FRONTGRADE Gaisler

NOEL3 User's Manual

Version 1.0, 2025-12-19

Table of contents

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

1. Introduction

1.1. Scope

This document is the user's manual for the NOEL3 IP core. It describes the capabilities of the multi-threaded RV32 RISC-V processor and contains information on use of FPGA configuration that has demonstration systems of NOEL3. The NOEL3 FPGA demonstration designs are described in section 11 of this document.

1.2. Updates and feedback

The latest version of this document and example designs can be found via https://www.gaisler.com/NOEL3.

Feedback: support@gaisler.com

For commercial questions please contact sales@gaisler.com

2. Overview

NOEL3 is a multi-threaded RV32 RISC-V processor implemented as a barrel architecture. It is designed to support deterministic code execution. In this context, deterministic execution time implies that the total execution of a single thread for a certain program remain consistent regardless of the different workloads being run by the other threads of the barrel processor.

2.1. Barrel architecture

A barrel processor is a type of microprocessor implementation where there are as many threads as there are number of stages in the processor's pipeline. Threads are alternated throughout every stage of the pipeline, such that the first stage of the pipeline will start fetching an instruction from thread 0, and the last stage of the pipeline will be retiring an instruction from thread N-1, where N is the number of stages of the pipeline. At each cycle, threads advance through the pipeline, from the fetch stage all the way to retirement, such that an instruction from a different thread is retired at each cycle.

Since each pipeline stage of the processor is performing work from different threads, typical hazards stemming from pipelined processors are no issue, and there is no need for a forwarding network, since no information must be routed to different stages of the pipeline (each thread performs its own work independently of the others). The lack of forwarding logic will help to reduce critical paths, yielding greater frequencies, and allowing for ease of verification since each stage is independent of the rest. Also, a single thread will not need to know the outcome of a branch before it reaches the end of the pipeline, since the preceding pipeline stages will be doing useful work from other threads - this removes the need for branch prediction.

3. Pipeline (integer unit)

The pipeline has a number of stages that can be configured from 2 to 6 stages. The configuration with four stages is expected to provide the best compromise between number of threads and performance per thread.

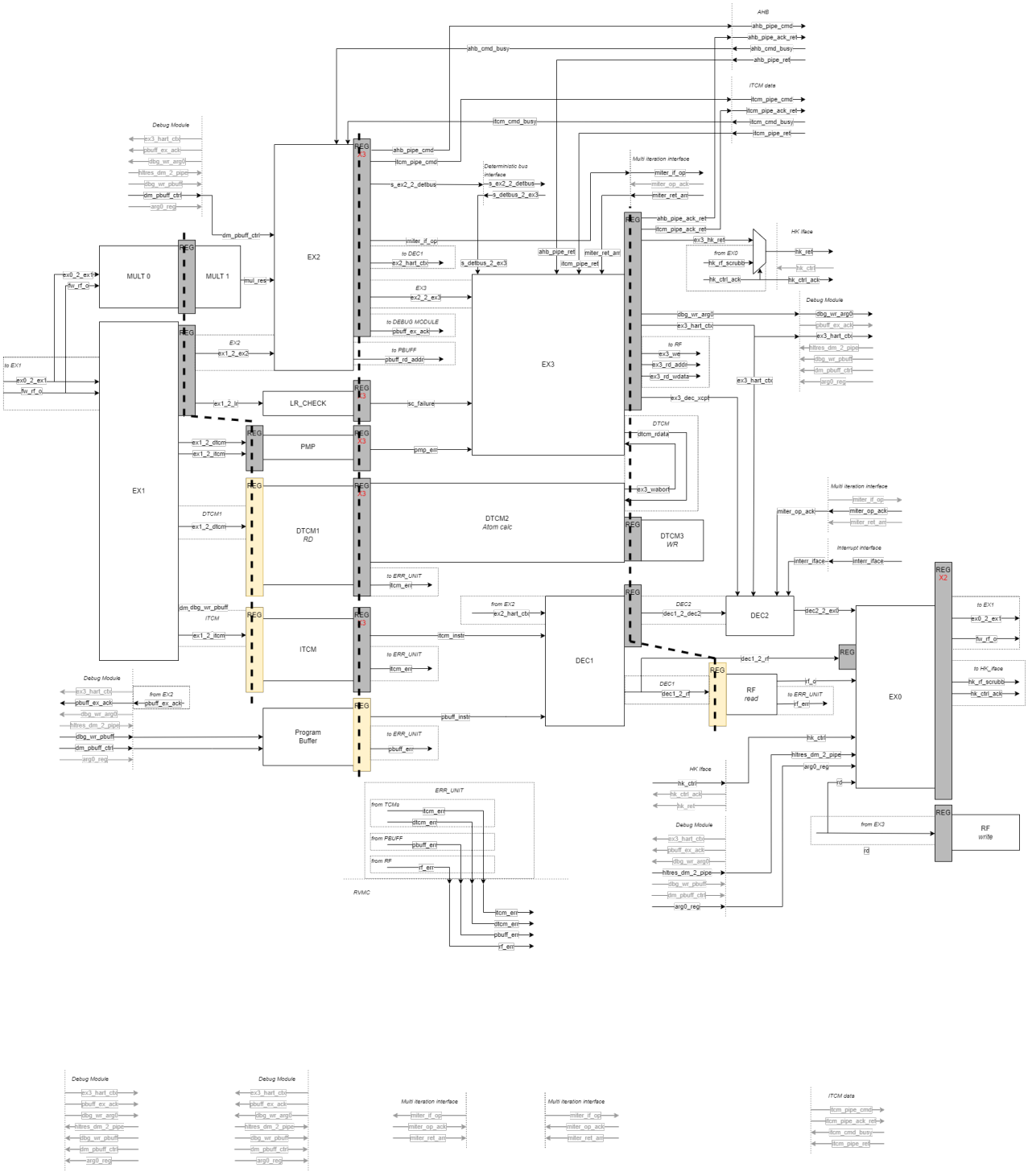


Figure 1. Pipeline blockdiagram

When the processor leaves reset the first instruction will take an extra iteration through the pipeline. The starting address is set to 0x80000000.

3.1. Multi-iteration operations

While most instructions are handled by a single pass through the NOEL3 pipeline, that is not always possible. Non-deterministic operations such as accesses towards external memory will take longer, but also division, floating-point operations and certain CSR accesses.

The term used in the NOEL3 documentation for one pass through the pipeline is an "iteration". Thus, multi-iteration operations are those that have to do one or more extra passes.

Dealing with exceptions will also take multiple iterations, reading and updating CSRs, both before entering the handler and when returning from it.

3.2. Supported extensions

The NOEL3 is configurable, with the full set of supported extensions being:

Table 1. RISC-V Unprivilege specification (Version 20250508)

Extension	Version	Description
RV32I	2.1	The base ISA.
M	2.0	Integer multiply and divide
A	2.1	Atomic operations
F	2.2	32-bit floating point
B	1.0	The bit manipulation extension (ie Zba, Zbb, Zbs)
Zicond	1.0	Conditional assignment operations
Zimop	1.0	No-operation instructions
Zifencei	2.0	Instruction fence.
Zicsr	2.0	CSR registers
Zicntr	2.0	Basic performance CSRs.

Table 2. RISC-V Privilege specification (Version 20250508)

Extension	Version	Description
Machine ISA	1.13	Machine-Level ISA

4. Memory system

NOEL3 has two internal tightly-coupled memories, one for data and another for instructions (see 4.2 and 4.3), and access to two buses:

- **Deterministic bus**: Mainly intended for deterministic access to I/O, but it is also possible to have data memory there. See 4.4.
- **Non-deterministic bus**: Mainly intended for large external RAM, but can also be used for peripherals. As the name implies, access time to this is non-deterministic. It is also quite slow. See 4.5.

4.1. Memory map

Table 3 shows the address ranges defined for the different memories and interfaces.

Table 3. Memory map

Range	Function
0x80000000 - 0x9FFFFFFF	Instruction tightly coupled memory (ITCM)
0x40000000 - 0x5FFFFFFF	Data tightly coupled memory (DTCM)
0x20000000 - 0x3FFFFFFF	Deterministic peripheral bus interface
0x00000000 - 0x1FFFFFFF	Non-deterministic AHB bus interface

4.2. Tightly-coupled data memory (DTCM)

The memory will be inferred using two-port memory, and is divided into blocks. If the memory size requires multiple memory blocks, some internal hardware will be duplicated per block generated to improve timing. DTCM size can be configured in the configuration files, both total size and the size of each individual block.

4.3. Tightly-coupled instruction memory (ITCM)

ITCM follows the same architecture as DTCM, except for a simpler internal structure due to the lack of support for some kinds of accesses, such as read-modify-writes needed by atomics. ITCM size can be configured in the configuration files, both total size and the size of each individual block.

4.4. Peripheral bus interface

Since one of the main ideas about the NOEL3 is to be able to run code deterministically, access to at least some peripherals (and possibly on-chip memory other than the TCMs) must also be deterministic. While it would be possible to do this using hart-specific buses for each hart, this can severely complicate the usage. Instead, NOEL3 uses a modification of the APB2 standard where all the slaves must answer in the same cycle, which allows each hart to do an APB-like access from EX2, and get a response in the same cycle. In short, the peripheral bus interface, hereafter referred-to as deterministic bus, behaves in such a manner:

- All slaves connected to the deterministic bus shall answer in the same cycle as the command arrives.
- No atomic operations are allowed through this bus, only memory-mapped peripherals.
- No other masters can be connected to the deterministic bus, since that would affect the deterministic behavior of the core.

4.4.1. Overview

The deterministic bus implemented for NOEL3 is an APB2-like memory-mapped peripheral bus, whose key distinguishing feature is determinism: all deterministic bus (detbus) peripherals must be combinational. A detbus write or read request presented by the detbus master is forwarded combinationally to all slaves, and the matching slave's response is immediately visible to the master within the same cycle. The master interface can be optionally implemented with registers on its outputs, allowing the combinational path to be broken at the master output. The output register is enabled for implementations with higher number of pipeline stages.

Detbus peripherals are memory mapped. They must be purely combinational, such that the design of more complex peripherals must also entail the consideration of their connection to either the deterministic bus or the multi-iteration interface. We reserve the details of the specific decoding of detbus addresses for a future, final version of the document, as this is still subject to change as we finalize the overall memory map distribution of NOEL3.

At reset, the detbus master and all peripherals shall have all their outputs be zero-filled responses.

The deterministic bus has two types of components:

- **Detbus master**
  - **Pipeline-to-master detbus request**: the detbus master takes as input from the pipeline's EX2 stage the target address to interact with a certain peripheral, a write-enable flag and write data.

  - **Master-to-slaves request broadcast**: sends a single peripheral request to all peripherals, such that each peripheral will check a different bit of a one-hot selection array generated by the detbus master. This is where a future memory-mapped organization of different peripherals determine which bit of the one-hot selection array will be enabled
  - **Master-to-pipeline response**: takes as input an array of peripheral responses and returns the read data from the targeted peripheral into the pipeline's EX3 stage. The response is sent either immediately or registered, depending on the number of stages of NOEL3.

- **Detbus slave(s)**
  - **Master-to-slaves request broadcast**: all slaves take as input the same request coming from the detbus master.
  - **Request processing**: a peripheral may generate a response bound to the detbus master and/or update its internal state depending on whether its selection bit (configured by a generic) was enabled or not. Detbus slaves must always respond/operate combinationally.
    - Reads:
      - Selection bit not enabled, a zero-filled response is generated.
      - Selection bit enabled, response is generated.
    - Writes:
      - Selection bit not enabled, internal state is not updated, zero-filled response is generated.
      - Selection bit enabled, internal state is updated. Behavior on writes is left at the discretion of each peripheral, choosing to return a valid response or not.

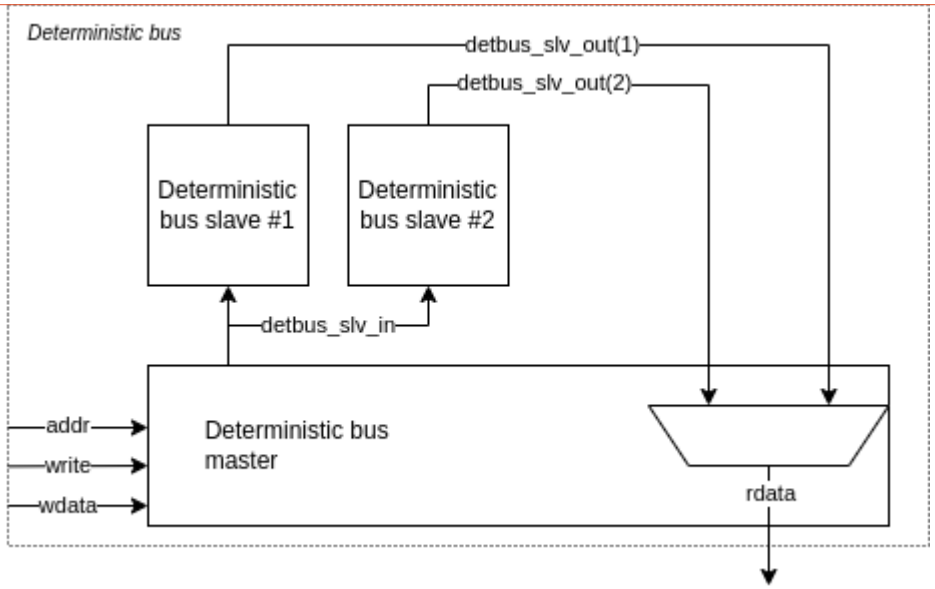


Figure 2. Deterministic bus block diagram

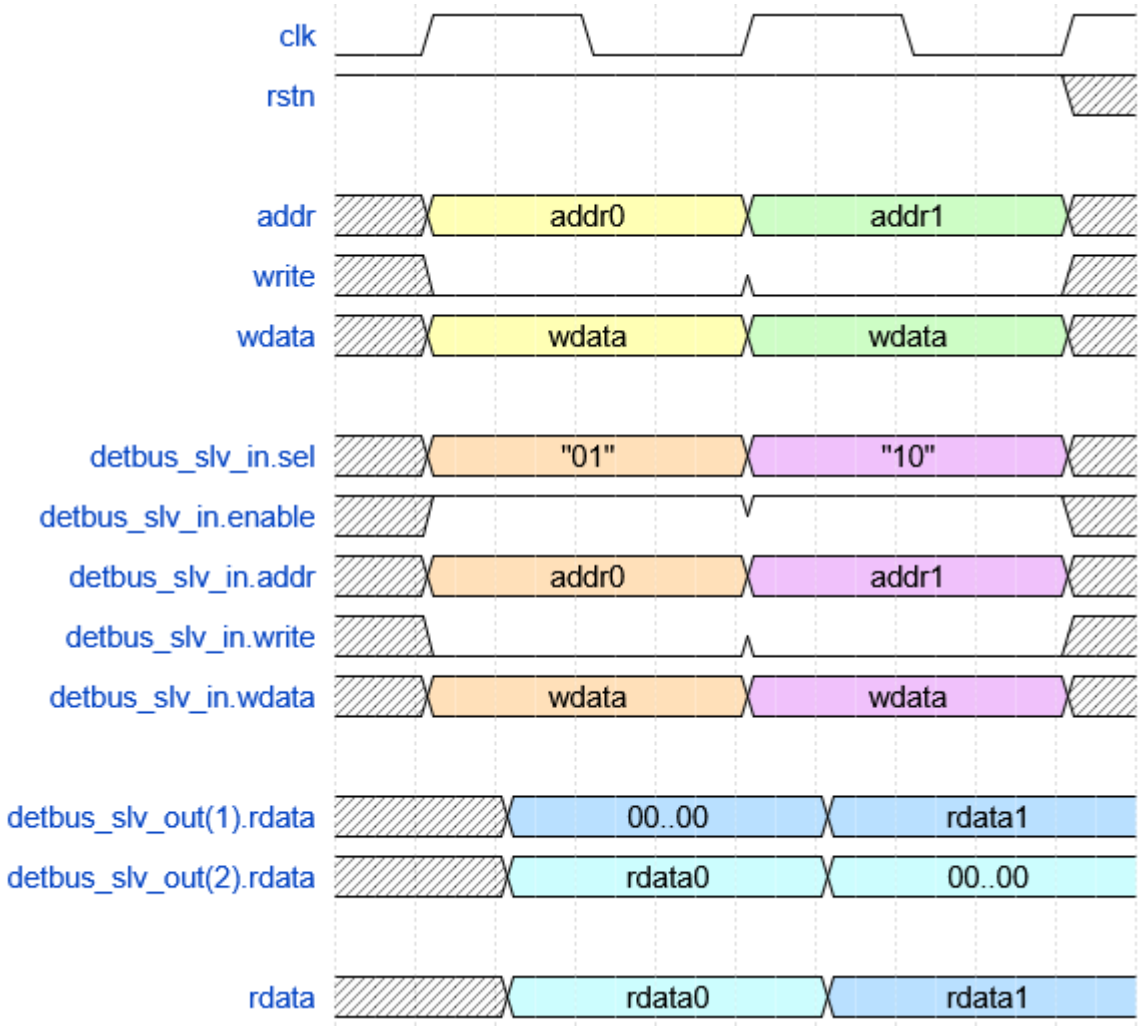


Figure 3. Consecutive reads to different peripherals

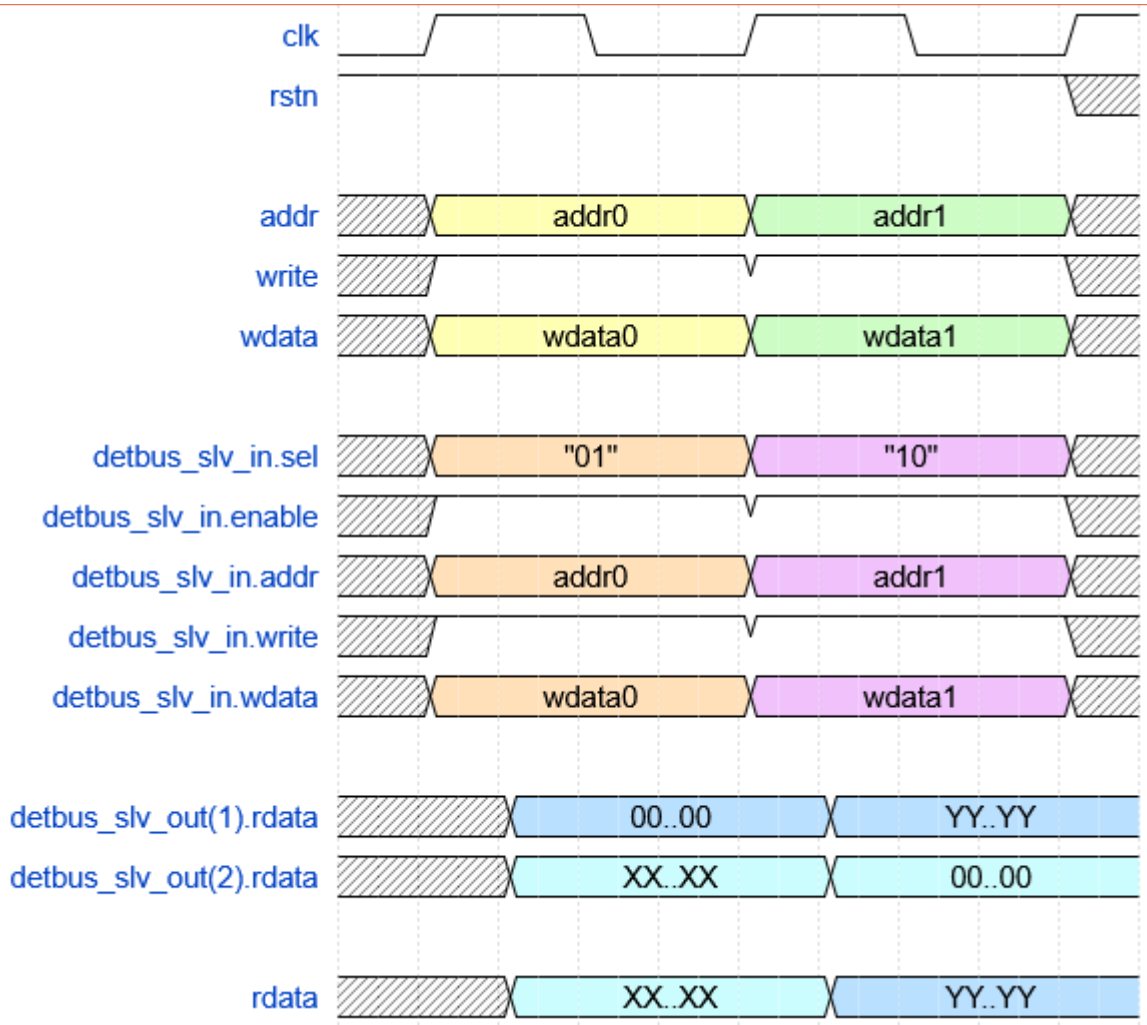


Figure 4. Consecutive writes to different peripherals

4.4.2. Notes

A pipelined bus could allow for starting an access every cycle while still reducing the critical path (assuming no atomic operations, or handling them at the destination). This is not be attempted for the NOEL3 core.

The scheme above does not allow for any external accesses to the bus, since that would make things non-deterministic. While it would be possible to modify the scheme to allow external masters, by for example giving full priority to the processor or extending latency. This will not be attempted for the NOEL3 core.

Separate non-deterministic bus interfaces can also be connected to the NOEL3, but using them will then obviously void any guarantees about determinism for the given hart. For example accesses to AHB bus, more about that in 4.5.

4.5. External bus access (AHB)

Connection to an external bus is mandatory in any microcontroller. The deterministic bus is not enough, since it will only allow a limited number of compatible slave IPs, see 4.4. An AHB master interface is provided towards an external bus, where any AHB IP can be connected.

Since the AHB is a shared resource between all threads, with behavior that is dependent on external events, we can treat it in a similar manner to the ITCM data access as follows:

1. In iteration 1 to N: Instruction notices that it needs to read/write from/to AHB and sends the request through the AHB internal operations interface if it is not busy. This step will be retried until the command can be sent.
2. N+1 to M-1: The pipeline iterates indefinitely, until it gets a response from the AHB return command, which will make it do one last iteration.
3. Last iteration (M): If load, the data is gathered from the AHB return command and stored in the RF. Ack is sent to the AHB operations interface.

This bus connection will only be for data - no execution is allowed through it. Code could be copied though, from the AHB into the ITCM and DTCM to then be executed there.

NOTE As the number of cycles taken by the AHB operations interface depends on other thread behavior (taking up the shared resource), as well as external events (slave could take quite long to answer), any AHB access will be non-deterministic.

5. Control and Status Registers (CSRs)

The NOEL3 provides the CSR registers required by its supported RISC-V extensions - most of them as one instance per thread.

Note that, due to the internal workings of the core, not all CSRs can be accessed in the same amount of time as described below.

5.1. CSRs in flip-flops

Any CSRs that need to be accessed implicitly during execution are stored in flip-flops. Except for MCYCLE[H], these are all per thread.

CSR	Addr (hex)	Implemented readable fields	Writable fields	Notes
fflags	001	NV, DZ, OF, UF, NX	NV, DZ, OF, UF, NX	
frm	002	rounding mode	rounding mode	
fcsr	003	`frm[7:5]`, `fflags[4:0]`	`frm[7:5]`, `fflags[4:0]`	Combination of fflags and frm.
mstatus	300	`FS`, `MPP`, `MPIE`, `MIE`	`FS`, `MPP`, `MPIE`, `MIE`	
mie	304	`MSIE`, `MTIE`, `MEIE`	`MSIE`, `MTIE`, `MEIE`	
mip	344	`MSIP`, `MTIP`, `MEIP`		
minstret	B02	all bits	all bits	
minstreth	B82	all bits	all bits	
mcycle	B00	all bits	all bits	shared
mcycleh	B80	all bits	all bits	shared
dcsr	7B0	`xdebugver[31:28]`, `ebreakm`, `cause[2:0]`, `prv[1:0]`	`ebreakm`	For debug module use.
arg0	7FF	all bits	all bits	For debug module use.

5.2. CSRs in registers

To use less resources, CSRs that do not need to be accessed implicitly are put in the same memory block as the register files. Accessing these registers normally takes two hart iterations,

but this time is doubled for read-modify-write with the same integer register as source and destination.

CSR	Addr (hex)	Implemented readable fields	Writable fields	Notes
mtvec	305	BASE, MODE	BASE, MODE	
mscratch	340	all bits	all bits	
mepc	341	all bits	all bits	
mcause	342	Interrupt, Exception Code	Interrupt, Exception Code	
dpc	7B1	all bits	all bits	For debug module use.

5.3. CSRs with constant value

The following CSRs are available for SW to discover capabilities of the implementation (cap*)

CSR	Addr (hex)	Implemented readable fields	Notes
misa	301	MXL, ISA extension bits	
mhartid	F14	hartid	Is unique for each hart.
cap[h]	FC0/FD0	all bits	
cap2[h]	FC1/FD1	all bits	
cap3[h]	FC2/FD2	all bits	
cap4[h]	FC3/FD3	all bits	
cap5[h]	FC4/FD4	all bits	

6. Floating point unit (FPU)

The FPU, which supports the RISC-V F extension (32-bit floating-point), has its own set of 32-bit registers. It is fully pipelined, and interfaced to the rest of the NOEL3 in such a way that the execution is fully deterministic. Each floating-point operation will finish after two iterations of the issueing thread - half the speed of most integer operations.

To use less resources, the FPU is configured without hardware support for divide and square root. While currently not being RISC-V compliant, this is supported by gcc (-mno-fdiv).

The NOEL3 currently support implementation with the OpenHW Group cvfpu/FPnew FPU. Support for additional FPU implementations will be added in future updates.

7. Integer division

Doing integer division in a pipelined fashion is very expensive. An iterative hardware implementation, which is cheap, can still be significantly faster than software, though. The NOEL3 supports such an implementation, connected via a special interface that guarantees deterministic execution time even for iterative connected units - by making all operations take the maximum possible time, times the number of threads.

An integer division on NOEL3 currently always takes 33 iterations, no matter how many threads try to do it simultaneously.

The NOEL3 currently makes use of serdiv from the OpenHW Group cva6.

8. Debug support

The NOEL3 processor implements version 1.0 of the RISC-V Debug Specification.

8.1. Debug module

Only features required by the specification together with the features defined in this section are supported by NOEL3, to minimize the size and complexity of the implementation.

8.1.1. Program buffer

Debug access to CPU registers (GPR, CSR, FPR) and memory is supported using the program buffer and abstract commands. To read and write GPR via the abstract command, a custom CSR is implemented (0x7FF). This CSR is only accessible when in debug mode and should not be accessed directly. Only purpose is to map data0 into CSR address space.

The optional feature Abstract Command Autoexec is supported to automatically rerun the abstract command when a data register has been accessed. This feature can be used to optimize memory accesses, and currently the free debugger openOCD relies on this feature existing.

8.1.2. Hart control

In additional to all required features for controlling (halt, resume, reset of a single hart), the optional feature to control multiple harts (all harts in NOEL3) at the same time is supported.

9. Fault tolerance

NOEL3 protects memory blocks by either enabled built-in ECC features in supported FPGA technologies or implement a RTL ECC solution using BCH (39,32,7) coding. To prevent error build-up a scrubber is implemented to scrub TCMs and register file (RF). Detection of correctable and uncorrectable error is signaled to the SoC design.

9.1. Clock and Reset

The NOEL3 is designed with a single clock and active low synchronous reset domain.

9.2. Technology mapping

All memory blocks in the design use a two-port memory (one read and one write port). To support read and write at the same time to the same address, write-through logic has been added in the wrapper around the memory, rather than relying on the target technology to provide this functionality. In order to prevent simultaneous read and write operations towards the memory, the read-enable is masked in this case.

The data width for all memory blocks is 32 bit. This makes it possible to use the built-in ECC (when available in the target FPGA) and also adding 7 check bits for an RTL ECC solution.

The mapping of the instantiated memories to the targeted technology is done in a technology mapping layer. This makes the processor implementation code technology-independent.

10. Configuration options

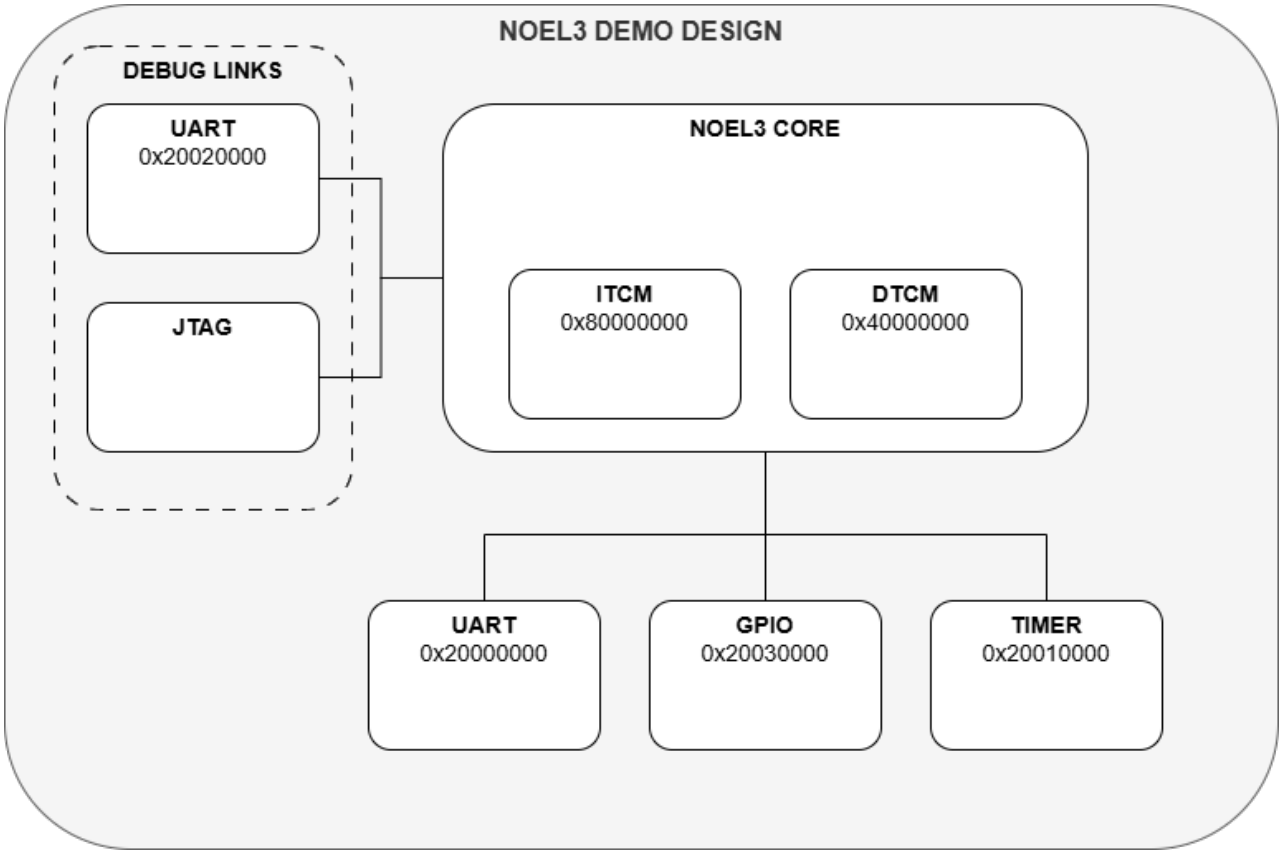
Table 4. Configuration

Generic	Function	Allowed range	Default
memtech	Vendor library for regfile and TCM RAMs.	0 - NTECH	0 (inferred)
hmindex	AHB master index	0 - NAHBMST-1	0
nharts	Number of active harts	1 - 3	3
itcmconf	Size of instruction tightly coupled memory in Kbytes.	0 - 16#FFFF#	1024
dtcmconf	Size of data tightly coupled memory in Kbytes.	0 - 16#FFFF#	1024
hindex_dm	AHB slave index for debug-module	0 - NAHBSLV-1	0
haddr_dm	ADDR field of AHB BAR for the debug-module	0 - 16#FFF#	0
hmask_dm	MASK field of AHB BAR for the debug-module	0 - 16#FFF#	0
hindex	AHB slave index for TCM and deterministic bus access	0 - NAHBSLV-1	0
haddr_itcm	ADDR field of AHB BAR for the instruction tightly coupled memory	0 - 16#FFF#	16#800#
hmask_itcm	MASK field of AHB BAR for the instruction tightly coupled memory	0 - 16#FFF#	16#E00#
haddr_dtcm	ADDR field of AHB BAR for the data tightly coupled memory	0 - 16#FFF#	16#400#
hmask_dtcm	MASK field of AHB BAR for the data tightly coupled memory	0 - 16#FFF#	16#E00#
haddr_dbus	ADDR field of AHB BAR for the deterministic bus	0 - 16#FFF#	16#200#
hmask_dbus	MASK field of AHB BAR for the deterministic bus	0 - 16#FFF#	16#E00#

11. Example bitstream

11.1. SoC design

A set of demonstrator designs are available for NOEL3 testing and evaluation. Except for sizes of instruction memory (ITCM) and data memory (DTCM), they are all based on the same processor configuration and peripherals, as outlined in the blockdiagram. 3 hardware threads are activated in the NOEL3 core.



Peripheral Block	Base Address	GRLIB IP CORE
UART (debug link)	0x20020000	AHBUART- AMBA AHB Serial Debug Interface
JTAG	(NA)	AHBJTAG - JTAG Debug Link with AHB Master Interface
UART	0x20000000	APBUART - AMBA APB UART Serial Interface
GPIO	0x20080000	GRGPIO - General Purpose I/O Port
TIMER	0x20010000	GPTIMER - General Purpose Timer Unit

For details on the included peripehrals, refer to the GRLIB IP Core User's Manual

After loading the target FPGA with the example bitstream, the hardware debugger GRMON is the main tool for interacting with the design.

For details on the GRMON hardware debugger, refer to the GRMON4 User's Manual http://www.gaisler.com/GRMON.

11.1.1. GRMON quick start guide

For accessing processor registers, peripherals, memories etc and uploading applications, GRMON hardware debugger is used.

GRMON is used with UART debug link (a JTAG debuglink is also implemented).

The peripheral UART output is easily monitored with GRMON with UART forwarding.

NOTE	Please note that support for NOEL3 in GRMON is currently at `beta` level and there is some limitations regarding available features.

To connect with GRMON to the following command is used:

```
grmon -uart /dev/ttyUSB15 -u uart0
```

GRMON switches

-uart, with this swtich the desired host UART is selected for connecting to the target debuglink. In the example above a Linux host system is used. On a Windows host system the /dev/ttyUSB argument would be one of the COM ports. Note that the UART debug link automatically detects the correct baudrate.

-u, This switch enables the forwardig of the peripeherla UART ooutput to GRMON. This is a convinient way to monitor the UART output without having to establish a seaprete conection for the UART TX/RX signals.

After successfully connecting to the NOEL3 system the printout below is expected.

```
  GRMON debug monitor v4.0.7-67-gb539d65f6 64-bit eval version

  Copyright (C) 2025 Frontgrade Gaisler - All rights reserved.
  For latest updates, go to https://www.gaisler.com/
  Comments or bug-reports to support@gaisler.com

  This eval version will expire on 18/06/2026

Parsing -uart /dev/ttyUSB15
  using port /dev/ttyUSB15 @ 115200 baud
  Device ID:           0x296
  GRLIB build version: 4299
  Detected system:     NOEL3 SOC
  Detected frequency:  50.0 MHz

  Component                            Vendor
  NOEL3 RISC-V Processor               Frontgrade Gaisler
  AHB Debug UART                       Frontgrade Gaisler
  JTAG Debug Link                      Frontgrade Gaisler
  AHB/APB Bridge                       Frontgrade Gaisler
  NOEL3 debug module                   Frontgrade Gaisler
  Generic UART                         Frontgrade Gaisler
  Modular Timer Unit                   Frontgrade Gaisler
  General Purpose I/O port             Frontgrade Gaisler

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

```
grmon4> info sys
  cpu0       Frontgrade Gaisler  NOEL3 RISC-V Processor
             AHB Master 0
             AHB: 80000000 - a0000000
             AHB: 40000000 - 60000000
             AHB: 20000000 - 40000000
  ahbuart0   Frontgrade Gaisler  AHB Debug UART
             AHB Master 1
             APB: 20020000 - 20020100
             Baudrate 115200, AHB frequency 50.00 MHz
  ahbjtag0   Frontgrade Gaisler  JTAG Debug Link
             AHB Master 2
  apbmst0    Frontgrade Gaisler  AHB/APB Bridge
             AHB: 20000000 - 20100000
  dm0        Frontgrade Gaisler  NOEL3 debug module
             AHB: fe000000 - ff000000
```

```
            hart0: ISA rv32imafb, Modes M
                   i, m, a, f, smepmp, smrnmi, smstateen, sscofpmf, zaamo
                   zalrsc, zba, zbb, zbc, zbs, zcb, zcmp, zicntr, zicond
                   zifencei, zihpm, zimop, zk
                   Stack pointer 0x3ffffff0
                   icache not implemented
                   dcache not implemented
                   1 triggers,
            hart1: ISA rv32imafb, Modes M
                   i, m, a, f, smepmp, smrnmi, smstateen, sscofpmf, zaamo
                   zalrsc, zba, zbb, zbc, zbs, zcb, zcmp, zicntr, zicond
                   zifencei, zihpm, zimop, zk
                   Stack pointer 0x3ffffff0
                   icache not implemented
                   dcache not implemented
                   1 triggers,
            hart2: ISA rv32imafb, Modes M
                   i, m, a, f, smepmp, smrnmi, smstateen, sscofpmf, zaamo
                   zalrsc, zba, zbb, zbc, zbs, zcb, zcmp, zicntr, zicond
                   zifencei, zihpm, zimop, zk
                   Stack pointer 0x3ffffff0
                   icache not implemented
                   dcache not implemented
                   1 triggers,
  uart0     Frontgrade Gaisler  Generic UART
            APB: 20000000 - 20000100
            IRQ: 1
            Baudrate 38343, FIFO debug mode available
  gptimer0  Frontgrade Gaisler  Modular Timer Unit
            APB: 20010000 - 20010100
            IRQ: 2
            16-bit scaler, 2 * 32-bit timers, divisor 50
  gpio0     Frontgrade Gaisler  General Purpose I/O port
            APB: 20030000 - 20030100
```

Now GRMON is connected and ready for user commands.

11.2. GR740-MINI

The GR740-MINI evaluation board is a compact evaluation platform designed to showcase the capabilities of the CertusPro-NX FPGA and the GR740 quad-core LEON4FT SoC.

The NOEL3 example design for this board is entirely based around the CertusPro-NX FPGA independently from the GR740 chip.

The design is loaded with openFPGALoader via the USB connector J2 (tested loader, other loaders exist).

After the FPGA design has been loaded LED13 should flash slowly as a sanity check that FPGA PLL clock is present.

UART debug link is bridged via the board USB connection J2 on the GR740-MINI evaluation board.

Debug UART TX/RX signal are mapped to LEDs for debugging purpose. When GRMON communicates with the board the LEDs 15 and 14 should flicker.

11.2.1. Frequency and Resource utilization

The NOEL3 system clock is at 50 MHz and both data and instruction memories set to 128kB, using 72% of the available block BRAM in FPGA.

Core frequency	50 MHz
BRAMs	151
FFs	6416
LUTs	9870

11.3. Avant-X MINI

With this board NOEL3 can be evaluated on the Avant-X high-speed mid-range FPGA.

The design is loaded with Radiant programmer from Lattice via the USB connector J2 (tested loader, other loaders exist).

After the FPGA design has been loaded LED1 should flash slowly as a sanity check that FPGA PLL clock is present.

UART debug link is bridged via the board USB connection J2 on the Avant-X MINI evaluation board.

Debug UART RX signal are mapped to a LED for debugging purpose. When GRMON communicates with the board the LED0 should flicker.

11.3.1. Frequency and Resource utilization

The NOEL3 system clock is at 80 MHz and both data and instruction memories set to 512kB, using 27% of the available block BRAM in FPGA.

Core frequency	80 MHz
BRAMs	270
FFs	6583
LUTs	10500

11.4. Microsemi Polarfire splash kit

The design is loaded with openFPGALoader via the USB connector J1 (tested loader, other loaders exist).

After the FPGA design has been loaded LED4 should flash slowly as a sanity check that FPGA PLL clock is present.

UART debug link is bridged via the board USB connection J1 on the Polarfire splash kit.

Debug UART TX/RX signal are mapped to LEDs for debugging purpose. When GRMON communicates with the board LEDs 2 and 3 should flicker.

11.4.1. Frequency and Resource utilization

The NOEL3 system clock is at 50 MHz and both data and instruction memories set to 512kB, using 54% of the available block BRAM in FPGA.

Core frequency	50 MHz
BRAMs	518
FFs	6433
LUTs	13495

11.5. Xilinx KCU105

The NOEL3 example design for the Xilinx KCU105 is loaded with the Xilinx Vivado toolset.

For this design JTAG as is used as for the debug link:

```
grmon -digilent -jtagserial 210308AC5C0E
```

After starting GRMON with the command above a printout as under 11.1.1

11.5.1. Frequency and Resource utilization

The NOEL3 system clock is at 50 MHz and both data and instruction memories set to 1MB, using 97% of the available block BRAM in FPGA.

Core frequency	50 MHz
BRAMs	585
FFs	5656
LUTs	10503