GR716B-BOARD
Quick Start Guide

USER MANUAL

RELEASED MAY 2026

Frontgrade Gaisler AB | Kungsgatan 12 | SE-411 19 | Göteborg | Sweden | frontgrade.com/gaisler

Table of Contents

1. Introduction

1.1. Overview

This document is a quick start guide for the GR716B BOARD Development Board.

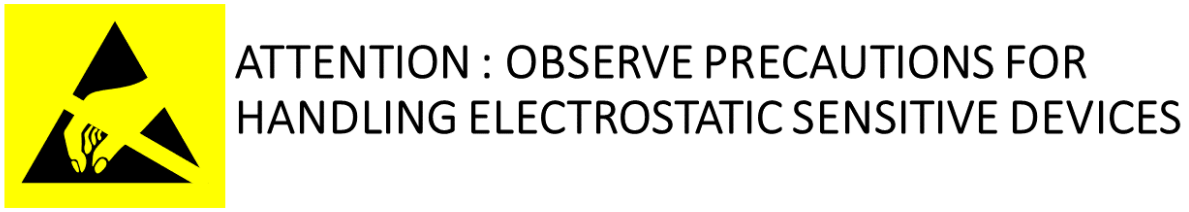
The purpose of this document is to get users quickly started using the board.

For a complete description of the board please refer to the GR716B-BOARD Development Board User's Manual.

The GR716B system-on-chip is described in the GR716B Data sheet and User's Manual.

This quick start guide does not contain as many technical details and is instead how-to oriented. However, to make the most of the guide the user should have glanced through the aforementioned documents and should ideally also be familiar with the GRMON debug monitor.

1.2. Handling



This unit contains sensitive electronic components which can be damaged by Electrostatic Discharges (ESD). When handling or installing the unit observe appropriate precautions and ESD safe practices.

When not in use, store the unit in an electrostatic protective container or bag.

When connecting/disconnecting cables, ensure that the unit is in an un-powered state.

This equipment has SpW ports that use Low Voltage Differential Signalling (LVDS) which has limited common mode voltage protection. Please refer to the user’s manual for instructions on how to ensure that the grounds of equipment are connected together when using SpaceWire.

1.3. References

Table 1.1. References

RD-1	GR716B-BOARD Development Board User's Manual
RD-2	GR716B Data sheet and User's Manual [https://www.gaisler.com/products/gr716b]
RD-4	GRMON User's Manual [https://www.gaisler.com/doc/grmon4.pdf]
RD-5	TSIM User's Manual [https://gaisler.com/index.php/products/simulators]
RD-10	Bare C Cross-Compilation System [https://www.gaisler.com/index.php/products/operating-systems/bcc]
RD-11	BCC User's Manual [https://www.gaisler.com/doc/bcc2.pdf]

The referenced documents can be downloaded from https://www.gaisler.com.

2. Overview

The GR716B BOARD Development Board provides a comprehensive and rapid software prototyping platform for the GR716B LEON3FT Microcontroller. The PC/104 style stackable headers (2 x 64 pin) allow for easy expansion, accessibility and integration. Along with the microcontroller, the subject board supports the following features:

- Processor
  - GR716B microcontroller in CQFP132 Package
- Memory
  - SPI Flash (up to 512 Mbit)
- Power Supply
  - Power from External supply (range 5V to 14.5V)
  - Options for single supply operation or individual external supply
- Interfaces
  - PC/104 style stackable headers (2 x 64 pin) for GPIO and interface signals
    - Provides access to all GPIO/interface signals for monitoring/debug
    - Provides stacking interface to User Defined modules (Memory, Digital, Analog I/F)
    - Provides stacking interface to GR716-CPCI-DEV development board in 6U rack or box format
  - Serial debug using GRMON4
  - LVDS in/out (4 + 4 pairs) for 2 x SPW interface or 1 x SPI4S
  - 64 GPIO pins with support for for: Digital I/O, 6 x UART, external memory interface, MIL-1553, CAN, Ethernet, FPGA Scrubber, I2C, 3 x SPI I/F, 16 x Analog In, 4 x Analog DAC out, ACOMP, APWM outputs, SpaceWire.
  - Crystal (20 MHz)
  - Option for External Clocks
  - DIP switch for Bootstrap options

The board has the dimension of 80 x 100 mm.

Figure 2.1. GR716B-BOARD

Table 2.1. Major board items

Item	Description	Section
D1	Reset LED	3.5
D2	3V3 power supply LED	3.3
D2	1V8 power supply LED	3.3
J1	Power connector for standalone operation	3.3
J2	Alternative power connector	3.3.1
Y1	Socket for external crystal.	3.6
JP1 - JP9	Power supply source configuration	3.3
JP10 - JP11	Select system clock source	3.6
P1, P2	GR716B-BOARD connector	9
S1, S2	Boot configuration switch	3.4

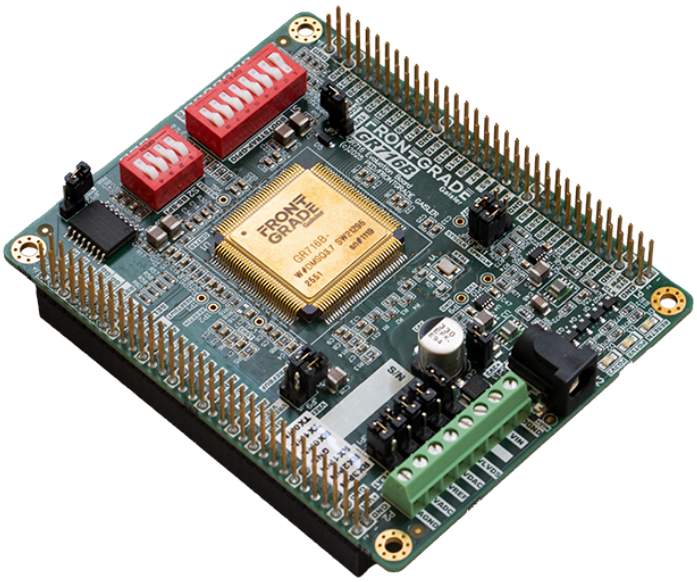


Figure 2.2. GR716B BOARD Development Board

3. Board Configuration

3.1. Overview

The primary sources of information for board configuration is the GR716B-BOARD Development Board User's Manual ([RD-1] and the GR716B Data sheet and User's Manual ([RD-2]). Before start using the GR716B-BOARD, clock sources have to be installed, bootstrap signals need to be set correctly and the desired interfaces have to be enabled.

GR716B shares some of its IO signals due to a limited number of package pins. For that reason, the pin multiplexing has to be set up depending on the desired interfaces and memory configuration.

3.2. Default configuration

This guide provides a default configuration which uses

- SPI flash PROM as boot memory.
- UART over FTDI as debug link.
- Some General I/O and SPI flash PROM are enabled.

The complete default configuration can be found in Appendix B and GR716B-BOARD Development Board User's Manual. If this is your first time using the GR716B-BOARD, please use this configuration as a starting point.

Default configuration

To achieve the default configuration please follow the instructions on each box note like this one.

3.3. Power

A single supply with a +5V (minimum) / +14.5V (maximum) is required to power the board. All other board voltages are derived from this input using on-board discrete power circuits.

Default configuration

Default power configuration for stand-alone board is to supply the J2 connector with 5V.

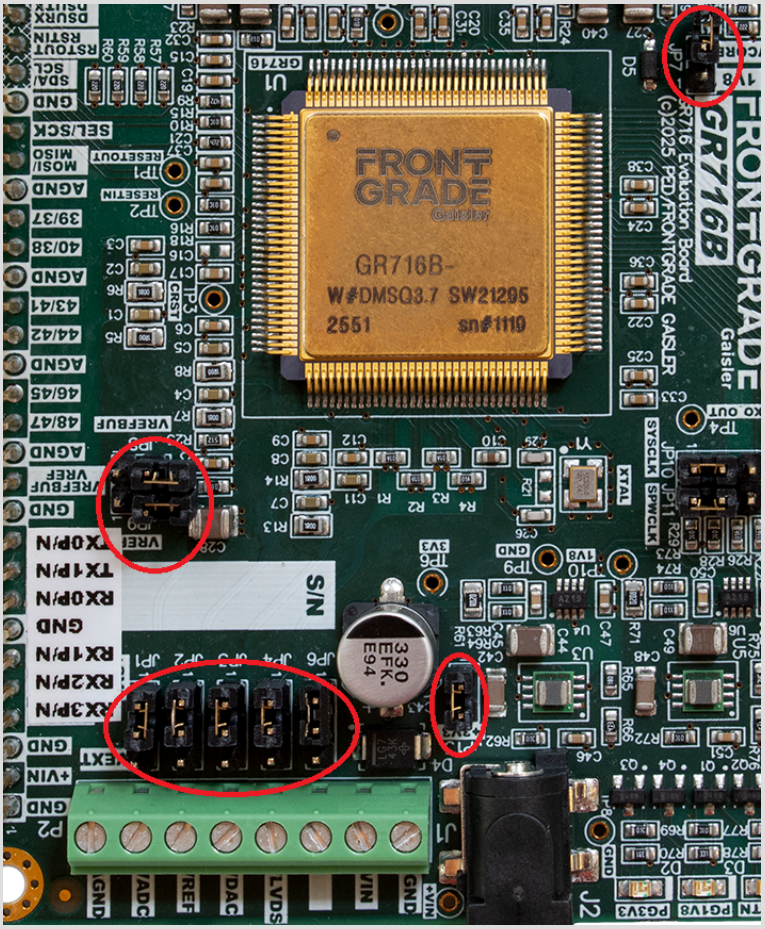


Figure 3.1. GR716B-BOARD default power configuration jumpers and unconnected terminal block (green).

The default configuration of the board uses (single 3.3V supply):

- J1 should be open i.e. no connection to external supply or ground plane
- JP1 should connect pin position 1 and 2
- JP2 should connect pin position 1 and 2
- JP3 should connect pin position 1 and 2
- JP4 should connect pin position 1 and 2
- JP5 is removed and not available in this board
- JP6 should connect pin position 1 and 2
- JP7 should be open i.e. no connection between any pins
- JP8 should be open i.e. no connection between any pins
- JP9 should be open i.e. no connection between any pins
- JP13 should connect pin position 1 and 2

Always leave JP8 *open* when connecting GR716B-BOARD with the GR716B-DEV board.

If the GR716B is to be operated from both the POL generated 3.3V and 1.8V supplies on board, then JP6 should be set to position 2-3 and JP7 installed. In this situation, the internal LDO is disabled and VDDCORE voltage of 1.8V is provided from the POL regulator.

3.3.1. Alternative power

The +5V (minimum) / +14.5V (maximum) can be supplied via the PC/104 style stackable headers.

Do NOT supply the board via the the connector J2 and the PC/104 style stackable headers at the same time.

3.4. Bootstrap

Bootstrap signals configure the chip at reset and are listed in the section *Bootstrap signals* of GR716B Data sheet and User's Manual. All of these signals can be controlled on the GR716B-BOARD via the SPST DIP switches S1 and S2.

Each of the DIP switches have two positions: pull-down (PD), pull-up (PU).

The bootstrap signals controlled via SPST DIP switches S1 and S2 have the following impact on the system behaviour:

- Remote boot access (S2-3) enables remote access to the GR716B microcontroller after initialization has been completed. Enable of remote access will force the processor to power down after initialization.
- Disable use of internal boot ROM (S1-4), disables initialization of processor and internal memories. If internal boot ROM is disabled, the processor will start execute software directly from selected source or power down after a remote interface has been enabled.
- Select boot source (S2-1, S2-2) will together with the remote boot access (S2-3) switch select boot source. This pin has dual functionality which deepeneds on boot strap configuration with higher precedence
- Enable use of ASW image format (S1-8). This enables the on-chip ROM application loader which can copy application sections from external memories (SRAM, PROM, SPI flash or I2C memory) to the local RAM. Application image integrity is verified using a CRC-16 algorithm.
- Enable use of redundant memory (S1-7). When ASW image format (S1-8) is enabled, the system can be configured to start from redundant memory when an error occurs on the first application image.
- EDAC for external memories (S1-1) enables error detection and correction for external SRAM, PROM and SPI memory. When this bit is enabled system expects correction codes to be stored together with application software. Active low enable.
- When the GR716B microcontroller shall boot via a remote source. Switch (S1-1) enables internal SpaceWire frequency generation using the PLL. If disabled the internal PLL is bypassed.
- Select SpaceWire default frequency (S1-8, S1-7).
- (S1-3) (S1-6) Determines bit 2 and bit 1 of the CAN-FD and I2C node ID.
- (S1-2) Determines the selection of nominal or redundant bus for CAN-FD. Set to low for nominal bus, and high for redundant bus.

Default configuration

Default configuration is to start execute application software from the SPI memory after internal ROM initialization software has executed. The ASW container format is not used by default.

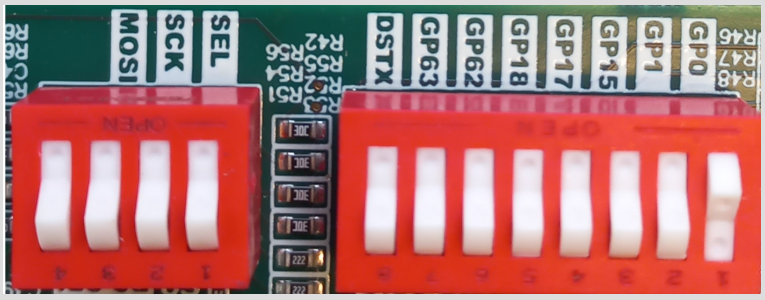


Figure 3.2. GR716B-BOARD default bootstrap configuration

The default configuration of the board uses:

- S1-1 PU
- S1-2 PD
- S1-3 PD
- S1-4 PD
- S1-5 PD
- S1-6 PD
- S1-7 PD
- S1-8 PD
- S2-1 PD
- S2-2 PD
- S2-3 PD
- S2-4 RESERVED (not used)

For informatoin on how to use S1 and S2 to select other processor boot options, see [RD-1].

3.5. Reset

Default configuration

The default configuration is to use the internal *Power-On-Reset* functionality. The input signal RESET_IN_N should be left floating.

3.5.1. External Reset and break switch

Reset switch, Break switch and UART to USB debug connection is supported via the USB-JTAG Programming module. The USB-JTAG programming module is connected to connector P2. See Figure 3.3.

3.6. Clock

The default configuration of the board is to use the external 20 MHz crystal oscillator (crystal mount on board, oscillator from GR716B on-chip) as system and SpaceWire clock.

The internal frequencies for ADC, DAC, MIL-1553, SpaceWire and System depend on the crystal oscillator frequency and on the internal clock logic and PLL configuration. External system and SpaceWire clock source is selected with JP10 and JP11. If SpaceWire is selected as remote boot option, the PLL should be configured via the SPST DIP switch S1 position 7 and 8.

Table 3.1. System and SpaceWire clock configuration

Y1 (MHz)	JP10-{1,2}	JP10-{2,3}	JP11-{1,2}	JP11-{2,3}	Comment
50, 10, 20, 25	Closed	Open	Closed	Open	Allows remote boot over SpaceWire

Default configuration

Default configuration is to use 20 MHz as System and SpaceWire clock.

- JP10 should connect pin position 1 and 2
- JP11 should connect pin position 1 and 2

3.7. Pin multiplexing

GR716B shares IO between memory and communication interfaces due to limited number of package pins. A complete description is given in the GR716B Data sheet and User's Manual.

There are a number things to take into consideration when configuring the pin multiplexing:

- Boot strap option might cause sub-sections of shared pins to be used for memories or remote access.
- GR716B-BOARD might be connected to hardware not aware of the GR716B e.g. when user has connected the GR716B-BOARD with a GR-CPCI-GR716-DEV board.

Pin muxing configuration is set in the *System IO configuration* registers described in the section *Configuration Registers* in the GR716B Data sheet and User's Manual.

The *System IO configuration registers* are updated after reset to reflect the system configuration. E.g. if the system is configured to use external SRAM memory, a number of *System IO configuration registers* are set to the value `0x2` to select SRAM for pin location.

It is recommended to preserve the configuration of unrelated pins when updating a pin IO configuration.

3.8. Pin properties

There are built-in pullup and pulldown resistor in the GR716B microcontroller which can be programmed when the system is running. These built-in resistors are accessed by setting pin mode to input-pullup or input-pulldown mode.

Pullup and pulldown resistors should only be enabled in input mode and must not be enabled in analog mode.

3.9. Debug communication link

The UART debug communication link (AHBUART) is available via the PC/104 style stackable headers. An external FTDI FT423HL adapter board can provide a UART to USB conversion supported by GRMON4 (Chapter 5).

Do NOT connect the USB cable to the host computer until the board is supplied via the connector J2 or PC/104 style stackable headers.

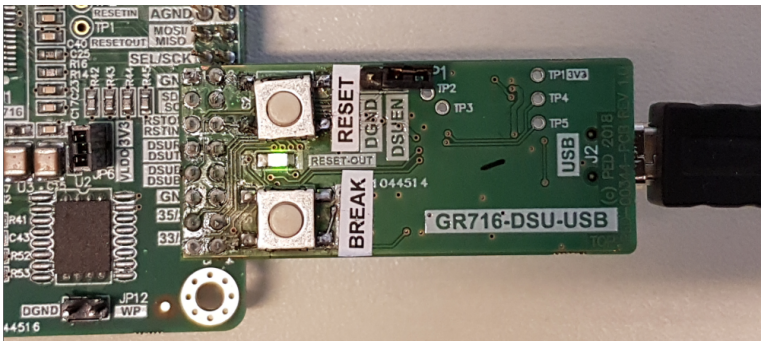


Figure 3.3. Connect to UART debug communication link using the FTDI adapter board

For more information on the external FTDI FT423HL chip board, see Section 9.1.

4. Software Development Environment

4.1. Overview

Frontgrade Gaisler provides a comprehensive set of software tools to run several different operating systems. The GR716B platform supports the following:

BCC	the Bare C Cross-Compiler System is a toolchain to compile bare C or C++ applications directly on top of the processor without the services provided by an operating system

Frontgrade Gaisler also provides a set of debug tools. The GR716B platform is supported by the following:

GRMON	Used to run and debug applications on GR716B-BOARD hardware. See (Chapter 5).
TSIM	Used to run and debug applications on a simulated GR716B-BOARD. See (Chapter 6).

TSIM is mainly used when no hardware is available. It can also be integrated into larger simulation networks to simulate, for example, entire satellite systems. TSIM provides precise code coverage capture and large instruction/bus trace buffers.

Developer tools are generally provided for both Linux and Windows host operating systems. Frontgrade Gaisler also provides an integrated, easy-to-use solution to help programmers with the task of developing for the LEON. The LEON Integrated Development Environment for Eclipse (LIDE) is an Eclipse plug-in integrating compilers, software and hardware debuggers in a graphical user interface. The plugin makes it possible to cross-compile C and C++ application for LEON, and to debug them on either simulator and target hardware (TSIM or GRMON4).

The recommended method to load software onto a LEON board is by connecting to a debug interface of the board through the GRMON4 hardware debugger (Chapter 5). Execution of programs by a PROM-loaded boot loader is also possible.

5. GRMON4 hardware debugger

5.1. Overview

GRMON4 is a debug monitor used to develop and debug GRLIB/LEON systems. The target system, including the processor and peripherals, is accessed on the AHB bus through a debug-link connected to the host computer. GRMON4 has GDB support which makes C/C++ level debugging possible by connecting GDB to the GRMON4's GDB socket. With GRMON4 one can for example:

- Inspect LEON and peripheral registers
- Upload applications to RAM with the **load** command.
- Program the FLASH with the **flash** command.
- Control execution flow by starting applications (**run**), continue execution (**cont**), single-stepping (**step**), inserting breakpoints/watchpoints (**bp**) etc.
- Inspect the current CPU state listing the back-trace, instruction trace and disassemble machine code.

The first step is to set up a debug link in order to connect to the board. The following section outlines which debug interfaces are available and how to use them on the GR716B BOARD Development Board. After that, a basic first inspection of the board is exemplified.

GRMON4 is described on the homepage [https://www.gaisler.com/index.php/products/debug-tools] and in detail in [RD-4].

It is recommended to use version GRMON4 or later with GR716B.

5.2. Debug links

5.2.1. Connecting via the FTDI USB/UART interface

Please see GRMON User's Manual for instructions how to connect GRMON4 to a board using a serial UART connection. The PC is connected using a USB cable to the J1 connector and then starting GRMON4 with the **-uart PORTNAME** debug-link option and device name. For example:

```
grmon -uart /dev/ttyUSB0
```

It is recommended to use the GRMON4 command line option **-baud 230400** to increase the AHBUART debug link speed.

5.3. First steps

The previous sections have described which debug-links are available and how to start using them with GRMON4. The subsections below assume that GRMON4, the host computer and the GR716B-BOARD board have been set up so that GRMON4 can connect to the board.

When connecting to the board for the first time it is recommended to get to know the system by inspecting the current configuration and hardware present using GRMON4. With the **info sys** command more details about the system is printed and with **info reg** the register contents of the I/O registers can be inspected. Below is a list of items of particular interest:

- AMBA system frequency is printed out at connect, if the frequency is wrong then it might be due to noise in auto detection (small error). See `-freq` flag in the GRMON User's Manual [RD-4].
- Memory location and size configuration is found from the **info sys** output.
- The GR716B has a clock-gating unit which is able to disable/enable clocking and control reset signals. Clocks must be enabled for all cores that LEON software or GRMON4 will be using. The **grcg** command is described in the GRMON User's Manual [RD-4].

5.4. Connecting to the board

In the following example the UART (over USB) debug-link is used to connect to the board. The auto-detected frequency, memory parameters and stack pointer are verified by looking at the GRMON4 terminal output below.

GRMON4 is started with the **-u** and **–cginit 0x00010000** options in order to redirect UART output to the GRMON4 terminal.

```
$ grmon -u -cginit 0x00010000 -uart /dev/ttyUSB0
  GRMON debug monitor v4.0.0-4-gc3dfb26 64-bit pro version

  Copyright (C) 2024 Frontgrade Gaisler - All rights reserved.
  For latest updates, go to https://www.gaisler.com/
  Comments or bug-reports to support@gaisler.com

  using port /dev/ttyUSB0 @ 115200 baud
  Device ID:           0x716
  GRLIB build version: 4282
  Detected system:     GR716B
  Detected frequency:  20.5 MHz

  Component                           Vendor
  AHB-to-AHB Bridge                   Frontgrade Gaisler
  MIL-STD-1553B Interface             Frontgrade Gaisler
  GRSPW Router DMA interface          Frontgrade Gaisler
  SPI to AHB Bridge                   Frontgrade Gaisler
  I2C to AHB Bridge                   Frontgrade Gaisler
  CAN-FD Controller with DMA          Frontgrade Gaisler
  AHB Debug UART                      Frontgrade Gaisler
  AHB-to-AHB Bridge                   Frontgrade Gaisler
  PacketWire Receive Interface        Frontgrade Gaisler
  PacketWire Transmitter with DMA     Frontgrade Gaisler
  GRDMAC2 DMA Controller              Frontgrade Gaisler
  GR Ethernet MAC                     Frontgrade Gaisler
  Dual-port SPI Slave                 Frontgrade Gaisler
  LEON3FT SPARC V8 Processor          Frontgrade Gaisler
  AHB-to-AHB Bridge                   Frontgrade Gaisler
  AHB Memory Scrubber                 Frontgrade Gaisler
  GRSCRUB FPGA Scrubber               Frontgrade Gaisler
  GR716 Analog PWM                    Frontgrade Gaisler
  LEON3 SPARC V8 Processor            Frontgrade Gaisler
  GR716 RTA AHB Interface             Frontgrade Gaisler
  LEON3 SPARC V8 Processor            Frontgrade Gaisler
  GR716 RTA AHB Interface             Frontgrade Gaisler
  AHB-to-AHB Bridge                   Frontgrade Gaisler
  AHB Debug UART                      Frontgrade Gaisler
  Dual-port AHB(/CPU) On-Chip RAM     Frontgrade Gaisler
  Dual-port AHB(/CPU) On-Chip RAM     Frontgrade Gaisler
  Generic AHB ROM                     Frontgrade Gaisler
  Memory controller with EDAC         Frontgrade Gaisler
  SPI Memory Controller               Frontgrade Gaisler
  SPI Memory Controller               Frontgrade Gaisler
  AHB/APB Bridge                      Frontgrade Gaisler
  AHB/APB Bridge                      Frontgrade Gaisler
  AHB/APB Bridge                      Frontgrade Gaisler
  AHB/APB Bridge                      Frontgrade Gaisler
  Memory controller with EDAC         Frontgrade Gaisler
  GRSPW Router                        Frontgrade Gaisler
  AHB/APB Bridge                      Frontgrade Gaisler
  AHB/APB Bridge                      Frontgrade Gaisler
  AHB/APB Bridge                      Frontgrade Gaisler
  AHB/APB Bridge                      Frontgrade Gaisler
  AHB/APB Bridge                      Frontgrade Gaisler
  AHB/APB Bridge                      Frontgrade Gaisler
  AMBA Trace Buffer                   Frontgrade Gaisler
  LEON3 Debug Support Unit            Frontgrade Gaisler
  GR716 RTA Debug Support Unit        Frontgrade Gaisler
  GR716 RTA Debug Support Unit        Frontgrade Gaisler
  Multi-processor Interrupt Ctrl.     Frontgrade Gaisler
  Modular Timer Unit                  Frontgrade Gaisler
  Modular Timer Unit                  Frontgrade Gaisler
  GR716 AMBA Protection unit          Frontgrade Gaisler
  Clock gating unit                   Frontgrade Gaisler
  Clock gating unit                   Frontgrade Gaisler
  General Purpose Register Bank       Frontgrade Gaisler
  LEON3 Statistics Unit               Frontgrade Gaisler
  AHB Status Register                 Frontgrade Gaisler
  CCSDS TDP / SpaceWire I/F           Frontgrade Gaisler
  General Purpose Register Bank       Frontgrade Gaisler
  General Purpose Register            Frontgrade Gaisler
  GR716 AMBA Protection unit          Frontgrade Gaisler
  GR716 Phase-locked loop             Frontgrade Gaisler
  GR716 Brownout detector             Frontgrade Gaisler
  PacketWire Receiver with DMA        Frontgrade Gaisler
  General Purpose Register Bank       Frontgrade Gaisler
  Generic UART                        Frontgrade Gaisler
  Generic UART                        Frontgrade Gaisler
  Generic UART                        Frontgrade Gaisler
  Generic UART                        Frontgrade Gaisler
  Generic UART                        Frontgrade Gaisler
  Generic UART                        Frontgrade Gaisler
  AHB Status Register                 Frontgrade Gaisler
```

```
  SPI Controller                      Frontgrade Gaisler
  SPI Controller                      Frontgrade Gaisler
  General Purpose I/O port            Frontgrade Gaisler
  General Purpose I/O port            Frontgrade Gaisler
  AMBA Wrapper for OC I2C-master      Frontgrade Gaisler
  AMBA Wrapper for OC I2C-master      Frontgrade Gaisler
  General Purpose Register Bank       Frontgrade Gaisler
  GR716 Analog-to-Digital Converter   Frontgrade Gaisler
  GR716 Analog-to-Digital Converter   Frontgrade Gaisler
  GR716 Analog-to-Digital Converter   Frontgrade Gaisler
  GR716 Analog-to-Digital Converter   Frontgrade Gaisler
  GR716 Digital-to-Analog Converter   Frontgrade Gaisler
  GR716 Digital-to-Analog Converter   Frontgrade Gaisler
  GR716 Digital-to-Analog Converter   Frontgrade Gaisler
  GR716 Digital-to-Analog Converter   Frontgrade Gaisler
  I2C Slave                           Frontgrade Gaisler
  I2C Slave                           Frontgrade Gaisler
  LEON3 Statistics Unit               Frontgrade Gaisler
  General Purpose Register Bank       Frontgrade Gaisler
  General Purpose Register Bank       Frontgrade Gaisler
  General Purpose Register            Frontgrade Gaisler
  General Purpose Register            Frontgrade Gaisler

  Use command 'info sys' to print a detailed report of attached cores

grmon4> info sys cpu0
  cpu0       Frontgrade Gaisler  LEON3FT SPARC V8 Processor
             AHB Master 0

grmon4> info sys cpu1
  cpu1       Frontgrade Gaisler  LEON3FT SPARC V8 Processor
             AHB Master 0
             Device is disabled

grmon4> info sys cpu2
  cpu2       Frontgrade Gaisler  LEON3FT SPARC V8 Processor
             AHB Master 0
             Device is disabled
```

It is possible to limit the output to certain cores by specifying the core(s) name(s) to the **info sys** and **info reg** commands. Below is information listed for the first UART and the first timer core. Registers for the first AHB-STAT are printed in verbose format.

```
  grmon4> info sys uart0 gptimer0
  uart0     Frontgrade Gaisler  Generic UART
            APB: 80300000 - 80300100
            IRQ: 24
            Baudrate 38461, FIFO debug mode available
  gptimer0  Frontgrade Gaisler  Modular Timer Unit
            APB: 80003000 - 80003100
            IRQ: 9
            16-bit scalar, 7 * 32-bit timers, divisor 21

  grmon4> info reg -v ahbstat0
  AHB Status Register
      0x8000a000  Status register                          0x00000002
      13     me               0x0         Multiple Error detection
      12     fw               0x0         Filter Write
      11     cf               0x0         Correctable Error Filter
      10     af               0x0         AMBA ERROR Filter
       9     ce               0x0         Correctable error
       8     ne               0x0         New error
       7     hw               0x0         HWRITE on error
       6:3   hm               0x0         HMASTER on error
       2:0   hs               0x2         HSIZE on error

      0x8000a004  Failing address register               0x8000a004
```

The GR716B has a clock-gating unit which can disable and enable clock gating and generate reset signals of certain cores in the SOC. With the GRMON4 **grcg** command the current setting of the clock-gating unit can be inspected and changed, the command line switch `-cginit` also affects the clock-gating unit. See [RD-4] for more information. Below is an example where the first SPI master controller clocks are enabled (not gated). The `APB UART0` clock was enabled on the GRMON4 command line.

```
  grmon4> grcg
                        GR716B Primary  GR716B Secondary
  Unlock register:      0x00000000      0x00000000
  Clock enable register: 0x00010000     0x00000000
  Reset register:       0xfffeffff      0xffffffff

   GATE  CORE(S)   DESCRIPTION                UNLOCKED  ENABLED  RESET
     0   SPI2AHB   SPI to AHB Bridge             0         0       1
```

```
  1   I2C2AHB   I2C to AHB Bridge              0         0       1
  2   PWRX      PacketWire RX Interface        0         0       1
  3   PWTX      PacketWire TX Interface        0         0       1
  4   FTMCTRL   Memory ctrl with EDAC 0        0         0       1
  5   SPIMCTRL  SPI Memory Controller 0        0         0       1
  6   SPIMCTRL  SPI Memory Controller 1        0         0       1
  7   SPICTRL   SPI Controller 0               0         0       1
  8   SPICTRL   SPI Controller 1               0         0       1
  9   I2CMST    I2C-master 0                   0         0       1
 10   I2CMST    I2C-master 1                   0         0       1
 11   I2CSLV    I2C Slave 0                    0         0       1
 12   I2CSLV    I2C Slave 1                    0         0       1
 13   ACOMP     GR716 Analog Comparator        0         0       1
 14   CLKDET    GR716 APWM Clk error det       0         0       1
 16   APBUART   APB UART 0                     0         1       0
 17   APBUART   APB UART 1                     0         0       1
 18   APBUART   APB UART 2                     0         0       1
 19   APBUART   APB UART 3                     0         0       1
 20   APBUART   APB UART 4                     0         0       1
 21   APBUART   APB UART 5                     0         0       1
 23   APWM      APWM subsystem                 0         0       1
 24   L3STAT    LEON3 Statistics Unit          0         0       1
 25   AHBUART   AHB Debug UART                 0         0       1
 26   MEMPROT   AMBA Protection unit          0         0       1
 28   SPWTDP    CCSDS TDP                      0         0       1
 29   SPI4S     SPI for Space Slave            0         0       1
 30   NVRAM     FT NVRAM Memory Interface      0         0       1
 31   GPIO      General Purpose I/O port 0     0         0       1
 32   GRDMAC    DMA Controller 0               0         0       1
 33   GRDMAC    DMA Controller 1               0         0       1
 34   APWM      Analog applications PWM        0         0       1
 35   GRETH     GR Ethernet MAC                0         0       1
 36   GR1553B   MIL-STD-1553B Interface        0         0       1
 37   GRCANFD   CAN-FD Controller with DMA     0         0       1
 38   LVDSIO    LVDS IO                        0         0       1
 39   SPWRTR    GRSPW Router                   0         0       1
 40   DAC       Digital-to-Analog 0            0         0       1
 41   DAC       Digital-to-Analog 1            0         0       1
 42   DAC       Digital-to-Analog 2            0         0       1
 43   DAC       Digital-to-Analog 3            0         0       1
 44   ADC       Analog-to-Digital 0            0         0       1
 45   ADC       Analog-to-Digital 1            0         0       1
 46   ADC       Analog-to-Digital 2            0         0       1
 47   ADC       Analog-to-Digital 3            0         0       1
 54   GRSCRUB   GRSCRUB FPGA Scrubber          0         0       1
 55   RTA       Real-Time Accelerator 0        0         0       1
 56   RTA       Real-Time Accelerator 1        0         0       1
 57   APWMDAC   GR716 APWM DAC 0               0         0       1
 58   APWMDAC   GR716 APWM DAC 1               0         0       1
 59   APWMDAC   GR716 APWM DAC 2               0         0       1
 60   APWMDAC   GR716 APWM DAC 3               0         0       1
 61   FIR       GR716 APWM FIR1                0         0       1
 62   GPTIMER   Modular Timer Unit 1           0         0       1
 63   GPIO      General Purpose I/O port 1     0         0       1

grmon4> grcg enable 7

grmon4> grcg
                     GR716B Primary  GR716B Secondary
Unlock register:     0x00000000      0x00000000
Clock enable register: 0x00010080    0x00000000
Reset register:      0xfffeff7f      0xffffffff

 GATE  CORE(S)   DESCRIPTION                UNLOCKED  ENABLED  RESET
   0   SPI2AHB   SPI to AHB Bridge             0         0       1
   1   I2C2AHB   I2C to AHB Bridge             0         0       1
   2   PWRX      PacketWire RX Interface       0         0       1
   3   PWTX      PacketWire TX Interface       0         0       1
   4   FTMCTRL   Memory ctrl with EDAC 0       0         0       1
   5   SPIMCTRL  SPI Memory Controller 0       0         0       1
   6   SPIMCTRL  SPI Memory Controller 1       0         0       1
   7   SPICTRL   SPI Controller 0              0         1       0
   8   SPICTRL   SPI Controller 1              0         0       1
   9   I2CMST    I2C-master 0                  0         0       1
  10   I2CMST    I2C-master 1                  0         0       1
  11   I2CSLV    I2C Slave 0                   0         0       1
  12   I2CSLV    I2C Slave 1                   0         0       1
  13   ACOMP     GR716 Analog Comparator       0         0       1
  14   CLKDET    GR716 APWM Clk error det      0         0       1
  16   APBUART   APB UART 0                    0         1       0
  17   APBUART   APB UART 1                    0         0       1
  18   APBUART   APB UART 2                    0         0       1
  19   APBUART   APB UART 3                    0         0       1
  20   APBUART   APB UART 4                    0         0       1
```

```
21   APBUART   APB UART 5                  0        0      1
23   APWM      APWM subsystem              0        0      1
24   L3STAT    LEON3 Statistics Unit       0        0      1
25   AHBUART   AHB Debug UART              0        0      1
26   MEMPROT   AMBA Protection unit        0        0      1
28   SPWTDP    CCSDS TDP                   0        0      1
29   SPI4S     SPI for Space Slave         0        0      1
30   NVRAM     FT NVRAM Memory Interface   0        0      1
31   GPIO      General Purpose I/O port 0  0        0      1
32   GRDMAC    DMA Controller 0            0        0      1
33   GRDMAC    DMA Controller 1            0        0      1
34   APWM      Analog applications PWM     0        0      1
35   GRETH     GR Ethernet MAC             0        0      1
36   GR1553B   MIL-STD-1553B Interface     0        0      1
37   GRCANFD   CAN-FD Controller with DMA  0        0      1
38   LVDSIO    LVDS IO                     0        0      1
39   SPWRTR    GRSPW Router                0        0      1
40   DAC       Digital-to-Analog 0         0        0      1
41   DAC       Digital-to-Analog 1         0        0      1
42   DAC       Digital-to-Analog 2         0        0      1
43   DAC       Digital-to-Analog 3         0        0      1
44   ADC       Analog-to-Digital 0         0        0      1
45   ADC       Analog-to-Digital 1         0        0      1
46   ADC       Analog-to-Digital 2         0        0      1
47   ADC       Analog-to-Digital 3         0        0      1
54   GRSCRUB   GRSCRUB FPGA Scrubber       0        0      1
55   RTA       Real-Time Accelerator 0     0        0      1
56   RTA       Real-Time Accelerator 1     0        0      1
57   APWMDAC   GR716 APWM DAC 0            0        0      1
58   APWMDAC   GR716 APWM DAC 1            0        0      1
59   APWMDAC   GR716 APWM DAC 2            0        0      1
60   APWMDAC   GR716 APWM DAC 3            0        0      1
61   FIR       GR716 APWM FIR1             0        0      1
62   GPTIMER   Modular Timer Unit 1        0        0      1
63   GPIO      General Purpose I/O port 1  0        0      1
```

5.5. GR716B License options

The GRMON4 evaluation version which is freely available can be used to operate the GR716B-MINI board. The evaluation version does not support any other GR716B boards. The evaluation version is limited in certain regards compared with the GRMON4 professional product. GRMON4 can be downloaded from the GRMON homepage [RD-4].

The following table summarizes the GRMON4 license options for GR716B.

Table 5.1. GRMON4 license options for GR716B. All references to GRMON4 is for version 3.0.16 or later.

Program version	License	Supported hardware
GRMON4 professional	Professional	• All GR716B systems • All LEON/GRLIB systems
GRMON4 professional	GR716	• All GR716B systems
GRMON4 evaluation	Evaluation • No cost • No registration	• GR716B-MIDI

5.6. GR716B specific considerations

Information for users who are familiar with GRMON but new to GR716B.

- All GR716B APBUART are clock-gated on power-on. GRMON4 command line option `-u` will not clock-enable the APBUART. To clock enable `uart0`, use:

  ```
  grmon4> grcg enable 16
  ```

- The chip has two clock gating units: `grcg0` and `grcg1`. The command **grcg** takes the unit name as last parameter:

  ```
  grmon4> grcg grcg0
  grmon4> grcg grcg1
  ```

- Make sure applications are linked to the on-chip data/instruction SRAM which. Linking and executing in external SRAM is possible but slow.

- The on-chip boot ROM can be engaged from GRMON4 with **go 0**.
- When moving software between boards, note that different APBUART and pins are connected to the FTDI UART channels.

6. TSIM LEON simulator

6.1. Overview

TSIM is a simulator that can emulate a single- or multi-processor LEON computer system. It can be extended to emulate custom I/O functions through loadable modules. TSIM has GDB support which makes C/C++ level debugging possible by connecting GDB to the TSIM's GDB socket. With TSIM one can for example:

- Inspect LEON and simulated peripheral registers
- Load applications with the **load** command.
- Control execution flow by starting applications (**run**), continue execution (**cont**), single-stepping (**step**), inserting breakpoints/watchpoints (**bp**) etc.
- Inspect the current CPU state listing the back-trace, instruction trace and disassemble machine code.

The following section outlines how to use TSIM to emulate the GR716B BOARD Development Board.

TSIM is described on the homepage [https://www.gaisler.com/index.php/products/simulators] and in detail in [RD-5].

6.2. Startup

To start TSIM, use the command:

```
tsim-leon3 -gr716b
```

To emulate custom I/O functions it is possible to use loadable modules. To load a module start TSIM with the `-mod` option.

```
tsim-leon3 -gr716b -mod module.so
```

See [RD-5] for further information about loadable modules.

Other start options can be passed to TSIM for additional chip configuration, for available options see [RD-5].

After TSIM has been started all simulated peripherals can be listed by using the **info sys** command. Detailed descriptions of each cores registers can be listed with the **info reg** command. Using the **info reg [device name]** command will display only the specific core register information. Some cores also have a dedicated status command to display additional information. All TSIM commands can be listed with the **help** command.

```
tsim> info sys
  Address      Description                          Core name
  ----------   --------------------                 -----
  0x80000000   Memory controller with EDAC          mctrl0
  0x80002000   Multi-processor Interrupt Ctrl       irqmp0
  0x80003000   Modular Timer Unit0                  gptimer0
  0x80004000   Modular Timer Unit1                  gptimer1
  0x80100000   GRSPW2 Spacewire Link 0             grspw0
  0x80102000   CAN Controller with DMA 0           grcan0
  0x80103000   CAN Controller with DMA 1           grcan1
  0x80300000   Generic APB UART 0                  uart0
  0x80301000   Generic APB UART 1                  uart1
  0x80302000   Generic APB UART 2                  uart2
  0x80303000   Generic APB UART 3                  uart3
  0x80304000   Generic APB UART 4                  uart4
  0x80305000   Generic APB UART 5                  uart5
  0x80306000   AHB status register 0              ahbstatus0
  0x80309000   SPI Controller 0                   spi0
  0x8030a000   SPI Controller 1                   spi1
  0x8030c000   General purpose I/O port 0         gpio0
  0x8030d000   General purpose I/O port 1         gpio1
  0x80408000   GR716 Digital-to-Analog Converter 0   dac0
  0x80409000   GR716 Digital-to-Analog Converter 1   dac1
  0x8040a000   GR716 Digital-to-Analog Converter 2   dac2
  0x8040b000   GR716 Digital-to-Analog Converter 3   dac3
  0xfff00100   SPI Memory Controller 0             spim0
  0xfff00200   SPI Memory Controller 1             spim1

  To print core registers use the command 'info reg'
  followed by one or more core names.

  Additional info about some cores can be printed with 'coreX_status'.

  The value of the bootstrap register can be listed with the 'bootstrap_status' command.
```

```
  Some registers are implemented as dummies, they can be listed with
  the command 'print_dummies'

  Register     Register description              Value
  --------     --------------------             -----
  ASR16        LEONFT register file prot. reg    0x00000000
  ASR17        Processor config register         0x00000d1e
  ASR20        Alternative window register       0x001e0000
  ASR22        Up counter MSB                    0x80000000
  ASR23        Up counter LSB                    0x00000000

tsim> info reg uart0
  Generic APB UART 0
      0x80300000  UART Data register                    0x00000000
      0x80300004  UART Status register                  0x00000086
      0x80300008  UART Control register                 0x80000000
      0x8030000c  UART Scaler register                  0x00000000

tsim> help

  Command summary:
   ahb len                   Set amba bus trace buffer length
   ahb                       Show amba bus trace history
   batch                     Execute a batch file of TSIM commands
   bload                     Load a binary file
   bp                        Print or set breakpoints for a subset of the available CPUs.
   bp delete                 Deletes breakpoint/watchpoint num.
   bp watch                  Adds a watchpoint at a given address
   boot                      Cold boot, i.e. restart, reset and start execution
   bopt                      Enable idle-loop optimisation.
   bt                        Print backtrace for the current or specified CPUs.
   cont                      Continue execution
   coverage enable           Enables coverage
   coverage disable          Disables coverage
   coverage save             Saves coverage data to file.
   coverage print            Print coverage data
   coverage clear            Resets coverage data
   cp                        List coprocessor info.
   cpu                       List CPUs, or switch active CPU
   debug                     Set debug level
   dbgon                     Toggle debug a debug flag for all cores
   dcache print              Show contents of data cache
   dcache flush              Flush dcache, optionally for given address or symbol only
   dcache query              Print dcache status for given address or symbol
   disassemble               Disassemble instructions at a given address or program counter
   dump                      Dumps memory to file.
   ep clear                  Clear entry point for execution on all or given CPUs.
   ep                        Show or set entry point for all or given CPUs.
   event                     Show event queue
   exit                      Exit the simulator
   float                     Print the FPU registers of the current or given CPUs.
   gdb                       Start GDB server, listening for GDB connection
   gdb reset                 Prepares TSIM for a new run via GDB
   gdb postload              Performs final preparations after loading an image
   go                        Set PC on current CPU and continue simulation
   help                      Show help overview or help for a command or topic
   hist                      Show combined inst/ahb trace history
   icache print              Show contents of instruction cache
   icache flush              Flush icache, optionally for given address or symbol only
   icache query              Print icache status for given address or symbol
   info reg                  Show all system registers or specific cores and/or registers
   inst len                  Set instruction trace buffer length
   inst                      Show instruction trace history
   leon                      Display LEON peripherals registers
   load                      Load a file into simulator memory
   mcfg1                     Set or show user defined memory controller settings
   mcfg2                     Set or show user defined memory controller settings
   mcfg3                     Set or show user defined memory controller settings
   mem                       Display memory at a given address
   vmem                      Display virtual memory at a given address
   mmu                       Display the MMU registers for the current or given CPUs.
   mmu debug                 Set debug level for the MMU on current or given CPU.
   mmu ctrl                  Display or set the value of the MMU control register.
   mmu ctx                   Display or set the value of the MMU context register.
   mmu ctxptr                Display or set the value of the MMU context pointer register.
   mmu tlb                   Display the TLB for the current or given CPUs.
   nolog                     Suppress log output of a command.
   perf                      Show performance statistics
   perf reset                Reset performance statistics
   profile enable            Enable profiling
   profile disable           Disable profiling
   profile                   Show profiling information
   quit                      Exit the simulator
   reg                       Show/set CPU registers (or windows)
   reset                     Restart simulation
```

```
 restore                   Temporarily disabled: Restore simulator state from file
 run                       Restart, reset, initialize and start execution
 save                      Temporarily disabled: Save simulator state to file
 shell                     Execute shell command
 silent                    Suppress stdout of a command.
 stack                     Show, set or clear initial stack pointer on current or given CPU
 step                      Single step on the current CPU.
 symbols                   Load symbols from file
 symbols list              Show symbols.
 symbols lookup            Lookup a symbol
 thread                    Print thread info or backtrace
 version                   Print the TSIM version and build date
 vwmem                     Write word(s) to a virtual address (and onwards)
 walk                      Print a MMU table walk for the current of given CPUs
 wmem                      Write word(s) to a address (and onwards)
 wmemh                     Write half-word(s) to a address (and onwards)
 wmemb                     Write byte(s) to a address (and onwards)
 xwmem                     Write a word to a address in a given address space

IP core commands:
 bootstrap_status          Print bootstrap registers
 grspw0_status             Print GRSPW2 register status.
 grspw0_dbg                Toggle individual debug flags
 grspw0_connect            Connect GRSPW2 core to packet server at a given ip
 grspw0_server             Start packet server for GRSPW2 core on a given port
 grcan0_dbg                Toggle individual debug flags
 grcan1_dbg                Toggle individual debug flags
 spi0_dbg                  Toggle individual debug flags
 spi1_dbg                  Toggle individual debug flags
 gpio0_status              Print gpio ctrl status information
 gpio0_dbg                 Toggle individual debug flags
 gpio1_status              Print gpio ctrl status information
 gpio1_dbg                 Toggle individual debug flags
 print_dummies             Prints a list of all dummy registers
 canbus0_status            Print canbus status information
 canbus1_status            Print canbus status information

User commands:


Type Ctrl-C to interrupt execution

See manual for details and additional command arguments.
For native TCL commands use "help tcl"
```

7. Toolchains

7.1. Bare C Cross-Compiler System

7.1.1. Overview

The Bare C Cross-Compiler (BCC for short) is a GNU-based cross-compilation system for LEON processors. It allows cross-compilation of C and C++ applications for LEON2, LEON3 and LEON4. This section gives the reader a brief introduction on how to use BCC together with the GR716B BOARD Development Board. It will be demonstrated how to build an an example program and run it on the GR716B-BOARD using GRMON4.

The BCC toolchain includes the GNU C/C++ cross-compiler, GNU Binutils, Newlib embedded C library, the Bare-C run-time system with LEON support and the GNU debugger (GDB). The toolchain can be downloaded from [RD-10] and is available for both Linux and Windows. Further information about BCC can be found in [RD-11].

The installation process of BCC is described in [RD-11]. The rest of this chapter assumes that **sparc-gaisler-elf-gcc** is available in the PATH variable.

7.1.2. Compiling with BCC

The following command shows an example of how to compile a typical *hello, world* program with BCC.

```
$ cat hello.c
#include <stdio.h>

int main(void)
{
        printf("hello, world\n");
        return 0;
}

$ sparc-gaisler-elf-gcc -qbsp=gr716b -mcpu=leon3 -O2 -g hello.c -o hello.elf
```

All GCC options are described in the gcc manual. Some of the most common options are:

Table 7.1. BCC's GCC compiler relevant options

`-g`	generate debugging information - recommended for debugging with GDB
`-msoft-float`	emulate floating-point - must be used if no FPU exists in the system
`-O2`	optimize for speed
`-Os`	optimize for size
`-Og`	optimize for debugging experience
`-qsvt`	use the single-vector trap model
`-mflat`	enable flat register window model. The compiler will not emit SAVE and RESTORE instructions.

It is recommended to use the options

```
-qbsp=gr716b -mcpu=leon3
```

with GR716B. For more details, see [RD-10].

To compile binaries intended to run on the ;soc RTA it is recommended to use the following options for RTA0

```
-qbsp=gr716b -mcpu=leon3 -mflat -qsvt -T linkcmds-rta0
```

and for RTA1

```
-qbsp=gr716b -mcpu=leon3 -mflat -qsvt -T linkcmds-rta1
```

Note that the option

```
-T linkcmds-rtaX
```

is needed to link the program to the RTA, where X is the target RTA index.

7.1.3. Running and debugging with GRMON4

Once your application is compiled, connect to your GR716B-BOARD with GRMON4. The following log shows how to load and run an application. Note that the console output is redirected to GRMON4 by the use of the `-u` command line switch, so that the application standard output is forwarded to the GRMON4 console.

```
$ grmon -uart /dev/ttyUSB0 -u -cginit 0x00010000
  GRMON LEON debug monitor v4.0.0 64-bit pro version

  Copyright (C) 2019 Frontgrade Gaisler - All rights reserved.
  For latest updates, go to http://www.gaisler.com/
  Comments or bug-reports to support@gaisler.com

grmon4> load hello.elf
  31000000 .text                    23.6kB /  23.6kB   [===============>] 100%
  30005E70 .data                     2.7kB /   2.7kB   [===============>] 100%
  Total size: 26.29kB (803.58kbit/s)
  Entry point 0x31000000
  Image hello.elf loaded

grmon4> run
hello, world

  CPU 0:  Program exited normally.
```

To debug the compiled program you can insert breakpoints, step and continue execution directly from the GRMON4 console. Compilation symbols are loaded automatically by GRMON4 once you load the application. An example is provided below.

```
grmon4> load hello.elf
  40000000 .text                    23.6kB /  23.6kB   [===============>] 100%
  40005E70 .data                     2.7kB /   2.7kB   [===============>] 100%
  Total size: 26.29kB (806.59kbit/s)
  Entry point 0x40000000
  Image hello.elf loaded

grmon4> bp main
  Software breakpoint 1 at <main>

grmon4> run

  CPU 0:  breakpoint 1 hit
          0x40001928: b0102000  mov  0, %i0  <main+4>
  CPU 1:  Power down mode

grmon4> step
  0x40001928: b0102000  mov  0, %i0  <main+4>

grmon4> step
  0x4000192c: 11100017  sethi  %hi(0x40005C00), %o0  <main+8>

grmon4> cont
hello, world

  CPU 0:  Program exited normally.
```

Alternatively you can run GRMON4 with the `-gdb` command line option and then attach a GDB session to it. For further information see Chapter 3 of [RD-11].

7.1.4. Running and debugging with TSIM

Once your application is compiled, start TSIM with the `-gr716b` option. The following log shows how to load and run an application.

```
$ tsim-leon3 -gr716
 TSIM/LEON3 SPARC simulator, version [...]

 Copyright (C) 2019, Frontgrade Gaisler - all rights reserved.
 For latest updates, go to http://www.gaisler.com/
 Comments or bug-reports to support@gaisler.com
[...]
tsim> load hello.elf
  section: .text, addr: 0x31000000, size 22400 bytes
  section: .rodata, addr: 0x30000000, size 128 bytes
  section: .data, addr: 0x30000080, size 1216 bytes
  read 345 symbols
tsim> run
  starting at 0x31000000
```

```
hello, world


  Program exited normally.
```

To debug the compiled program you can insert breakpoints, step and continue execution directly from the TSIM console. Compilation symbols are loaded automatically by TSIM once you load the application. An example is provided below.

```
tsim> load hello.elf
  section: .text, addr: 0x31000000, size 22400 bytes
  section: .rodata, addr: 0x30000000, size 128 bytes
  section: .data, addr: 0x30000080, size 1216 bytes
  read 345 symbols
tsim> bp main
  breakpoint 1 at 0x3100124c:  main + 0x4
tsim> run
  starting at 0x31000000

  breakpoint 1  main + 0x4
tsim> step
   858  3100124c  b0102000  mov       0, %i0                  [00000000]
tsim> step
   859  31001250  110c0000  sethi     %hi(0x30000000), %o0    [30000000]
tsim> cont
hello, world


  Program exited normally.
```

Alternatively you can run TSIM with the `-gdb` command line option and then attach a GDB session to it. For further information see Chapter 3 of [RD-11].

8. Software examples

This chapter will describe a software example collection applicable for GR716B and GR716B-BOARD applications. The approach is to provide a starting point for integrating different IO functionality with an application.

8.1. Overview

8.1.1. BCC device driver library

BCC ([RD-10]) includes a device driver library which can be used on different LEON systems. The device driver library comes with its own example collection which is also applicable to GR716B.

Functionality covered by the BCC driver examples are mainly communication interfaces.

Available in the BCC installation in the directory `src/libdrv/examples/`.

8.1.2. Custom GR716B interface examples

A set of examples with GR716B specific functionality is distributed together with this document. It contains programs which which can be used on different LEON systems. The device driver library comes with its own example collection which is also applicable to GR716B

Functions covered by the custom examples include ADC, DAC, PLL, programmable IO and more.

These files are distributed in the archive named `gr716-examples-<DATE>.zip`.

8.2. Environment

All examples are designed to be loaded and run using GRMON4. Any board preparation is described in connection to each example.

Some of the examples demonstrate how to use the on-chip boot loader. Those examples use grmon for loading while running is done automatically at power-on.

8.3. IO switch matrix configuration

GR716B external pins can be reprogrammed for different functionality. A complete list of IO to pin routing options and values is available in [RD-2], section named *I/O switch matrix overview*.

Pin function is programmed by setting fields in a set of chip registers.

GPIO	GR716B Data sheet and User's Manual	GRMON4 variable
0..7	`SYS.CFG.GP0`	`gpreg1::gpio0::gpio<N>`
8..15	`SYS.CFG.GP1`	`gpreg1::gpio1::gpio<N>`
16..23	`SYS.CFG.GP2`	`gpreg1::gpio2::gpio<N>`
24..31	`SYS.CFG.GP3`	`gpreg1::gpio3::gpio<N>`
32..39	`SYS.CFG.GP4`	`gpreg1::gpio4::gpio<N>`
40..47	`SYS.CFG.GP5`	`gpreg1::gpio5::gpio<N>`
48..55	`SYS.CFG.GP6`	`gpreg1::gpio6::gpio<N>`
56..63	`SYS.CFG.GP7`	`gpreg1::gpio7::gpio<N>`

All IO are configured for GPIO operation (value 0) at power-on. The on-chip boot loader reconfigures some functions depending on the boot interface selected by the bootstrap.

8.3.1. Configuring with GRMON4

To list the current IO configuration using GRMON4, the **info reg gpreg1** command can be used:

Example 8.1.

```
grmon4> info reg -v gpreg1
  General Purpose Register Bank
      0x8000d000  System IO register for GPIO 0 to 7      0x00000000
      31:28  gpio7             0x0         Functional select for GPIO 7
      27:24  gpio6             0x0         Functional select for GPIO 6
      23:20  gpio5             0x0         Functional select for GPIO 5
      19:16  gpio4             0x0         Functional select for GPIO 4
      15:12  gpio3             0x0         Functional select for GPIO 3
      11:8   gpio2             0x0         Functional select for GPIO 2
       7:4   gpio1             0x0         Functional select for GPIO 1
       3:0   gpio0             0x0         Functional select for GPIO 0

      0x8000d004  System IO register for GPIO 8 to 15     0x00000000
      31:28  gpio15            0x0         Functional select for GPIO 15
      27:24  gpio14            0x0         Functional select for GPIO 14
      23:20  gpio13            0x0         Functional select for GPIO 13
      19:16  gpio12            0x0         Functional select for GPIO 12
      15:12  gpio11            0x0         Functional select for GPIO 11
      11:8   gpio10            0x0         Functional select for GPIO 10
       7:4   gpio9             0x0         Functional select for GPIO 9
       3:0   gpio8             0x0         Functional select for GPIO 8
[...]
```

Since the registers are mapped to Tcl variables, they can be used in the interactive terminal or scripts. The following example sets and prints the numeric value of the GPIO39 IO configuration.

Example 8.2.

```
grmon4> set gpreg1::gpio4::gpio39 1
grmon4> puts [set gpreg1::gpio4::gpio39]
  1
```

8.3.2. Configuring with the CPU

A C function is provided with the GR716B example collection for the purpose of configuring GR716B IO. It is named `set_pinfunc()` and is located in `common/include/pinhelper.h` and `common/pinhelper.c`.

```
/*
 * configure one IO switch matrix entry
 *
 * This function updates one field in SYS.CFG.GPx to configure the
 * specified pin with the functionality requested by mode.
 *
 * Parameters pin and mode are range checked before registers are written.
 *
 * pin:           GPIO pin number (0..63)
 * mode:          Any of IO_MODE_* (0..0xe)
 * return:        0 on success, else non-zero.
 */
int set_pinfunc(
        unsigned int pin,
        unsigned int mode
);
```

In addition to the function prototype, a set of preprocessor symbols (constants) are defined for the different functions in `pinhelper.h`. For example:

Example 8.3.

```
#define IO_MODE_GPIO      0x0
#define IO_MODE_APBUART   0x1
#define IO_MODE_MEM       0x2
```

The below example demonstrates how the function `set_pinfunc()` can be used to configure a pin for UART operation. It has the same effect as Example 8.2.

Example 8.4.

```
#include <stdio.h>
#include <pinhelper.h>

int main(void)
{
        int ret;
        printf("Configuring GPIO39 for UART operation... ");
        ret = set_pinfunc(39, IO_MODE_APBUART);
        if (ret) {
                puts("FAIL");
```

```
                return ret;
        }
        puts("OK");
        return 0;
}
```

8.4. BCC device drivers

All BCC device drivers are documented in [RD-11]. The example source code and driver source code is distributed with the BCC binary distribution ([RD-10]).

In the following subsections, the symbol `$BCC` is used represent the BCC installation path. For example

```
BCC=/opt/bcc-2.0.7-gcc
```

8.4.1. GPIO

For an example on how to operate the GPIO, see the section named *GPIO driver* in the BCC User's Manual.

An option to using the device driver is to access the GPIO registers directly from the application. Register descriptions are available in the C header file `drv/regs/grgpio.h`. It can be used like this:

```
#include <stdio.h>
#include <drv/regs/grgpio.h>

int main(void) {
  volatile struct grgio_regs *gpio0 = (void *) 0x8030C000;
  printf("gpio[0] data is 0x%08x\n", (unsigned int) gpio0->data);
  return 0;
}
```

8.4.1.1. Preparation

Enable the GPIO function for the GPIO signals to be used. See Section 8.3.

8.4.2. Clock gating unit

The BCC driver provides functions for enabling and disabling the clock for selected cores.

- `clkgate_enable()`
- `clkgate_disable()`

The example named `clkgate` prints the status of each clock gating unit. An example on how to enable/disable clock is also in the source code file `clkgate.c`, and can be activated by the user.

8.4.2.1. Preparation

none

8.4.2.2. Build

```
cd $BCC/src/libdrv/examples/clkgate
make bsp=gr716b
```

8.4.2.3. Run

```
grmon4> load bin/clkgate.elf
grmon4> run

EXAMPLE BEGIN
Init with GR716 static tables
clkgate_dev_count() -> 2
 clkgate0: addr=80006000, interrupt= 0, device_id=0x02c, version=1
 clkgate1: addr=80007000, interrupt= 0, device_id=0x02c, version=1

clkgate0 configuration:
  enabled=08810060
  disabled=777eff9f

clkgate1 configuration:
  enabled=00000000
```

```
  disabled=003fffff

EXAMPLE END

  Program exited normally.

grmon4>
```

8.4.3. Memory protection unit

The example named `memprot` demonstrates how the memory protection unit can be programmed to generate bus error on write in a specific address range. For the purpose of this example, the CPU performs a write attempt which trigs a CPU trap because of the bus error response.

8.4.3.1. Preparation

The MEMPROT unit needs to be clock enabled. See the GRMON4 transcript below.

8.4.3.2. Build

```
  cd $BCC/src/libdrv/examples/memprot
  make bsp=gr716b
```

8.4.3.3. Run

```
grmon4> grcg enable 26 grcg0
grmon4> load bin/memprot.elf
grmon4> run

GR716 memory protection unit example.
NOTE: This example is functional only on the GR716.

INFO: memprot_dev_count() -> 2

memprot0: addr=0x80005000, interrupt=63
memprot1: addr=0x8010a000, interrupt=63

example0: open memprot0...
example0: memprot0 has 4 segments

example0: device configuration at open():
SEGMENT 0
  start = 00000000
  end   = 00000000
  g     = 00000000
  en    = 0

SEGMENT 1
  start = 00000000
  end   = 00000000
  g     = 00000000
  en    = 0

SEGMENT 2
  start = 00000000
  end   = 00000000
  g     = 00000000
  en    = 0

SEGMENT 3
  start = 00000000
  end   = 00000000
  g     = 00000000
  en    = 0

example0: reset()...
printstarted: device is STOPPED

example0: install example configuration on segment 0...
example0: reading back segment 0...
  start = 80004000
  end   = 800040ff
  g     = 00000004
  en    = 1

example0: trying to write 0x80004000...
example0: PASS - expected since core is disabled

example0: starting (enabling) memprot0...
printstarted: device is STARTED
```

```
example0: writes to 0x80004000 is now expected to trap with tt = 0x2B.
example0:   HINT: Use the GRMON 'reset' command to disable memory protection
example0: trying to write 0x80004000...

  IU exception (tt = 0x2B, data store error)
  0x310004e8: 90122078  or  %o0, 0x78, %o0  <example0+468>

grmon4>
```

8.4.4. Memory scrubber

The example named `memscrub` sets up the hardware scrubber device to repeatingly scrub a memory area. If multiple correctable errors occur during the same scrub iteration, the scrubber is switched into regeneration mode. When regeneration has completed, the ordinary scrubbing is resumed.

For information on how to configure the example, see the top of the source code file `$BCC/src/libdrv/examples/memscrub/memscrub.c`.

Memory scrubbing policy is highly application and mission dependent. The purpose of this example is to demonstrate how the driver can be used and how to trig uncorrectable errors, and shall not be regarded as representative strategy on how to perform memory scrubbing.

8.4.4.1. Forcing errors

The bus which memscrub0 probes is configurable in GR716B. It is controlled by the *"Force Scrubber (FS)"* bit in *"`SYS.CFG.SCFG` - Interrupt test configuration register"*.

If this bit is 0, then all scrub accesses will get bus error response. It is possible to set and unset this bit when running the example to collect uncorrectable erros.

Clearing and setting FS can be done in GRMON4:

```
grmon4> set gpreg2::scfg::fs 0

grmon4> set gpreg2::scfg::fs 1
```

8.4.4.2. Preparation

None

8.4.4.3. Build

```
cd $BCC/src/libdrv/examples/memscrub
make bsp=gr716b
```

8.4.4.4. Run

In the transcript below, `Interrupted!` means that execution was stopped by typing **Ctrl+C** in the the GRMON4 terminal. After the system state has been changed by the user, the GRMON4 command **cont** continues execution.

```
grmon4> set gpreg2::scfg::fs 1
grmon4> info reg -v gpreg2::scfg
  General Purpose Register
      0x8000e000  Interrupt test configuration register   0x00000800
      20:18  vref              0x0         Enable external voltage reference
      17:16  spw               0x0         SpaceWire loopback production test
      15     ll                0x0         LVDS External Loop
      14:13  ls                0x0         LVDS production test enable
      12     le                0x0         Support Locked transfers in SCRUBBER
      11     fs                0x1         Force Scrubber on main bus
      10:9   prot              0x0         Interrupt test protection bits
       8:3   irq               0x0         Generate interrupt
       2     mr                0x0         Disable reset signal from MIL core
       1     we                0x0         Disable reset request
       0     ee                0x0         Disable reset generation

grmon4> load bin/memscrub.elf
grmon4> run

Memory scrubber example.
```

```
INFO: memscrub_dev_count() -> 1
  memscrub0: addr=0xfff00000, interrupt=63
example0: open memscrub0 -> OK

-- Scrubber BCC 2.0.6 test application --
Config: Mem range: 30000000-3000ffff
Opermode:quiet


  Interrupted!
  0x31000980: 01000000  nop  <memscrub_msgq_pop+72>

grmon4> set gpreg2::scfg::fs 0

grmon4> cont

  Interrupted!
  0x31001b94: 02800004  be  0x31001ba4  <__bcc_int_get_source+12>

grmon4> set gpreg2::scfg::fs 1

grmon4> cont
[R] UE detected, addr=30005eb0, rd, mst=2, size=4
[R] UE detected, addr=3000080c, rd, mst=2, size=4
[R] UE detected, addr=30000928, rd, mst=2, size=4
[R] UE detected, addr=30000a48, rd, mst=2, size=4
[R] UE detected, addr=30000b64, rd, mst=2, size=4


  Interrupted!
  0x31000980: 01000000  nop  <memscrub_msgq_pop+72>

grmon4>
```

The number of correctable and uncorrectable errors detected by the example are available in the variables `ce_count` and `ue_count`:

```
grmon4> mem ue_count 4
  0x30000b34  0000ea4d                                ...M

grmon4> mem ce_count 4
  0x30000b38  00000000
```

8.4.5. AHB status register

The example named `ahbstat-isr` demonstrates how AHBSTAT interrupts can be used. For the purpose of the example, the ISR counts the total number of errors encountered and properties of the last error.

8.4.5.1. Forcing errors

Correctable and uncorrectable errors can be simulated by writing directly to the AHB status register. For example:

```
set ::ahbstat0::status::ne 1
```

8.4.5.2. Preparation

None

8.4.5.3. Build

```
  cd $BCC/src/libdrv/examples/ahbstat
  make bsp=gr716b
```

8.4.5.4. Run

Note that the `memscrub` device is backwards compatible with the `ahbstat` device. Thus the AHB status register driver can be used with the `memscrub` device, with limited functionality.

In the transcript below, the device named `ahbstat2` is the same as `memscrub0`.

```
grmon4> load bin/ahbstat-isr.elf
grmon4> run

EXAMPLE BEGIN
Init with GR716 static tables
ahbstat_dev_count() -> 3
 ahbstat0: addr=8000a000, interrupt=63, device_id=0x052, version=0
```

```
 ahbstat1: addr=80306000, interrupt=63, device_id=0x052, version=0
 ahbstat2: addr=fff00000, interrupt=63, device_id=0x057, version=0
Trig interrupt...
Interrupt condition cleared
User ISR has been called 1 times
 count=1
 last_status=00000302
 last_failing_address=80300010
EXAMPLE END

  Program exited normally.

grmon4>
```

8.4.6. APBUART

This section describes how to communication over UART using the `apbuart` controller.

8.4.6.1. BCC and UART

BCC by default uses `apbuart0` for its console. In particular the console is associated with the files `stdin`, `stdout` and `stderr` which is used by many of the C standard library `<stdio.h>` functions. The BCC console driver uses hardware FIFO when available but never uses interrupts. Information on how to redirect the BCC console driver is available in the BCC User's Manual.

8.4.6.2. The APBUART device driver

The APBUART device driver included in the BCC peripheral driver library is independent of the BCC console UART functionality. Using this device driver allows for operating the `apbuart` with full control and performance. Interrupt mode is available.

It is recommended not to use the BCC peripheral driver library driver on the same `apbuart` device as the BCC console driver.

8.4.6.3. Preparation

The following prepares to run the example with `apbuart3` with UART TX output on GPIO35 and UART RX input on GPIO36.

1. Connect a terminal to `apbuart3` and set 115200 BAUD. When using GR716B-BOARD, and the serial communication program *Minicom*, the following can be used:

   ```
   $ minicom -D /dev/ttyUSB2 -b 115200
   ```
2. Clock enable `apbuart3`.
3. Configure the appropriate GR716B pins for UART functionality. Set `SYS.CFG.GP4.GPIO35 = 0x1` (UART_TX3) and set `SYS.CFG.GP4.GPIO36 = 0x1` (UART_RX3),

The below commands performs step 2 and 3.

```
grmon4> # clock enable UART3
grmon4> grcg enable 19
grmon4> # connect UART3 signals to pins
grmon4> set gpreg1::gpio4::gpio35 1
grmon4> set gpreg1::gpio4::gpio36 1
```

8.4.6.4. Build

Building the example for operating on `apbuart3` with 115200 BAUD is done with:

```
  cd $BCC/src/libdrv/examples/uart
  make bsp=gr716b CFLAGS="-DCFG_UART_INDEX=3 -DCFG_UART_BAUD=115200"
```

8.4.6.5. Run

```
grmon4> # clock enable UART0 for the BCC console output
grmon4> grcg enable 16
grmon4> load bin/uart.elf
grmon4> run

INFO: apbuart_dev_count() -> 6
apbuart0: addr=0x80300000, interrupt=24
apbuart1: addr=0x80301000, interrupt=25
apbuart2: addr=0x80302000, interrupt=42
```

```
apbuart3: addr=0x80303000, interrupt=44
apbuart4: addr=0x80304000, interrupt=45
apbuart5: addr=0x80305000, interrupt=46
INFO: end

  Program exited normally.

grmon4>
```

The following is displayed on the external terminal:

```
hello world
HELLO WORLD using interrupt based apbuart_write()
```

8.4.7. SPI master controller

GR716B has two *SPI controllers* (`SPICTRL`) typically named `spi0` and `spi1`. GR716B also has two *SPI memory controllers* (`SPIMCTRL`) typically named `spim0` and `spim1`. These four controllers operate independently.

The example in this section describes how to use the SPI controller device driver distributed with BCC. The driver is general enough to work with most SPI peripherals such as temperature sensors, ADC, DAC, etc.

8.4.7.1. Preparation

1. An internal loopback connection is setup by the example, effectively connecting `MISO` with `MOSI`. Thus, configuration of external IO is not needed.
2. Clock enable `spi0` with the GRMON4 command **grcg enable 7**.

8.4.7.1.1. External SPI bus

To disable the internal loopback and instead use an external SPI bus, the following example source code statement shall be removed:

```
REGSA->mode |= SPICTRL_MODE_LOOP;
```

The IO pins should also be configured approrpiately.

8.4.7.2. Build

```
  cd $BCC/src/libdrv/examples/spi
  make bsp=gr716b
```

8.4.7.3. Run

```
grmon4> load bin/spi.elf
grmon4> run

SPI example begin
PASS
SPI example end

  Program exited normally.

grmon4>
```

8.4.8. I2C master controller

GR716B has two *I2C master controllers*, `i2cmst0` and `i2cmst1` which can be connected to individual external I2C buses.

The example in this section describes how to use the I2C master controller device driver distributed with BCC. The driver is general enough to work with most I2C peripherals such as temperature sensors, ADC, DAC, etc.

8.4.8.1. GR716B-BOARD current measurement

GR716B-BOARD has two current measurement devices with I2C interface. The I2C bus of these devices are not directly connected to the GR716B. However, the buses are available on the expansion headers which allows to connect the current measurement by fitting jumpers on the connector.

- I2C clock (`PM_SCL`) is connected to P2 pin 52.

- I2C data (PM_SDA) is connected to P2 pin 51.

Jumpers can be fitted to connect these pins with GR716B I2C IO. See [RD-2] for a list of available pins.

8.4.8.2. Preparation

- Clock enable `i2cmst0` with the GRMON4 commands:

```
grmon4> grcg enable 9
grmon4> grcg enable 10
```

8.4.8.3. Build

```
cd $BCC/src/libdrv/examples/i2cmst
make bsp=gr716b
```

8.4.8.4. Run

In the following transcript, two I2C slaves were available on the bus of `i2cmst1`.

```
grmon4> load bin/scan.elf
grmon4> run

This program tries to detect I2C devices on all available
I2CMST controllers. It does so by performing a read on each
non-reserved I2C address. 7-bit addressing is used.

INFO: i2cmst_dev_count() -> 2

i2cmst0: addr=0x8030e000, interrupt=50
 - This I2C master got no response on any I2C address

i2cmst1: addr=0x8030f000, interrupt=51
 Detected I2C device at I2C address 0x50
 Detected I2C device at I2C address 0x51
 - This I2C master got response on 2 I2C address(es)

  Program exited normally.

grmon4>
```

8.5. GR716B interfaces

The examples used in the following subsections are available in the GR716 example collection.

In the following, the symbol `$EXAMPLE` is used represent the example base directory.

8.6. Boot loader

The GR716B on-chip ROM contains a boot loader which is engaged on power-on. The boot loader can copy a user application image from non-volatile external memory to on-chip RAM and start executing.

8.6.1. The ASW format

The ASW (application software) image format encapsulates an application, and includes:

- Entry point
- Multiple program or data sections with information on
  - Section length
  - Source location in non-volatile ROM
  - Destination address in volatile RAM
  - CRC

For details on the ASW format, see [RD-2].

8.6.2. Scripts

File	Description
`bchfile.tcl`	Calculates BCH check bytes for binary input data. It write both the input data bytes and the BCH bytes to an output file appropriate for loading into SPI or parallel memory.

File	Description
	• Input: Binary data • Output: Binary data or ELF
`img-gr716b.tcl`	Tool for transforming an ELF file into an ASW image. By default a binary file ASW image file is created. If the -a option is present, then an ELF file suitable for loading with GRMON4 and TSIM is also created. • Input: ELF file • Output: Binary data or ELF

8.6.2.1. ASW image tool

A tool is provided in `$EXAMPLE/scripts/img-gr716b.tcl` for generating ASW image files from application compiled and linked to RAM. The generated ASW image files are compatible with the ASW image format described in [RD-2] and can be loaded into non-volatile memory.

Command line parameter syntax for `img-gr716b.tcl` is:

```
img-gr716b.tcl [-a ASM_ADDR] [-o OUTFILE] INFILE...
```

Parameter *`INFILE`* is the file name of an ELF image linked to RAM, representing a user application compiled with for exampe BCC (Bare-C compiler) or RCC (RTEMS compiler). More than one file name can be provided on the command line in which case all ELF files will be included in the resulting ASW image.

The optional *`-o OUTFILE`* parameter specifies the output file name of the ASW image. If this option is not given on the command line, then the default output file name for the ASW image is the first *`INFILE`* with the word `.img` appended.

`-a ASM_ADDR` is also optional and can be used to specify the target address of the ASW image in Application Storage Memory. If this option is used, then an additional output file is created, named `<FILE>.elf`. It contains the same data as the `<FILE>` file, but also has ELF information with load address. This allows for loading the image with the GRMON4 commands **load**, **flash load** or **spim flash load** and the TSIM command **load**.

The image format is position independent and the image can be relocated in the same or in another Application Storage Memory without the need for re-generating. This is independent of the *`-a`* parameter.

8.6.2.2. BCH generation tool

A script named `$EXAMPLE/scripts/bchfile.tcl` is available for generating BCH check bytes to arbitrary input data.

Command line parameter syntax for `bchfile.tcl` is:

```
bchfile.tcl [-elf] [-a ADDRESS] SOURCE DEST SIZE
```

Parameter	Description
`-elf`	Generate an elf output file in addition to the binary output. The ELF file contains target address information for convenient loading with GRMON4 or TSIM.
`-a ADDRESS`	Target address for the `-elf` option. Default is `0`.
`SOURCE`	name of the input binary file
`DEST`	name of the output binary file which will contain the SOURCE binary appended with calculated BCH check bytes.
`SIZE`	target ROM size, in KiB.

8.6.3. ASW image in SPI flash example

This section will describe how to convert an application compiled with BCC into an application image stored in external SPI flash memory for automatic start. The BCC application is linked to on-chip RAM and can also be loaded and executed directly with GRMON4 or TSIM.

An example on ASW loading from SPI flash is available in `$EXAMPLE/aswboot`. The main application is in the file `hello.c`.

File	Description
`hello.c`	Application `main()` function which prints `hello, world`.
`init60.c`	BCC run-time init hook `__bcc_init60()` which configures • UART clock gating • UART pins
`Makefile`	Build script

8.6.3.1. Build

```
cd $EXAMPLE/aswboot
make
```

The make command automatically runs the scripts described above:

```
img-gr716b.tcl fixup.elf hello-board.elf -a 0x02000000 -o hello-board.elf.img
bchfile.tcl -elf -a 0x02000000 hello-board.elf.img hello-board.elf.img.rom-with-bch 16384
```

The following files of interest are generated:

File	Description
`hello-board.elf`	Application linked for execution in internal RAM. This file can be loaded and run with GRMON4 or TSIM.
`hello-board.elf.img`	Raw ASW image for loading into non-volatile memory.
`hello-board.elf.img.rom-with-bch.elf`	Same as above, and also includes BCH checkbytes for use with EDAC enabled memory controller.

The target load sections can be investigated:

Example 8.5.

```
$ sparc-gaisler-elf-objdump -h hello-board.elf.img.rom-with-bch.elf

hello-board.elf.img.rom-with-bch.elf:     file format elf32-sparc

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .data         00004240  02000000  02000000  00000034  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  1 .bch          00001090  02ffef70  02ffef70  00004274  2**0
                  CONTENTS, ALLOC, LOAD, DATA
```

In the above example, a separate ELF section (`.bch`) containing EDAC check bytes is located at the end of the non-volatile memory space. Whenever a read is requested by the CPU for data in the `.data` section, the controller will automatically also read the corresponding check bytes.

8.6.3.2. Load and Run

The same boot image file can be used on hardware and in simulation. The load and run procedure is a bit different.

8.6.3.2.1. GRMON4

When using GRMON4 the image can be loaded with:

```
grmon4> spim flash detect
  Got manufacturer ID 0xc2 and device ID 0x2019
  Detected device: Macronix MX25L25635F

grmon4> spim flash load -erase hello-board.elf.img.rom-with-bch.elf
  02000000 .data                    16.6kB /  16.6kB   [===============>] 100%
  02FFEF70 .bch                      4.1kB /   4.1kB   [===============>] 100%
  Total size: 20.70kB (2.56kbit/s)
  Entry point 0x00000000
```

```
  Image hello-board.elf.img.rom-with-bch.elf loaded

grmon4> verify hello-board.elf.img.rom-with-bch.elf
  02000000 .data                    16.6kB /  16.6kB   [===============>] 100%
  02FFEF70 .bch                      4.1kB /   4.1kB   [===============>] 100%
  Total size: 20.70kB (64.05kbit/s)
  Entry point 0x00000000
  Image of hello-board.elf.img.rom-with-bch.elf verified without errors
```

Make sure the bootstrap signals are set for SPI ASW boot (default) and then power-off and power-on the board. The image can also be started by issuing **go 0** in GRMON4.

8.6.3.2.2. TSIM

When using TSIM start TSIM with the `-bootstrap 0x0000c002` option to set the bootstrap register to boot from SPI ASW. If running the provided example it is recommended to also use the `-u 3` option to forward UART3 output to the TSIM terminal. Once TSIM is started use the **load** command.

```
$ ./tsim-leon3 -gr716b -bootstrap 0x0000c002 -u 3
...
tsim> load hello-board.elf.img.rom-with-bch.elf
  section: .data, addr: 0x2000000, size 16960 bytes
  section: .bch, addr: 0x2ffef70, size 4240 bytes
  read 5 symbols
```

Start the image with the **go 0** command.

8.6.3.3. Output

The program prints a message to UART3 each second. Below is an example output.

```
hello, world at time 1
hello, world at time 2
hello, world at time 3
hello, world at time 4
```

9. Expansion boards

This section provides an overview of expansion boards which can be attached to the GR716B-BOARD expansion connectors.

9.1. DSU UART FTDI

The GR716B-DSU-USB board can be connected to the GR716B-BOARD to:

- Debug UART to USB conversion supported by GRMON4 (Chapter 5).
- Enable/Disable internal Debug unit (DSUEN)
- External Reset button
- External Break button

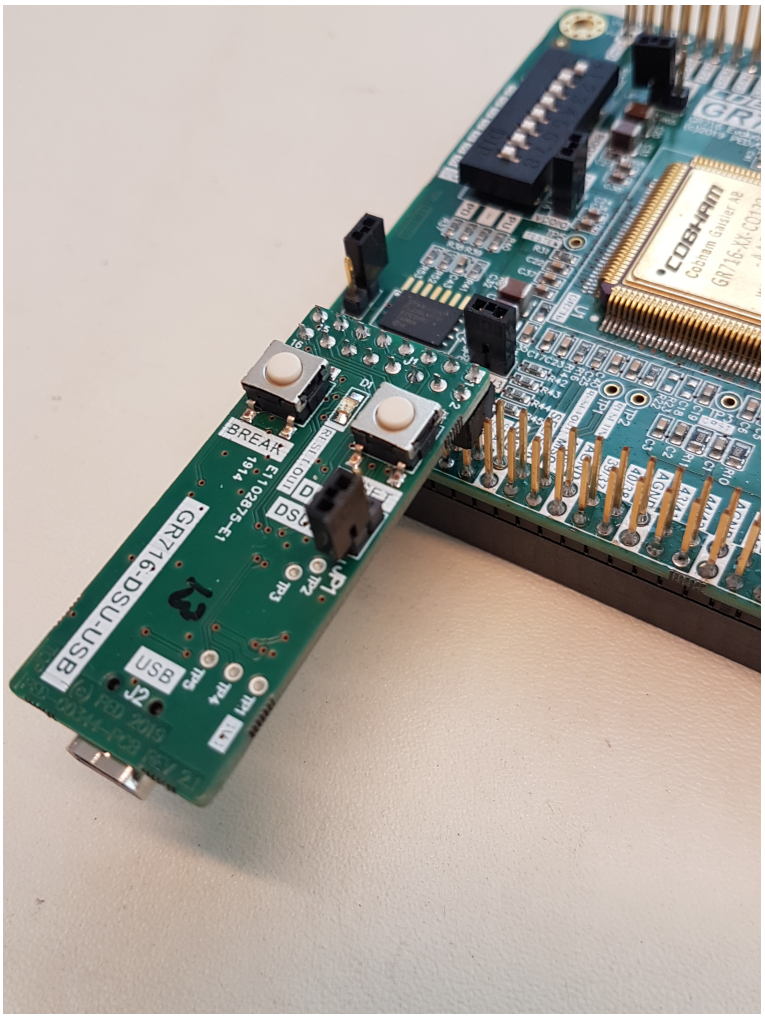


Figure 9.1. GR716B DSU Board for UART to USB conversion and debug control

9.2. Memory board

The GR716B-TEST-MEMORY board when connected to GR716B-BOARD provides:

- Redundant SPI PROM/FLASH
- 4 MiB SRAM with possibility to test different chip select setups
- 16 MiB FLASH with possibility to test different chip select setups
- Possibility to test different boot variants from SRAM/FLASH/SPI memories with and without redundancy.
- Use external System and SpaceWire oscillators

Best practice is to connected the GR716B-TEST-MEMORY board to the GR716B-BOARD via the PC/104 style stackable headers on the backside as shown in Figure 9.2.

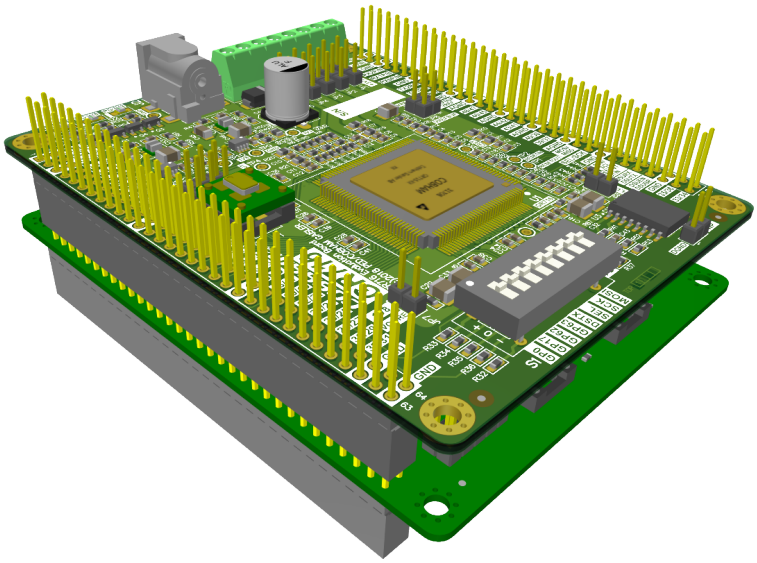


Figure 9.2. GR716B Board with a GR716B Memory board connected via the PC/104 connector on the backside.

9.3. Interface boards

The GR716B-DEV board extends the GR716B-BOARD with the Debug Interface, Control interface and possibility to connect to communication interfaces.

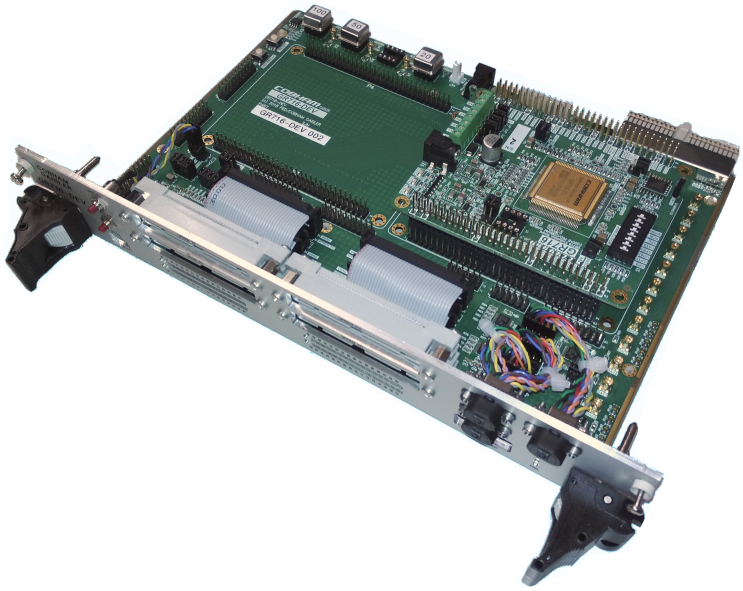


Figure 9.3. GR-CPCI-GR716B-DEV board with a GR716B-BOARD mounted in slot #1

10. Support

For support contact the support team at support@gaisler.com.

When contacting support, please identify yourself in full, including company affiliation and site name and address. Please identify exactly what product that is used, specifying if it is an IP core (with full name of the library distribution archive file), component, software version, compiler version, operating system version, debug tool version, simulator tool version, board version, etc.

The support service is only for paying customers with a support contract.

Appendix A. Assembly drawing

Figure A.1. GR716B-BOARD top

Appendix B. Default configuration

This appendix describes the jumper and switch positions for the default configuration of the board. For more details, see the section named *Setting Up and Using the Board* in the GR716B-BOARD Development Board User's Manual [RD-1].

Table B.1. Jumper configuration

Item	Default	Description of the default configuration
JP1	1-2	Select power supply for ADC
JP2	1-2	Select power supply for REF
JP3	1-2	Select power supply for DAC
JP4	1-2	Select power supply for LVDS
JP5	NC	Not available
JP6	1-2	Select power supply for LDO
JP7	NC	Select power supply for VDDCORE
JP8	NC	Select internal reference (VREFBUF) Always leave JP8 *open* when connecting GR716B-BOARD with the GR716B-DEV board.
JP9	NC	External Reference sense (VREF)
JP10	1-2	Select system clock source (CLK)
JP11	1-2	Select PLL clock source (SPWCLK)
JP12	NC	Write protect SPI PROM
JP13	1-2	Install to provide onboard power to 3V3

Table B.2. Switch configuration

Item	Default	Description of the default configuration
S1-1	PU	Disable EDAC
S1-2	PD	CAN-FD bus Primary or Redundant
S1-3	PD	CAN-FD/I2C node ID[2]
S1-4	PD	Bypass internal Boot PROM
S1-5	PD	CAN-FD remote boot
S1-6	PD	CAN-FD/I2C node ID[2]
S1-7	PD	Redundant Memory Available
S1-8	PD	Copy ASW image/SPW default frequency
S2-1	PD	Boot source 0
S2-2	PD	Boot source 1
S2-3	PD	Remote access/Boot from memory
S2-4	PD	Reserved not used

Frontgrade Gaisler AB
Kungsgatan 12
411 19 Göteborg
Sweden
frontgrade.com/gaisler
sales@gaisler.com
T: +46 31 7758650
F: +46 31 421407