed. On this basis, the reconfiguration sequences are read from the external memory and stored in the internal buffers for processing.

The core includes a state machine STAT.STATE register of which the state can be modified by the reconfiguration sequences. The updates to the state machine by reconfiguration sequence allows another reconfiguration sequence to be executed for the same alarm if a previous reconfiguration attempt did not remedy the alarm situation.

165.5 Reconfiguration Log

After occurrence of every alarm event, the alarm pattern is stored in a circular buffer (available in the external memory area) allocated for reconfiguration log. Only previous 16 patterns (alarm events) can be stored. The alarm log address ALA register specify the location where the log must be stored. Each

log consist of 2 words. The log consist of time instance at which the alarm event occurred and the alarm pattern (state+alarm). The content of the log is shown in the table below.

Table 2781. Alarm Log Content

Word 1	Coarse Time (32 bits)	
Word 2	Fine Time (23 bits)	ALARM PATTERN (9 bits)

165.6 Reconfiguration Sequences

The reconfiguration sequences are fetched and stored into the internal buffers for an alarm event. The commands are processed one by one. Each reconfiguration sequence is made up of 64 Commands sent to Command Pulse Distribution Unit (CPDU) separated by delay and 64 commands for RM internal configuration (split into two - internal1 and internal2 of each containing 32 commands). The order of execution is internal1 commands, external commands followed by internal2 commands. The starting address of the initial (highest priority alarm) reconfiguration sequence location is provided by the RSA.SA register and for the other alarms the address from which the sequences are fetched is computed by the core itself using the particular alarm event and state of the alarm. A spreadsheet is provided along with this document to facilitate the configuration of these reconfiguration sequences, the output file from the spreadsheet can be used to load the reconfiguration sequences into an external memory.

165.6.1 Internal commands

The internal 64 commands are split into 2 blocks each containing 32 commands. The internal commands will perform internal control of the RM like masking the alarm currently in process, changing the state machine value (to try different pattern or reconfigure). The table below list the commands.

Table 2782. Internal Commands

Command	Parameter (maximum 24 bits)	Code	Purpose
Clear all pending	No	0x00	Clear all pending alarms
Clear current pending	No	0x01	Clear current pending alarm
Mask all	No	0x02	Mask all alarms
Mask current	No	0x03	Mask current pending alarms
Unmask all	No	0x04	Unmask all alarms
Unmask current	No	0x05	Unmask current alarm
Rearm Watchdog Nominal	Yes (24 bit value)	0x06	Rearm watchdog with the value in the parameter
Rearm Watchdog Redundant	Yes (24 bit value)	0x07	Rearm watchdog with the value in the parameter
Delay	Yes (16 bit value)	0x08	Perform delay of processing according to the parameter value. The prescaler for this delay is taken from the CP register
Set state	Yes (4 bit value)	0x09	Set state according to parameter
Set active SGM	Yes (3 bit value)	0x0A	Set which safe guard memory (SGM) contains the active content. According to this the GSC.CONT registers are updated
Check alarm	Yes (4 bit value)	0x0B	Check that the alarm is not triggered again after a reconfiguration, if triggered again increment the state and perform another reconfiguration sequence. This must be used in combination with other internal commands (delay, pending mask). For example: clear the corresponding pending alarm, unmask the processed alarm, provide a delay (if needed to check that the alarm happens again) and finally provide the Check alarm command, the state will be incremented only when the alarm is still pending.
Processor Write protect	Yes (4 bit value)	0x0C	According to this value the PSC.PMWP registers are updated. The registers are driven as output signals and connected to the write protection unit
End internal command	No	0xFE	After this the internal command execution moves to the next stage (to external command if internal command 1 was processed or end of command processing if internal 2 was processed)
Do nothing	No	0xFF	Skip a command execution

165.6.2 External commands

The External commands are sent as CPDU packets using PacketWire interface.

The format of each external command is shown below and the format of the final packet generated is explained in the next section. If all the external commands are not needed for processing the external command execution can be stopped by setting the delay section (see below table) of the reconfiguration sequence to be 0xFFFE.

Table 2783. External command format

31 16	15 4	3	2 0
Delay	PULSE	-	LEN
-			

31: 16	Delay - Delay needed in between different external commands (resolution of 1 ms and configurable using the CP register)
15: 4	PULSE - Pulse output number
3	Reserved
2: 0	LEN - Pulse Length

165.7 CPDU Packet Generator

The external commands (PULSE and LEN in the above table) in the reconfiguration sequences are formed into CPDU Packets that contain one Command Instruction. These are sent as output on a PacketWire interface as TeleCommand segments.

The block that generates the CPDU packets uses the flowing input data: 11-bit Application Process ID (programmable through CC.APID register), 12-bit pulse output number and 3-bit pulse length (PULSE and LEN in the above table). The Segment is illustrated in the table below.

*Table 2784.*CPDU Segment.

Telecommand Segment													
Segment Header		Packet Header							Packet Data Field				
Seq. Flags	MAP Id.	Packet ID				Packet Seq Control		Packet Length	Command Instruction				Packet Error Control
		Version Number	Type	Data Filed Header Flag	Applica-tion Pro-cess ID	Seq. Flags	Seq. Count		Output No. LSB	Output No. MSB	Reserv-ed	Pulse Length	
3	0	0	1	0	Configu-rable	3	Auto Incr.	3	Configu-rable	Configu-rable	0	Configu-rable	Enabled
2	6	3	1	1	11	2	14	16	8	4	1	3	
8		16				16		16	16				16

The sequence counter is automatically incremented after each packet

165.8 Initial Core Configuration

During initialization the core can configure itself or other systems available in the AHB bus. The Initial configuration address ICA.CA register specify the initial configuration base address. Each configuration consist of four word of information described in the table below.

*Table 2785.*Initial Configuration

Commands	**Purpose**
Address Source	Source from where the data must be fetched
Address Destination	Destination to where the data must be stored
Length in words (4 bytes)	number of words to be transferred. Always one more than the number specified here. (Zero means 1 and 31 means 32, maximum 512 bytes i.e. 128 words)
Pointer	Next command base address. Least significant bit specify next command is available or not. If more configurations need to be performed set the least significant bit must be set to'1' other wise '0'

The configuration must be performed before the initial delay before start of alarm monitoring ID.DELAY register pass out. This delay is provided for the external units and processor to set the alarm inputs at their right nominal state and thus ensure that untimely reconfiguration are not executed because the system is still booting.

165.9 Health Communication

The RM can send its role and health informations to another RM if an health link is available in between them. By default one of the RM is set to be the master and the other Slave. The CTRL.ACTMST register specify whether the RM is active or inactive.

The active master will set itself to be in execute state (after the initial delay timeout) and the inactive slave set itself to be in the monitor state. The health state is available in the health status HSTAT.HSTATE register. The master RM can send active message to the slave RM using the communication link. The HCTRL.COMM, HCTRL.TXEN and HCTRL.RXEN must be set in both RM to properly send and receive. The interval of the message transfer is configurable using HMSG register, similarly the timeout to receive the message for slave is configurable using HMTO register. If the health send and health message timeout is enabled then when the message is not received by the slave after the timeout the slave set itself to be master. The previous master RM set itself to be in inactive state and mask all the alarms. The redundant RM will set itself to be master and process any pending alarm. The redundant RM move from monitor to execute state.

Similar to this the health alarm timeout HATO register can be configured, in this case when an alarm is not processed for a long time the active RM set itself to be inactive and send a message to the redundant RM to go active. If the health alarm timeout is enabled in the slave then redundant RM set itself to be active and set the other to be inactive.

By using the HCTRL.FORCE register the state of active and inactive RM can be changed. The force can move an RM from execute to inactive state and inactive to execute state. Using the HCTRL.FORCE the RM can also move from monitor to execute state.

165.10 RM Error

The following errors can occur in the RM unit which are updated in the RME register, Init error IER - the initial delay (before start of alarm monitoring) timed out before the configurations are completed, Log write error LOGER - error writing the alarm log, Initial configuration read error CRER, Initial configuration write error CWER and Sequence Read Error SEQER - Error in reading the reconfiguration sequences after an alarm event. RM Error (RMER) - This field will go high when any of the above error occurs. The errors can be cleared by writing ’1’ to the corresponding register. The errors must be rectified before it can be cleared otherwise the same errors will occur. For example the init error can be rectified by properly configuring the device before timeout (CTRL.CONFDONE is high before timeout) and clear the init error. Similarly the configuration read and write error can be cleared by configuring the RM unit using external unit and clear the corresponding error bits. The configuration read and write error can also be corrected by fixing the memory fetch in which the error occurs and making a core reset by activationg CTRLRESET.

It is possible to reconfigure (switch other RM to be active) because of an error, if the CTRL.EREC is enabled. The switchover can be based on any of the following critical error - Initial configuration read error CRER, Initial configuration write error CWER and Sequence Read Error SEQER. The Switchover can happen only when the health communication is available between the two RM. If no health communication is available it is possible to provide reset using the HCTRL.RESET register. By providing the HCTRL.RESET the RM will set itself to be in default init state and the RM registers must be configured properly by an external unit to start functioning again.

165.11 Registers

The core is programmed through registers mapped into APB address space.

Table 2786. Reconfiguration Module registers

APB address offset	Register
0x00	Control
0x04	Initial Configuration Address
0x08	Reconfiguration Sequence Address
0x0C	Alarm Log Address
0x10	Watchdog Prescaler
0x14	Common Prescaler
0x18	Initial Delay Before Start Of Alarm Monitoring
0x1C	CPDU Control
0x20	Health Control
0x24	Health Message Time-Out
0x28	Health Alarm Time-Out
0x2C	Health Send Message
0x40	Alarm Mask
0x44	Alarm Level
0x48	Alarm Edge
0x4C	Prescaler For Alarm Delay
0x50 - 0x8C	Alarm 0-15 Delay
0xB0	Status
0xB4	Alarm Status
0xB8	Alarm Pending
0xBC	Health Status
0xC0	RMerror
0xD0	Watchdog Counter Nominal PM
0xD4	Software Alarm Nominal PM
0xE0	Watchdog Counter Redundant PM
0xE4	Software Alarm Redundant PM
0xF0	PM-SGM Control
0xF4	Ground-SGM Control

Table 2787. 0x00 - CTRL - Control

31 24	23 18	17 16	15 14	13 12	11 6	5	4	3	2	1	0
KEY1	RESERVED		CXWDSW	CXEALARM	RESERVED	RESET	EREC	R	ACTMST	CONF-DONE	CON-FEN
0	0		0	0	0	0	0		-	0	1
w	r		rw	rw	r	rw	rw	r	rw	rw	rw

Table 2787. 0x00 - CTRL - Control

31: 24	KEY1 = 0x55
23: 16	RESERVED
15: 14	Cross strapped watch dog and SW alarms (CXWDSW) 0b00 = None Available 0b01 = Only Nominal 0b10 = Only External 0b11 = Both Available
13: 12	Cross strapped External alarms (CXEALARM) 0b00 = None Available 0b01 = Only Nominal 0b10 = Only External 0b11 = Both Available
11: 6	RESERVED
5	reset (RESET) - Resets all the registers in the core.
4	Error reconfigure (EREC) -- If this field is one then for every error mentioned in the RME register switch over to redundant reconfiguration module will occur.
3	RESERVED
2	ACTIVE - If this field is set to one then active master. By default an external input is provided to this field which specify the system is active or not, Connected to the signal master
1	Configuration done (CONFDONE) - If the CONFEN is set to one then if the initial configuration is successfully performed this field will go high. It is not mandatory to use the RM DMA for initial configuration, the initial configuration can be performed by some other unit and this field can be enabled by the external unit performing the initial configuration. This field must be enabled by any of the above specified option and also this must be enabled before the initial time out is configured in the ID register otherwise RM error is triggered (RME.IE register). Connected to the signal confdone
0	Configuration enable (CONFEN) - External units can write 1 to this field and start the initial configuration. This field is also controller by how the core is implemented, the reset value for this field is provided by an external input signal. If the initial configuration should be performed by the RM DMA after the system reset is disabled the external input should be 1, other option is by keeping the input zero and writing this field with 1 by some other unit to start the initialization process. Connected to the signal initconf

Table 2788. 0x04 - ICA - Initial Configuration Address

31 24	23 0
KEY1	CA
	0
w	rw

31: 24	KEY1= 0x55
23: 0	Configuration address CA - The initial configuration performed by the Reconfiguration module DMA fetches the starting configuration commands from this Address.

Table 2789. 0x08 - RSA - Reconfiguration Sequence Address

31 24	23 12	11 0
KEY1	RESERVED	SA
		0x001
w		rw

Table 2789. 0x08 - RSA - Reconfiguration Sequence Address

31: 24	KEY1= 0x55
23: 12	RESERVED
11: 0	Sequence Start Address - SA - The starting address of the first reconfiguration sequence.

Table 2790. 0x0C - ALA - Alarm Log Address

31 24	23 0
KEY1	ALA
	0X400000
w	rw

31: 24	KEY1= 0x55
23: 0	Alarm Log Address - ALA - The MSB address bits point to Alarm Log Area

Table 2791. 0x10 - WDP - Watchdog Prescaler

31 24	15 0
KEY1	WP
	0x00C350
w	rw

31: 24	KEY1= 0x55
23: 16	RESERVED
15: 0	Watchdog Prescaler - WP - Prescaler for the Watchdog counters.

Table 2792. 0x14 - CP - Common Prescaler

31 24	23 0
KEY1	CP
	0x00C350
w	rw

31: 24	KEY1= 0x55
23: 0	Common Prescaler - CP - Prescaler for the delay between external commands, initial delay before start of alarm monitoring, health message time-out and health alarm time-out

Table 2793. 0x18 - ID - Initial Delay Before Start Of Alarm Monitoring

31 24	23 17	16 0
KEY1	RESERVED	DELAY
	0	0xFA00
w	r	rw

31: 24	KEY1= 0x55
23: 16	RESERVED
15: 0	Initial Delay

Table 2794. 0x1C - CC - CPDU Control

31 24	23 18	8 7	0
KEY1	RESERVED	APID	DIVCTRL
	0	0x123	0x10
w	0	rw	rw

Table 2794. 0x1C - CC - CPDU Control

31: 24	KEY1= 0x55
23: 19	RESERVED
18: 8	Application Identifier - (APID) - Sets the Application Identified in transmitted CPDU packets.
7: 0	Transmitter clock divider control - (DIVCTRL)- Contains the System clock division factor (indicates the width of the high and low phases of the outgoing PacketWire bit clock in number of system clock periods -1)

Table 2795. 0x20 - HCTRL - Health Control

31 24	23 8	7 5	4	3	2	1	0
KEY1	BAUD	RESERVED	RESET	FORCE	-	RXEN	TXEN	COMEN
	0X1B2	0	0	0	r	0	0	0
w		r	rw	rw	0	rw	rw	rw

31: 24	KEY1= 0x55
23: 8	Baud required (BAUD) - This field must provide the value corresponding to the Baud rate required which can be calculated using the system clock and the baud rate required (SYSTEMCLK/BAUD rate) Default baud value if 50 MHz and 115200 baud rate.
5: 7	RESERVED
4	Reset (RESET) - RESET - This will cause all the health registers to be in reset state and set the RM inactive.
4	Force (FORCE) Forced Switchover
3	RESERVED
2	Receive Enable - RXEN - Enable message reception when 1
1	Transmit Enable - TXEN - Enable message transmission when 1
0	Communication Enable - COMEN - Enable Communication between Master and Slave when 1

Table 2796. 0x24 - HMTO - Health Message Time-Out

31 24	23	22 16	15 0
KEY1	EN	RESERVED	HMTO
	0	0	0x1F4
w	rw	r	rw

31: 24	KEY1= 0x55
23	Enable (EN)
22: 16	RESERVED
15: 0	Health Message Time-Out - HMTO - When message transmission is enabled between the two RM a message should be received before this time-out.

Table 2797. 0x28 - HATO - Health Alarm Time-Out

31 24	23	22 16	15 0
KEY1	EN	RESERVED	HATO
	0	0	0xFFF0
w	rw	r	rw

31: 24	KEY1= 0x55
23	Enable (EN)
23: 16	RESERVED
15: 0	Health Alarm Time-Out - HATO - An active Alarm should be processed before this time-out.

Table 2798. 0x2C - HSMS - Health Send Message

31 24	23	22 15	0
KEY1	EN	RESERVED	MSG
	0	0	0xFA
w	rw	r	rw

31: 24 KEY1= 0x55

23 Enable (EN)

23: 16 RESERVED

15: 0 Health Send Message (HSMS) - Send Health message at this time interval

Table 2799. 0x40 - AM - Alarm Mask

31 24	23 20	19 0
KEY2	RESERVED	Mask
	0	0
w	r	rw

31: 24 KEY2= 0xB6

23: 20 RESERVED

19: 0 Alarm Mask (Mask) - Each alarm is masked by writing 1 to respective bits.

Fields 15 to 0 are External alarms

Fields 16 Watchdog nominal

Fields 17 Watchdog redundant

Fields 18 Software alarm nominal

Fields 19 Software alarm redundant

Table 2800. 0x44 - AP - Alarm Polarity

31 24	23 15	0
KEY2	RESERVED	POL
	0	0
w	r	rw

31: 24 KEY2= 0xB6

23: 16 RESERVED

15: 0 Alarm Polarity - POL - When 1b corresponding alarm is active when rising edge or high level else falling edge or low level. (only for External Alarm)

Table 2801. 0x48 - AE - Alarm Edge

31 24	23 x	0
KEY2	RESERVED	EDGE
	0	0
w	r	rw

31: 24 KEY2= 0xB6

23: 16 RESERVED

15: 0 Alarm Edge (EDGE) - When 1 corresponding alarm is Edge triggered else Level. (only for External Alarm)

Table 2802. 0x4C - PFAD - Prescaler For Alarm Delay

31 24	23 16	15 0
KEY2	RESERVED	PFAD
	0	0xC350
w	r	rw

Table 2802. 0x4C - PFAD - Prescaler For Alarm Delay

31: 24 KEY2= 0xB6

23: 16 RESERVED

15: 0 Prescaler (PFAD) - Prescaler for external alarm delay

Table 2803. 0x50-0x8C - AD - Alarm 0-15 Delay

31 24	23	22 16	15 0
KEY2	PRE-SEL	RESERVED	DELAY
	0	0	0
w	rw	r	rw

31: 24 KEY2= 0xB6

23 Prescaler Select - (PRESEL) - If this field is one then the external alarm delay includes the prescaler (PFAD).

22: 16 RESERVED

15: 0 DELAY

Table 2804. 0xB0 - STAT - Status

31 12	11 8	7 4	3	2	1	0
RESERVED	LOGCOUNT	STATE	R	IDD	Master	CD
	0	0		0	-	
r	r	r	r	r	r	r

31: 8 RESERVED

11: 8 LOGCOUNT - All the alarm events are logged into an external memory, this register provides the location of the pointer at which the next alarm will be written.

7: 4 STATE - Current state of the an alarm.

3 RESERVED

2 Init delay done (IDD) - Initial delay before start of alarm monitoring timeout.

1 Master - initial configuration of the core after reset.

0 Configuration Done - CD - When initial configuration is completed.

Table 2805. 0xB4 - ASTAT - Alarm Status

31 16	15 0
RESERVED	EAIN
	0
r	r

31: 16 RESERVED

15: 0 External Alarm Input (EAIN) - Direct external alarm inputs

Table 2806. 0xB8 - AP - Alarm Pending

31 0
AP
0
r

Table 2806. 0xB8 - AP - Alarm Pending

31: x RESERVED

31: 0 Alarm Pending - (AP)

Fields 15 to 0 are External alarms

Fields 16 Watchdog nominal

Fields 17 Watchdog redundant

Fields 24 to 18 Software alarm nominal

Fields 31 to 25 Software alarm redundant

Table 2807. 0xBC - HSTAT - Health Status

31 RESERVED 17	16 HSTATE	15 RX 8	7 TX 0
	0	0	0
r	r	r	r

31: 18 RESERVED

17: 16 HSTATE - Current health monitor state

RESET = "00"

EXECUTE = "01"

MONITOR = "10"

INACTIVE = "11"

15: 8 RX - Received health message

7: 0 TX - Transmitted health message

Messages

MACTIVE: = 0x94

MGOACTIVE: = 0xa2

MGOINACTIVE = 0xb1

Table 2808. 0x00 - RME - RM Error

31 KEY3 24	23 RESERVED 6	5 IER	4 LOGER	3 CRER	2 CWER	1 SEQER	0 RMER
w	r	wc	wc	wc	wc	wc	wc

31: 24 KEY3=0x3E

23: 18 RESERVED

4 Init error (IER) - The initial delay before start of alarm monitoring timed out before the configurations are completed.

4 Log write error (LOGER) - Error writing the alarm log

3 Initial configuration read error (CRER)

2 Initial configuration read error (CWER)

1 Sequence Read Error (SEQER) - Error in reading the reconfiguration sequences after an alarm event.

0 RM Error (RMER) - This field will go high when any of the above error occurs.

Table 2809. 0xD0 - WDNP - Watchdog Counter Nominal PM

31 KEY4 24	23 RESERVED 7	6 WDPM0 0
	0	0xFA
w	r	rw

Table 2809. 0xD0 - WDNP - Watchdog Counter Nominal PM

31: 24 KEY4=0xC7

23: x RESERVED

x: 0 Watchdog Time-out PM 0 - WDPM0 -

Table 2810. 0xD4 - SWNP - Software Alarm Nominal PM

31 24	23 5	4	3	2 0
KEY5	RESERVED	R	EN	SWPM0
	0	0	0	0
w	r	r	rw	rw

31: 24 KEY5=0x2A

23: 5 RESERVED

4 RESERVED

3 Enable (EN) - Should be written with 1 when a software alarm needed to be triggered, (cleared internally)

2: 0 Software Alarm PM 0 (SWPM0) - Software alarm from the nominal processor module. (any value other than 0b000 trigger corresponding software alarm)

Table 2811. 0xE0 - WDRP - Watchdog Counter Redundant PM

31 24	23 7	0
KEY6	RESERVED	WDPM1
	0	0xFA
w	r	rw

31: 24 KEY6=0xD4

23: x RESERVED

x: 0 Watchdog Time-out PM 1 - WDPM1 -

Table 2812. 0xE4 - SWRP - Software Alarm Redundant PM

31 24	23 5	4	3	2 0
KEY7	RESERVED	R	EN	SWPM1
	0	0	0	0
w	r	r	rw	rw

31: 24 KEY7=0xEB

23: 5 RESERVED

4 RESERVED

3 Enable (EN) - Should be written with 1 when a software alarm needed to be triggered, (cleared internally)

2: 0 Software Alarm PM 1 (SWPM1) - Software alarm from the redundant processor module. (any value other than 0b000 trigger corresponding software alarm)

Table 2813. 0xF0 - GSC - Ground-SGM Control

31 24	23 6	4	3	2	1	0
KEY8	RESERVED	GNDP2	GNDP1	CONT3	CONT2	CONT1
		0	0	0	0	0
w	r	rw	rw	rw	rw	rw

Table 2813. 0xF0 - GSC - Ground-SGM Control

31: 24	KEY8=0x83
23: 5	RESERVED
4	Writeprotect Area 2 from Ground access (GNDP2) - Active when set 1
3	Writeprotect Area 1 from Ground access (GNDP1) - Active when set 1
2	Active Context 3 (Cont3) - When set SGM area #3 currently contains the active context
1	Active Context 2 (Cont2) - When set SGM area #2 currently contains the active context
0	Active Context 1 (Cont1) - When set SGM area #1 currently contains the active context

Table 2814. 0xF4 - PSC - PM-SGM Control

31 24	23 4	3	2	1	0
KEY9	RESERVED	PMwP4	PMWP3	PMWP2	PMWP1
		0	0	0	0
w	r	rw	rw	rw	rw

31: 24	KEY9=0x92
23: 2	RESERVED
3	Writeprotect Area 4 from PM access (PMP4) - Active when set 1
2	Writeprotect Area 3 from PM access (PMP3) - Active when set 1
1	Writeprotect Area 2 from PM access (PMP2) - Active when set 1
0	Writeprotect Area 1 from PM access (PMP1) - Active when set 1

165.12 Vendor and device identifiers

The core has vendor identifier 0x01 (Frontgrade Gaisler) and device identifier 0x09A. For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

165.13 Implementation

165.13.1Reset

The core does not change reset behaviour depending on settings in the GRLIB configuration package (see GRLIB User's Manual). The core makes use of synchronous reset and resets a subset of its internal registers.

165.13.2Endianness

The core does not change endianness behaviour depending on the settings in the GRLIB configuration package (see GRLIB User's Manual). The core is designed for big-endian systems.

165.14 Configuration options

Table 2815 shows the configuration options of the core (VHDL generics).

*Table 2815.*Configuration options

Generic	Function	Allowed range	Default
gCONFADDR	Starting address location for the initial configuration commands	0 - 16#FFFFFF#	16#100000#0
gSEQADDR	Starting address location for the reconfiguration sequence	0 - 16#FFF#	16#300#
gLOGADDR	Starting address location for the alarm logs	0 - 16#FFFFFF#	16#400000#0
gNOEXT	Number of external alarms	2 - 16	16
gALARMPRE	Alarm delay preamble bits	1 - 24	16
gALARMDLY	Alarm delay timeout bits	1 - 23	8
gWATCHDOG	Watchdog availability (0- No watchdog alarm, 1- only nominal, 2- both nominal and redundant)	0 - 2	2
gWDOGPRE	Watchdog preamble bits	1 - 24	16
gWDOGDLY	Watchdog timeout bits	1 - 24	8
gSWALARM	Software alarm available (0- No software alarm, 1- only nominal, 2- both nominal and redundant)	0 - 2	2
gNOSW	Number of software alarms	1 - 7	7
gCOMPRE	Common preamble bits	1 - 24	24
gINITDLY	Initial timeout bits	1 - 16	16
gSYSTEMCLK	System clock frequency [Hz]	Integer	50000000
gBAUD	UART baud rate	Integer	115200
gPROG	Programmable baud rate when 1	0 - 1	1
gHEALTH	Health communication availability	0 - 1	1
gHTHBITS	Health timeout bits	1 - 23	16
gBUFTYPE	Type of RAM used (0 - Generic RAM interface selected, 1 - Gaisler RAM interface selected)	0	0
gPINDEX	APB slave index	0 - NAPBSLV-1	0
gHINDEX	AHB master index	0 - NAHBMST-1	0
gPADDR	ADDR field of the APB BAR	0 - 16#FFF#	0
gPMASK	MASK field of the APB BAR	0 - 16#FFF#	16#FFF#
gPIRQ	Not implemented	0 - NAPBIRQ	1

165.15 Signal descriptions

Table 2816 shows the interface signals of the core (VHDL ports).

*Table 2816.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
EXTALARM	N/A	Input	External alarm input signals	-
EXTTIME	N/A	Input	External time input signals	-
PKTWREADY	N/A	Input	Ready for packet	High
PKTWBUSY_N	N/A	Input	Not ready for octet	Low
PKTWCLK	N/A	Output	Bit clock	-
PKTWDATA	N/A	Output	Data	-
PKTWVALID	N/A	Output	Delimiter	High
PKTWABORT	N/A	Output	Abort	High
HEALTHRX	N/A	Input	UART receiver data	-
HEALTHTX	N/A	Output	UART transmit data	-
MASTER	N/A	Input	Select RM master or slave	High
INITCONF	N/A	Input	To enable initial configuration externally	High
CONFDONE	N/A	Input	If initial configuration not required tie this signal to active High	High
ALARM_EVT	N/A	Output	AHB master output signals	-
ALARMLOG	N/A	Output	AHB master input signals	High
RMERROR	N/A	Output	AHB master output signals	High
GNDWPRT2	N/A	Output	This signal reflects the contents of the GSC.GNDP2 register	High
GNDWPRT1	N/A	Output	This signal reflects the contents of the GSC.GNDP1 register	High
PMWPRT4	N/A	Output	This signal reflects the contents of the PSC.PMWP4 register	High
PMWPRT3	N/A	Output	This signal reflects the contents of the PSC.PMWP3 register	High
PMWPRT2	N/A	Output	This signal reflects the contents of the PSC.PMWP2 register	High
PMWPRT1	N/A	Output	This signal reflects the contents of the PSC.PMWP1 register	High

* see GRLIB IP Library User's Manual

165.16 Signal definitions and reset values

The signals and their reset values are described in table 2817.

*Table 2817.*Signal definitions and reset values

Signal name	Type	Function	Active	Reset value
txd[]	Output	UART transmit data line	-	Logical 1
rtsn[]	Output	Ready To Send	Low	Logical 1
rxd[]	Input	UART receive data line	-	-
ctsn[]	Input	Clear To Send	Low	-

165.17 Timing

The timing waveforms and timing parameters are shown in figure 406 and are defined in table 2818.

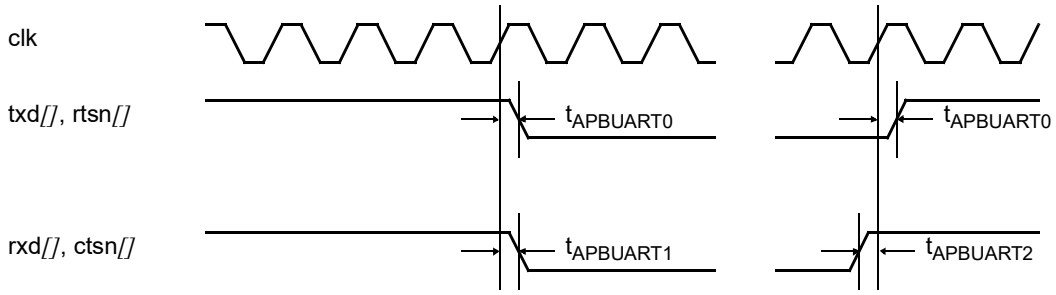


Figure 406. Timing waveforms

*Table 2818.*Timing parameters

Name	Parameter	Reference edge	Min	Max	Unit
$t_{APBUART0}$	clock to output delay	rising *clk* edge	TBD	TBD	ns
$t_{APBUART1}$	input to clock hold	rising *clk* edge	-	-	ns
$t_{APBUART2}$	input to clock setup	rising *clk* edge	-	-	ns

Note: The *ctsn[]* and *rxd[]* inputs are re-synchronized internally. These signals do not have to meet any setup or hold requirements.

165.18 Library dependencies

Table 2819 shows libraries that should be used when instantiating the core.

*Table 2819.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	APB signal definitions
TMTC	GRRM	Signals, component	Signal and component declaration

165.19 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
```

```
library gaisler;
use gaisler.grreconfigmodule.all;
entity grrm_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- external alarms
    extalarm:   in std_logic_vector(gNOEXT-1 downto 0);
    -- external time
    exttime:    in std_logic_vector(54 downto 0);

    -- packetwire interface
    pktwready:  in  std_logic;
    pktwbusy_n: in  std_logic;
    pktwclk:    out std_logic;
    pktwdata:   out std_logic;
    pktwvalid:  out std_logic;
    pktwabort:  out std_logic;
    -- health communication
    healthrx:   in  std_logic;
    healthtx:   out std_logic;
    );
end;

architecture rtl of apbuart_ex is

  -- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

begin

  -- AMBA Components are instantiated here
   ...
rm0: grrm
      generic map (
         gCONFADDR       => 16#000400#,
         gSEQADDR        => 16#000#,
         gLOGADDR        => 16#400001#,
         gNOEXT          => 16,
         gALARMPRE       => 16,
         gALARMDLY       => 16,
         gWATCHDOG       => 2,
         gWDOGPRE        => 16,
         gWDOGDLY        => 8,
         gSWALARM        => 2,
         gNOSW           => 7,
         gCOMPRE         => 24,
         gINITDLY        => 16,
         gSYSTEMCLK      => 50000000,
         gBAUD           => 115200,
         gPROG           => 1,
         gHEALTH         => 1,
         gHTHBITS        => 16,
         gBUFTYPE        => 0,
         gPINDEX         => 10,
         gHINDEX         => 5,
         gPADDR          => 10,
         gPMASK          => 16#FFF#,
         gPIRQ           => 10)
      port map (
        rstn             => rstn,
        clk              => clk,
        apbi             => apbi,
        apbo             => apbo(10),
        ahbmi            => ahbmi,
        ahbmo            => ahbmo(5),
        extalarm         => extalarm,
        exttime          => exttime,
        pktwclk          => pktwclk,
```

```
pktwdata          => pktwdata,
pktwvalid         => pktwvalid,
pktwabort         => pktwabort,
pktwready         => pktwready,
pktwbusy_n        => pktwbusy_n,
healthrx          => healthrx,
healthtx          => healthtx,
master            => dipsw(6),
initconf          => dipsw(7),
confdone          => '0',
alarm_evt         => open,
alarmlog          => alarmlog,
rmerror           => rmerror,
gndwprt2          => gndwprt2,
gndwprt1          => gndwprt1,
pmwprt4           => pmwprt4,
pmwprt3           => pmwprt3,
pmwprt2           => pmwprt2,
pmwprt1           => pmwprt1);
```

Frontgrade Gaisler AB
Kungsgatan 12
411 19 Göteborg
Sweden
www.frontgrade.com/gaisler
sales@gaisler.com
T: +46 31 7758650

Frontgrade Gaisler AB, reserves the right to make changes to any products and services described herein at any time without notice. Consult the company or an authorized sales representative to verify that the information in this document is current before using this product. The company does not assume any responsibility or liability arising out of the application or use of any product or service described herein, except as expressly agreed to in writing by the company; nor does the purchase, lease, or use of a product or service from the company convey a license under any patent rights, copyrights, trademark rights, or any other of the intellectual rights of the company or of third parties. All information is provided as is. There is no warranty that it is correct or suitable for any purpose, neither implicit nor explicit.

Copyright © 2025 Frontgrade Gaisler AB