NOEL-XCKU-EX Quick Start Guide

The most important thing we build is trust
# Table of Contents

1. Introduction .................................................................................................................. 4
   1.1. Overview ................................................................................................................. 4
   1.2. Availability ............................................................................................................. 4
   1.3. Prerequisites ........................................................................................................... 4
   1.4. References .............................................................................................................. 4

2. Overview .......................................................................................................................... 5
   2.1. Boards ...................................................................................................................... 5
   2.2. Design summary ...................................................................................................... 5
   2.3. Processor features ................................................................................................... 5
   2.4. Software Development Environment ....................................................................... 5
      2.4.1. RTEMS ............................................................................................................ 5
      2.4.2. Linux ................................................................................................................ 5
      2.4.3. VxWorks 7 ....................................................................................................... 6
      2.4.4. GRMON ........................................................................................................... 6

3. Board Configuration ......................................................................................................... 7
   3.1. Buttons and switches ............................................................................................... 7
   3.2. LEDs ......................................................................................................................... 7
   3.3. Connectors ............................................................................................................... 7
   3.4. Memories .................................................................................................................. 7
   3.5. Programming the bitstream ..................................................................................... 7
   3.6. FPGA configuration mode ....................................................................................... 7

4. GRMON hardware debugger .......................................................................................... 9
   4.1. Overview .................................................................................................................. 9
   4.2. NOEL-V support ..................................................................................................... 9
   4.3. NOEL-V limitations ............................................................................................... 9
   4.4. Debug-link alternatives ......................................................................................... 10
      4.4.1. Connecting via the Digilent USB/JTAG interface ............................................. 10
      4.4.2. Connecting via the Ethernet debug interfaces ................................................... 10
      4.4.3. Connecting via the UART debug link .............................................................. 10
   4.5. First steps ............................................................................................................... 10
   4.6. Connecting to the board ......................................................................................... 10
   4.7. Get system information ......................................................................................... 14
   4.8. Load a RAM application ......................................................................................... 14
   4.9. Debugging with GDB ............................................................................................ 14

5. RTEMS Real Time Operating System ............................................................................ 17
   5.1. Overview ............................................................................................................... 17
   5.2. Features ................................................................................................................. 17
   5.3. Install toolchain and kernel ................................................................................... 17
   5.4. Building an RTEMS sample application ................................................................ 17
   5.5. Running and debugging with GRMON ................................................................... 17
   5.6. Running with the Spike RISC-V ISA Simulator ...................................................... 19
   5.7. RISC-V and NOEL-V integration with RTEMS ....................................................... 19
      5.7.1. CSRs ............................................................................................................... 19
      5.7.2. Clock tick .......................................................................................................... 19
      5.7.3. Exceptions ........................................................................................................ 19
      5.7.4. NOEL-V BSP variants ...................................................................................... 20
      5.7.5. Console driver ................................................................................................. 20
      5.7.6. Memory layout ................................................................................................. 20
      5.7.7. Work area ........................................................................................................ 20
      5.7.8. Symmetric Multiprocessing .............................................................................. 20
   5.8. Device tree .............................................................................................................. 20
      5.8.1. Background ....................................................................................................... 20
      5.8.2. GRMON .......................................................................................................... 20
      5.8.3. Spike ............................................................................................................... 21
   5.9. Compiler options ..................................................................................................... 21
5.10. Building the kernel .............................................................. 21
  5.10.1. RTEMS test suite .......................................................... 22
  5.11. Building the tool chain ...................................................... 22

6. Linux .......................................................................................... 23
  6.1. Overview .............................................................................. 23
  6.2. Step by step instructions ..................................................... 23

7. RTEMS Example applications ...................................................... 27
  7.1. hello ..................................................................................... 27
  7.2. tasks ..................................................................................... 27
  7.3. dhrystone .............................................................................. 27
  7.4. coremark ............................................................................ 27
  7.5. demo ..................................................................................... 28
  7.6. Creating a custom application ............................................. 28

8. Support ...................................................................................... 29
1. Introduction

1.1. Overview

This document is a quick start guide for the NOEL-XCKU-EX design. The guide is mainly how-to oriented and does not go into many technical details. For more in-depth information we refer to respective products User's Manual. See the reference list below.

1.2. Availability

The FPGA bitstreams and software environment is available on the NOEL-XCKU-EX web page: https://www.gaisler.com/noel-xcku.

1.3. Prerequisites

To use the provided bitstream, the user needs:

- Xilinx KCU105 board

1.4. References

Table 1.1. References

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RD-3</td>
<td>RTEMS homepage [<a href="https://www.rtems.org">https://www.rtems.org</a>]</td>
</tr>
<tr>
<td>RD-4</td>
<td>RTEMS User Manual [<a href="https://docs.rtems.org/branches/master/user/index.html">https://docs.rtems.org/branches/master/user/index.html</a>]</td>
</tr>
<tr>
<td>RD-6</td>
<td>Buildroot homepage [<a href="https://www.buildroot.com">https://www.buildroot.com</a>]</td>
</tr>
</tbody>
</table>
2. Overview

2.1. Boards

The NOEL-XCKU-EX design can be used with the Xilinx KCU105 development board ([RD-5]).

2.2. Design summary

The NOEL-XCKU-EX is a GRLIB design which includes the following features:

- Cobham Gaisler NOEL RISC-V RV64G dual-core processor
- RISC-V Debug module
- L2 cache with 256 KiB in 4 ways
- Memory controller and 1 GiB SDRAM.
- Ethernet 10/100/1000 Mbit MAC interface
- APBUART serial interface
- GRLIB AMBA AHB bus controller
- JTAG, Ethernet EDCL and UART debug link
- AHB bus trace
- 20-pin GPIO controller

For more details on the NOEL-XCKU-EX design, see the NOEL-XCKU-EX User's Manual ([RD-1]). For details about the the interfaces' connections in the board, see Chapter 3.

2.3. Processor features

- 64-bit architecture
- Hardware multiply and divide units
- Atomic instruction extension
- 32/64 bit floating point extensions using non-pipelined area efficient FPU or high-performance fully pipelined IEEE-754 FPU
- Machine, supervisor and user mode. RISC-V standard MMU with configurable TLB.
- User level interrupts
- RISC-V standard PLIC (platform interrupt controller)
- RISC-V standard PMP (physical memory protection)
- RISC-V standard external debug support
- Advanced 7-stage dual-issue in-order pipeline
- Dynamic branch prediction, branch target buffer and return address stack
- Four full ALUs, two of them late in the pipeline to reduce stalls
- Separate instruction and data L1 cache (Harvard architecture) with snooping

2.4. Software Development Environment

2.4.1. RTEMS

RTEMS is a hard Real Time Operating System.

The NOEL-V software development environment includes an RTEMS kernel, BSP tool chain and examples. This allows for development of real-time multitasking applications with POSIX support. The RTEMS tool chain is currently provided for the Linux 64-bit host operating systems.

Chapter 5 describes how to use RTEMS with NOEL-XCKU-EX.

The recommended method to load software onto NOEL-XCKU-EX is by connecting to a debug interface of the board through the GRMON hardware debugger (Chapter 4).

2.4.2. Linux

Buildroot can be used to easily create a bootable Linux image for NOEL-V [RD-6]. It automatically creates a toolchain and supports a large number of useful userspace applications which can be included in the generated
root file system. Included in the software development environment is a NOEL-XCKU-EX BSP for Buildroot which provides the necessary driver support.

See Chapter 6 for instructions on how to create a Linux image for NOEL-XCKU-EX with Buildroot.

2.4.3. VxWorks 7

Please contact support@gaisler.com for information about NOEL-V BSPs for VxWorks 7.

2.4.4. GRMON

GRMON is a hardware monitor which allows non-intrusive debugging and execution control of software on NOEL-XCKU-EX. GRMON provides a RISC-V GDB server. GRMON is available for Linux and Windows host operating systems.

NOEL-V can be used with GRMON GUI.

Chapter 4 describes how to use GRMON with NOEL-XCKU-EX.
3. Board Configuration

This chapter describes boards items as used by the NOEL-XCKU-EX design.

Please see KCU105 Board User Guide for a detailed legend of the reference designators.

3.1. Buttons and switches

- **SW12[1..4]** DIP switch: GPIO0 inputs 0, 1, 2, 3.
- **SW7, SW8, SW9, SW10** buttons: GPIO0 inputs 4, 5, 6, 7
- **CPU_RST** button: Main reset to the FPGA design

The Switch SW12[1] also acts as select signal for the UART interface.

- When '1' the UART interface is connected to the UART debug link (AHBUART).
- When '0' the UART interface is connected to the APBUART.

3.2. LEDs

- **LED[0..3]**: Connected to GPIO0 outputs 16, 17, 18, 19
- **LED[4]**: Connected to SW12[1]
- **LED[6..7]**: When ON they indicate that the memory controller calibration is complete and the FPGA design has access to the on-board SDRAM.

3.3. Connectors

- **J87**: USB JTAG interface via Digilent module with micro-B USB connector. See (Chapter 4).
- **J4**: USB UART interface. AHBUART debug link or APBUART function selectable by SW12[1].
- **Ethernet PHY SGMII interface with RJ-45 connector. See (Chapter 4).**
- **J52**: PMOD Connector GPIO0 I/O 8-15.

3.4. Memories

The NOEL-XCKU-EX has 1 GiB of SDRAM available on the on-chip bus.

3.5. Programming the bitstream

A Xilinx Vivado script to program the FPGA is provided with the bitfile distribution. It has been tested using Vivado 2018.1 and 2019.2.

To program the FPGA please follow these instructions:

1. Connect the PC and the board using a standard micro-USB cable into the connector USB-JTAG J87.
2. Make sure that Vivado is added to your path variables
3. Issue the following command:
   ```
   vivado -mode tcl -source doprog.tcl
   ```
4. Once the FPGA has been programmed, it is possible to connect to the board using GRMON, using the command:
   ```
   grmon -digilent
   ```
   Please see (Chapter 4) for further information regarding GRMON and the available debug links.

Alternatively, the bitfile (.bit) can be programmed to the Xilinx KCU105 using the Vivado design suite graphical interface. Start Vivado and select the menu item **Flow -> Open Hardware Manager.** Once the FPGA has been programmed, remember to close the hardware manage before connecting with GRMON.

3.6. FPGA configuration mode

The board switch **SW15[6]** selects the configuration mode for the FPGA device. It is a board configuration and not part of the NOEL-V design.

- **0**: Master SPI configuration mode
• 1: JTAG configuration mode

The default and recommended configuration is to set SW15[6] in position 0 to use SPI configuration mode.
4. GRMON hardware debugger

4.1. Overview

GRMON is a debug monitor used to develop and debug GRLIB systems with NOEL and LEON processors. The target system, including the processor and peripherals, is accessed on the AHB bus through a debug-link connected to the host computer. GRMON has GDB support which makes C/C++ level debugging possible by connecting GDB to the GRMON’s GDB socket.

With GRMON one can for example:

- Inspect NOEL-V 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 NOEL-XCKU-EX design. After that, a basic first inspection of the board is exemplified.


4.2. NOEL-V support

Most of the GRMON commands available for LEON are also available for NOEL-V. GRMON commands available for NOEL-V include:

- `load`: RISC-V ELF file support. Symbols are loaded from the ELF file and can be used instead of addresses for most commands.
- `run`, `cont`, `go`, `step`: execution control
- `mem`, `wmem`: read/write any on-chip address.
- `disassemble`: RISC-V instruction disassembly
- `inst`: CPU instruction trace
- `bp`: Hardware and software breakpoint
- `bt`: call tree backtrace, based on dwarf debug information
- `reg`: read and write all RISC-V CPU general purpose registers and CSR registers. CSR registers can be specified by name or address.
- `mmu`: inspect and walk MMU tables
- `forward`: UART forwarding to the GRMON console
- `info reg`, `info sys`: Supports the NOEL-V related GRLIB devices.
- `gdb`: Creates a GDB server for connecting with a GDB compiled with RISC-V as target.

4.3. NOEL-V limitations

- GRMON can report only the following reasons for termination of execution:
  - An `ebreak` instruction was executed.
  - The signal `haltreq` was asserted by the debug module. Typically as a consequence of the user hitting `ctrl+c` in the GRMON terminal.

In the current NOEL-V release, execution can not be aborted at an arbitrary exception or hardware breakpoint.

- CPU local AHB trace is not available. The NOEL-XCKU-EX design includes an AHBTRACE which can be controlled with the GRMON command `at`.

The limitations listed above are present in the current release of NOEL-V. The features mentioned are part of the schedule for future releases.
4.4. Debug-link alternatives

4.4.1. Connecting via the Digilent USB/JTAG interface

Please see GRMON User’s Manual for information on how to set up the required Digilent Adept driver software. Then connect the PC and the board using a standard USB cable into the USB-mini J87 USB-JTAG connector and issue the following command:

`grmon -ftdi`

4.4.2. Connecting via the Ethernet debug interfaces

If another address is wanted for the Ethernet debug link then one of the other debug links must be used to connect GRMON to the board. The EDCL IP address can then be changed using GRMON’s `edcl` command. This new address will persist until next system reset.

With the Ethernet Debug Communication Link 0 address set to 192.168.0.51 the GRMON command to connect to the board is:

`grmon -eth 192.168.0.51`

4.4.3. Connecting via the UART debug link

Make sure that the switch SW12[1] select the UART debug link (ON position). Connect the PC and the board using a standard micro-USB cable into the connector USB-UART J4 and issue the following command:

`grmon -uart /dev/ttyUSB0`

4.5. First steps

The previous sections have described which debug-links are available and how to start using them with GRMON. The subsections below assume that GRMON, the host computer and the NOEL-XCKU-EX board have been set up so that GRMON 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 GRMON. 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-2].
- Memory location and size configuration is found from the `info sys` output.

4.6. Connecting to the board

In the following example the Xilinix platform cable debug-link is used to connect to the board. The auto-detected frequency, memory parameters and stack pointer are verified by looking at the GRMON terminal output below.

`grmon -digilent`

GRMON debug monitor v3.2.3 64-bit version
Copyright (C) 2020 Cobham Gaisler - All rights reserved.
For latest updates, go to http://www.gaisler.com/
Comments or bug-reports to support@gaisler.com

Commands missing help:

Device ID: 0x287
GRLIB build version: 4251
Detected frequency: 80.0 MHz

Component Vendor
AHB Debug UART Cobham Gaisler
JTAG Debug Link Cobham Gaisler
GR Ethernet MAC Cobham Gaisler
NOEL-V RISC-V Processor Cobham Gaisler
NOEL-V RISC-V Processor Cobham Gaisler
AHB-to-AHB Bridge Cobham Gaisler
RISC-V Debug Module Cobham Gaisler
AHB/APB Bridge Cobham Gaisler
AMBA Trace Buffer Cobham Gaisler
L2-Cache Controller Cobham Gaisler
Generic AHB ROM                      Cobham Gaisler
Xilinx MIG Controller                Cobham Gaisler
AHB/APB Bridge                      Cobham Gaisler
RISC-V CLINT                        Cobham Gaisler
RISC-VPLIC                         Cobham Gaisler
AHB Status Register                 Cobham Gaisler
General Purpose I/O port            Cobham Gaisler
Generic UART                        Cobham Gaisler
Version and Revision Register       Cobham Gaisler
Modular Timer Unit                  Cobham Gaisler

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

grmon3> info sys

ahbuart0  Cobham Gaisler  AHB Debug UART
          AHB Master 0
          APB: 80000e00 - 80000f00
          Baudrate 115200, AHB frequency 80.00 MHz

ahbjtag0  Cobham Gaisler  JTAG Debug Link
          AHB Master 1

greth0    Cobham Gaisler  GR Ethernet MAC
          AHB Master 2
          APB: 80000500 - 80000600
          IRQ: 5
          1000 Mbit capable
          edcl ip 192.168.0.222, buffer 2 kbyte

cpu0  Cobham Gaisler  NOEL-V RISC-V Processor
       AHB Master 0

cpu1  Cobham Gaisler  NOEL-V RISC-V Processor
       AHB Master 1

ahb2ahb0  Cobham Gaisler  AHB-to-AHB Bridge
          AHB Master 3
          AHB: 00000000 - 00000000
          AHB: 80000000 - 90000000
          AHB: e0000000 - 00000000
          USR: 00000111
          USR: ffe00000

dm0  Cobham Gaisler  RISC-V Debug Module
     AHB: 90000000 - a0000000

hart0: DXLEN 64, MXLEN 64, SXLEN 64, UXLEN 64
       ISA A I M, Modes M S U
       Stack pointer 0x7ffffff0
       icache 4 * 4 kB, 32 B/line, rnd
       dcache 4 * 4 kB, 32 B/line, rnd
       2 triggers

hart1: DXLEN 64, MXLEN 64, SXLEN 64, UXLEN 64
       ISA A I M, Modes M S U
       Stack pointer 0x7ffffff0
       icache 4 * 4 kB, 32 B/line, rnd
       dcache 4 * 4 kB, 32 B/line, rnd
       2 triggers

apbmst0  Cobham Gaisler  AHB/APB Bridge
         AHB: 80000000 - 80100000

ahbtrace0  Cobham Gaisler  AMBA Trace Buffer
             AHB: fff00000 - fff20000
             Trace buffer size: 128 lines

l2cache0  Cobham Gaisler  L2-Cache Controller
          AHB: 40000000 - 80000000
          AHB: f0000000 - f4000000
          USR: 00000110
          L2C: 4-ways, cachessize: 256 kbytes, mtrr: 0, AHB SPLIT support

ahbrom0  Cobham Gaisler  Generic AHB ROM
          AHB: 00000000 - 00100000
          32-bit ROM: 1 MB 8 0x00000000

mig0  Cobham Gaisler  Xilinx MIG Controller
       AHB: 40000000 - 80000000
       SDRAM: 1024 Mbyte

apbmst1  Cobham Gaisler  AHB/APB Bridge
         AHB: 80000000 - 80100000

clint0  Cobham Gaisler  RISC-V CLINT
        AHB: e0100000 - e0200000

plic0  Cobham Gaisler  RISC-V PLIC
        AHB: 84000000 - 88000000
        4 contexts, 32 interrupt sources

ahbstat0  Cobham Gaisler  AHB Status Register
          APB: 80000f00 - 80001000
          IRQ: 4

gpio0  Cobham Gaisler  General Purpose I/O port
        APB: 80000400 - 80000500

uart0  Cobham Gaisler  Generic UART
        APB: 80000100 - 80000200
        IRQ: 1
Baudrate 38461, FIFO debug mode available

gptimer0  Cobham Gaisler Modular Timer Unit

NOEL-V RISC-V Processor
ISA and extensions
Machine status register (mstatus) 0x0000000a00000000
Machine interrupt pending (mip) 0x0000000000000000

RISC-V Debug Module
0x90000010 Abstract Data 0 0x00000000
0x90000014 Abstract Data 1 0x00000000
0x90000018 Abstract Data 2 0x00000000
0x9000001c Abstract Data 3 0x00000000
0x90000040 Debug Module Control 0x00010002
0x90000044 Debug Module Status 0x00100073
0x90000048 Hart Info 0x00320344
0x90000050 Hart Array Window Select 0x00000000
0x90000054 Hart Array Window 0x00000003
0x90000058 Abstract Control and Status 0x00320344
0x90000060 Next Debug Module 0x00000000
0x90000064 Custom Features 0x00000000
0x90000068 Program Buffer 0 0x00000000
0x9000006c Program Buffers 0x00000000
0x90000070 Debug Module Control and Status 0x00000000
0x90000074 Halt Summary 0 0x00000003
0x90000078 Custom Features 0 0x00000000
0x90000080 Program Buffer 0 0x00000000
0x90000084 Program Buffer 0 0x00000000
0x90000088 Program Buffer 2 0x10c2a283
0x900000c0 Authentication Data 0x00000000
0x900000c8 Debug Module Control and Status 0x00000000
0x90000100 Halt Summary 0 0x00000003
0x90000104 Custom Features 0 0x00000000
0x90000108 Custom Features 1 0x00000000
0x9000010c Custom Features 1 0x00000000
0x90000110 Custom Features 1 0x00000000
0x90000114 Custom Features 1 0x00000000
0x90000118 Custom Features 1 0x00000000
0x90000120 Custom Features 1 0x00000000
0x90000124 Custom Features 1 0x00000000
0x90000128 Custom Features 1 0x00000000
0x9000012c Custom Features 1 0x00000000
0x90000130 Custom Features 1 0x00000000
0x90000134 Custom Features 1 0x00000000
0x90000138 Custom Features 1 0x00000000
0x90000140 Custom Features 1 0x00000000
0x90000144 Custom Features 1 0x00000000
0x90000148 Custom Features 1 0x00000000
0x90000150 Custom Features 1 0x00000000
0x90000154 Custom Features 1 0x00000000
0x90000158 Custom Features 1 0x00000000
0x90000160 Custom Features 1 0x00000000
0x90000164 Custom Features 1 0x00000000
0x90000168 Custom Features 1 0x00000000
0x90000170 Custom Features 1 0x00000000
0x90000174 Custom Features 1 0x00000000
0x90000178 Custom Features 1 0x00000000
0x90000180 Custom Features 1 0x00000000
0x90000184 Custom Features 1 0x00000000
0x90000188 Custom Features 1 0x00000000
0x90000190 Custom Features 1 0x00000000
0x90000194 Custom Features 1 0x00000000
0x90000198 Custom Features 1 0x00000000
0x900001a0 Custom Features 1 0x00000000
0x900001a4 Custom Features 1 0x00000000
0x900001a8 Custom Features 1 0x00000000
0x900001ac Custom Features 1 0x00000000
0x900001b0 Custom Features 1 0x00000000
0x900001b4 Custom Features 1 0x00000000
0x900001b8 Custom Features 1 0x00000000
0x900001c0 Custom Features 1 0x00000000
0x900001c4 Custom Features 1 0x00000000
0x900001c8 Custom Features 1 0x00000000
0x900001cc Custom Features 1 0x00000000
0x900001d0 Custom Features 1 0x00000000
0x900001d4 Custom Features 1 0x00000000
0x900001d8 Custom Features 1 0x00000000
0x900001de Custom Features 1 0x00000000
0x900001e0 Custom Features 1 0x00000000
0x900001e4 Custom Features 1 0x00000000
0x900001e8 Custom Features 1 0x00000000
0x900001ec Custom Features 1 0x00000000
0x900001f0 Custom Features 1 0x00000000
0x900001f4 Custom Features 1 0x00000000
0x900001f8 Custom Features 1 0x00000000
0x900001fc Custom Features 1 0x00000000

AMBA Trace Buffer
0xff00000  Trace buffer control register 0x00000800
0xff00004 Trace buffer index register 0x000007f0
0xff00008 Trace buffer time tag register 0x00000000
0xff0000c Trace buffer mst/slv filter register 0x00000000
0xff00010 Trace buffer bp 1 address 0x00000000
0xff00014 Trace buffer bp 1 mask 0x00000000
0xff00018 Trace buffer bp 2 address 0x00000000
0xff0001c Trace buffer bp 2 mask 0x00000000

L2-Cache Controller
0xff00000 L2C Control register 0x08040000
0xff00004 L2C Status register 0x4602103
<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xff000020</td>
<td>L2C Error status/control</td>
<td>0x02000000</td>
</tr>
<tr>
<td>RISC-V CLINT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xe01000000</td>
<td>hart0 msip</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0xe0104000</td>
<td>hart0 mtimecmp lo</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>0xe0104004</td>
<td>hart0 mtimecmp hi</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>0xe0100004</td>
<td>hart1 msip</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0xe0104008</td>
<td>hart1 mtimecmp lo</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>0xe010400c</td>
<td>hart1 mtimecmp hi</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>0xe0100008</td>
<td>hart2 msip</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0xe0104010</td>
<td>hart2 mtimecmp lo</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>0xe0104014</td>
<td>hart2 mtimecmp hi</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>0xe010000c</td>
<td>hart3 msip</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0xe0104018</td>
<td>hart3 mtimecmp lo</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>0xe010401c</td>
<td>hart3 mtimecmp hi</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>0xe010bff8</td>
<td>mtime lo</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0xe010bffc</td>
<td>mtime hi</td>
<td>0x00000000</td>
</tr>
<tr>
<td>RISC-V PLIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x84000004</td>
<td>Interrupt source 1 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000008</td>
<td>Interrupt source 2 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8400000c</td>
<td>Interrupt source 3 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000010</td>
<td>Interrupt source 4 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000014</td>
<td>Interrupt source 5 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000018</td>
<td>Interrupt source 6 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8400001c</td>
<td>Interrupt source 7 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000020</td>
<td>Interrupt source 8 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000024</td>
<td>Interrupt source 9 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000028</td>
<td>Interrupt source 10 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8400002c</td>
<td>Interrupt source 11 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000030</td>
<td>Interrupt source 12 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000034</td>
<td>Interrupt source 13 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000038</td>
<td>Interrupt source 14 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8400003c</td>
<td>Interrupt source 15 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000040</td>
<td>Interrupt source 16 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000044</td>
<td>Interrupt source 17 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000048</td>
<td>Interrupt source 18 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8400004c</td>
<td>Interrupt source 19 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000050</td>
<td>Interrupt source 20 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000054</td>
<td>Interrupt source 21 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000058</td>
<td>Interrupt source 22 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8400005c</td>
<td>Interrupt source 23 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000060</td>
<td>Interrupt source 24 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000064</td>
<td>Interrupt source 25 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000068</td>
<td>Interrupt source 26 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8400006c</td>
<td>Interrupt source 27 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000070</td>
<td>Interrupt source 28 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000074</td>
<td>Interrupt source 29 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000078</td>
<td>Interrupt source 30 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8400007c</td>
<td>Interrupt source 31 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000080</td>
<td>Interrupt source 32 priority</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000100</td>
<td>Interrupt Pending bit 0-31</td>
<td>0x00000010</td>
</tr>
<tr>
<td>0x84000104</td>
<td>Interrupt Pending bit 32-63</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000140</td>
<td>context 0 enable source 0-31</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000144</td>
<td>context 0 enable source 32-63</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000148</td>
<td>context 1 enable source 0-31</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8400014c</td>
<td>context 1 enable source 32-63</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000150</td>
<td>context 2 enable source 0-31</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000154</td>
<td>context 2 enable source 32-63</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000158</td>
<td>context 3 enable source 0-31</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8400015c</td>
<td>context 3 enable source 32-63</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x84000160</td>
<td>context 0 priority threshold</td>
<td>0x00000001</td>
</tr>
<tr>
<td>0x84000164</td>
<td>context 1 priority threshold</td>
<td>0x00000001</td>
</tr>
<tr>
<td>0x84000168</td>
<td>context 2 priority threshold</td>
<td>0x00000001</td>
</tr>
<tr>
<td>0x8400016c</td>
<td>context 3 priority threshold</td>
<td>0x00000001</td>
</tr>
<tr>
<td>AHB Status Register</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x800000f0</td>
<td>Status register</td>
<td>0x00000001a</td>
</tr>
<tr>
<td>0x800000f4</td>
<td>Failing address register</td>
<td>0x800000f0</td>
</tr>
<tr>
<td>General Purpose</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x80000040</td>
<td>I/O port data register</td>
<td>0x80000ff0c</td>
</tr>
<tr>
<td>0x80000044</td>
<td>I/O port output register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80000048</td>
<td>I/O port direction register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8000004c</td>
<td>I/O interrupt mask register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8000004d</td>
<td>I/O interrupt polarity register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8000004e</td>
<td>I/O interrupt edge register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8000004f</td>
<td>Capability register</td>
<td>0x00000013</td>
</tr>
<tr>
<td>Generic UART</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x80000104</td>
<td>UART Status register</td>
<td>0x00000006</td>
</tr>
<tr>
<td>0x80000108</td>
<td>UART Control register</td>
<td>0x80000003</td>
</tr>
<tr>
<td>0x8000010c</td>
<td>UART Scaler register</td>
<td>0x00000103</td>
</tr>
<tr>
<td>Modular Timer Unit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x80000300</td>
<td>Scalar value register</td>
<td>0x00000004f</td>
</tr>
<tr>
<td>0x80000304</td>
<td>Scalar reload value register</td>
<td>0x00000004f</td>
</tr>
<tr>
<td>0x80000308</td>
<td>Configuration register</td>
<td>0x000000112</td>
</tr>
<tr>
<td>0x80000310</td>
<td>Timer 0 Value register</td>
<td>0xffffffff</td>
</tr>
</tbody>
</table>
4.7. Get system information

One can limit the output to certain cores by specifying the core(s) name(s) to the `info sys` and `info reg` commands. As seen below the memory parameters, first UART and first Timer core information is listed.

```
grmon3> info sys mig0
mig0      Cobham Gaisler  Xilinx MIG DDR3 Controller
          AHB: 40000000 - 80000000
          SDRAM: 1024 Mbyte
```

```
grmon3> info sys uart0 gptimer0
uart0     Cobham Gaisler  Generic UART
          APB: 80000100 - 80000200
          IRQ: 1
          Baudrate 38422, FIFO debug mode available

gptimer0  Cobham Gaisler  Modular Timer Unit
          APB: 80000300 - 80000400
          IRQ: 2
          16-bit scalar, 2 * 32-bit timers, divisor 80
```

```
grmon3> info reg -v ahbstat0
AHB Status Register
0x80000f00  Status register                         0x00000012
  9     ce                0x0         Correctable error
  8     ne                0x0         New error
  7     hw                0x0         HWRITE on error
  6:3   hm                0x2         HMASTER on error
  2:0   hs                0x2         HSIZE on error

0x80000f04  Failing address register                0x80000f00
```

4.8. Load a RAM application

An application linked to RAM can be loaded directly with the `load` and run with `run`.

```
grmon3> batch /opt/rtems-noel-1.0.3/grmon/noel-xilinx-kcu105.tcl
```

```
grmon3> load hello.elf
40000000 .text                    142.0kB / 142.0kB   [===============>] 100%
400237D0 .rtemsroset                 96B              [===============>] 100%
40024840 .data                      4.4kB /   4.4kB   [===============>] 100%
Total size: 146.44kB (777.96kbit/s)
Entry point 0x40000000
Image hello.elf loaded
```

```
grmon3> forward enable uart0
I/O forwarding to uart0 enabled
```

```
grmon3> run
```

```
hello, world
CPU 0:  Forced into debug mode
  0x4001607c: 00100073  ebreak  <_CPU_Fatal_halt+36>
CPU 1:  Interrupted!
  0x40011018: 10500073  wfi     <_CPU_Thread_Idle_body+0>
```

The line `hello, world` is the program output which is forwarded to the GRMON terminal.

4.9. Debugging with GDB

It possible to connect the GDB debugger to GRMON to be able to debug programs at source level. Either start GRMON with the `-gdb` flag or enter the `gdb` command in GRMON.

```
grmon3> gdb
Started GDB service on port 2222.
```
GDB is included with the RTEMS toolchain as `riscv-rtems5-gdb`:

```bash
user@workstation:~$ riscv-rtems5-gdb
GNU gdb (GDB) 8.3
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=riscv-rtems5".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
Find the GDB manual and other documentation resources online at:
For help, type "help".
Type "apropos word" to search for commands related to "word".
``` (gdb)

Specify the filename of the image to debug using the `file` command:

```bash
(gdb) file /home/user/riscv/demo/hello/hello.exe
Reading symbols from /home/user/riscv/demo/hello/hello.exe...
```

Connect to GRMON using `target extended-remote`:

```bash
(gdb) target extended-remote :2222
Remote debugging using :2222
0x0000000000000000 in ?? ()
```

The image can be loaded onto the target using the `load` command. This needs to be done before starting or restarting the program.

```bash
(gdb) load
Loading section .start, size 0x4c lma 0x40000000
Loading section .text, size 0x132e8 lma 0x40000000
Loading section .rodata, size 0x120a0 lma 0x40013338
Loading section .sdata2, size 0x30 lma 0x400253d8
Loading section .eh_frame, size 0x4 lma 0x40025408
Loading section .init_array, size 0x8 lma 0x40025410
Loading section .fini_array, size 0x8 lma 0x40025418
Loading section .rtemsreset, size 0x68 lma 0x40025420
Loading section .data, size 0x768 lma 0x40025488
Loading section .htif, size 0x1000 lma 0x40025c00
Loading section .sdata, size 0xa8 lma 0x40027000
Start address 0x40000000, load size 158864
Transfer rate: 62 KB/sec, 7943 bytes/write.
```

RTEMS images expect register a1 to contain a pointer to a device tree description. This can be set up with the `ndtb` TCL function provided by the `dts.tcl` script. Use the `mon` prefix to execute a command in GRMON and load the script using `source`:

```bash
(gdb) mon source grmon-scripts/dts.tcl
NOTE: run has been patched to load grmon-scripts/noel-xilinx-kcu105.dtb to 46fffff0
```

The device tree description can then be loaded using `ndtb`. Use the `mon` prefix as the function is located in GRMON’s TCL shell. The `ndtb` function needs to be run before the image is started or restarted.

```bash
(gdb) mon ndtb
Total size: 1.60kB (547.00kbit/s)
Entry point 0x46fffff0
Image grmon-scripts/noel-xilinx-kcu105.dtb loaded
```

Use the `break` command to insert a breakpoint at the `Init` function:
The program can now be executed using the `run` command. GDB should break the execution once the program reaches the `Init` function.

```plaintext
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/user/riscv/demo/hello/hello.exe
Breakpoint 1, Init (ignored=1073914168) at test.c:13
13    puts("\n");
```

At this stage one can, for example, step through the program with `step` or `next`, print the values of variables with `p`, or continue execution with the `cont` command.

```plaintext
(gdb) cont
Continuing.

hello, world
```

The following message is printed if the RTEMS program exits normally.

```plaintext
*** FATAL ***
fatal source: 5 (RTEMS_FATAL_SOURCE_EXIT)
fatal code: 0 (0x00000000)
RTEMS version: 5.0.0.94bdde9c5daf258a8d981e63bf4180b7b249677
RTEMS tools: 9.3.0 20200312 (RTEMS 5, RSB 5 {3bd11fd4898b}, Newlib 7947581)
exuting thread ID: 0x08a010001
executing thread name: UI1
Program received signal SIGTRAP, Trace/breakpoint trap.
_CPU_Fatal_halt (source=source@entry=5, error=error@entry=0)
at /home/user/riscv/leon-rtems/build/../c/src/lib/libbsp/riscv/../../../../../../bsps/riscv/riscv/start/bsp_fatal_halt.c:43
43      asm (*ebreak*);
(gdb)
```
5. RTEMS Real Time Operating System

5.1. Overview

RTEMS is a real time operating system that supports many processor families [RD-3]. Cobham Gaisler distributes a precompiled RTEMS toolchain for NOEL-V. This section gives the reader a brief introduction on how to use RTEMS together with the NOEL-XCKU-EX design. It will be demonstrated how to install the toolchain and build an existing sample RTEMS project and run it on the board using GRMON.

The NOEL-V RTEMS distribution includes a prebuilt toolchain with GNU Binutils, GCC and Newlib. The supported host operating system is Linux. It also contains prebuilt RTEMS kernels for the NOEL-V, including RV64IMA and RV64IMAFD versions. Support is included for the NOEL-XCKU-EX interrupt controller, timer and UART.

Sample RTEMS projects are available within the toolchain package, installed in the `examples` directory.

5.2. Features

- **Kernel:**
  - BSP variants for `rv64ima` and `rv64imafd`.
  - RTEMS POSIX support
  - Based on RTEMS master as of June 2020. (Exact `rtems.git` commit hash can be found in the `README` file in the root directory of the toolchain distribution.)
- **NOEL-V BSP**
  - Console driver for APBUART
  - Interrupt controller (PLIC and CLINT)
  - Clock driver via CLINT `mtime`
  - Binary compatible with the *Spike RISC-V ISA Simulator*.
- GCC 9.3.0

5.3. Install toolchain and kernel

The toolchain and source can be downloaded from https://www.gaisler.com/noel-xcku.

First extract the toolchain and kernel archive into `/opt`. In order for the compiler to be found, the binary directory `/opt/rtems-noel-1.0.3/bin` has to be added to the `PATH` variable as below:

```bash
$ cd /opt
$ tar xf rtems-noel-1.0.3.tar.bz2
$ export PATH=$PATH:/opt/rtems-noel-1.0.3/bin
```

5.4. Building an RTEMS sample application

Once the toolchain is set up, you can compile and link a sample RTEMS application by doing:

```bash
$ cd /opt/rtems-noel-1.0.3/examples/hello
$ make
```

The default load address is at the start of the RAM, i.e. `0x40000000`.

See Chapter 7 for more information on the available examples.

5.5. Running and debugging with GRMON

Once your executable is compiled, connect to your NOEL-XCKU-EX with GRMON. The following log shows how to load and run an executable. Note that the console output is redirected to GRMON by the use of the `-u`
command line switch, so that target application console output (APBUART) is shown directly in the GRMON console.

Example 5.1.

```bash
$ grmon -digilent -u
GRMON debug monitor v3.2.3 64-bit version

grmon3> batch /opt/rtems-noel-1.0.3/grmon/noel-xilinx-kcu105.tcl

grmon3> load hello.exe
40000000 .start  76B  [=============] 100%
40000040 .text  98.6kB / 98.6kB  [=============] 100%
40018AB0 .rodata 83.5kB / 83.5kB  [=============] 100%
4002D910 .eh_frame  4B  [=============] 100%
4002D918 .init_array  8B  [=============] 100%
4002D920 .fini_array  8B  [=============] 100%
4002D928 .rtemsroset 112B  [=============] 100%
4002D998 .data  2.3kB / 2.3kB  [=============] 100%
4002E2C0 .htif  4.0kB / 4.0kB  [=============] 100%
40030000 .sdata  208B  [=============] 100%
Total size: 188.86kB (1.12Mbit/s)
Entry point 0x40000000
Image hello.exe loaded

grmon3> run
46FFFFF0 Binary data 1.6kB / 1.6kB  [=============] 100%
Total size: 1.61kB (43.11kbit/s)
Entry point 0x46ffffff
Image /opt/rtems-noel-1.0.3/grmon/noel-xilinx-kcu105.dtb loaded

hello, world

CPU 0: Forced into debug mode
0x4001607c: 00100073 ebreak <_CPU_Fatal_halt+36>

CPU 1: Interrupted!
0x40011018: 10500073 wfi <_CPU_Thread_Idle_body+0>

gomon3>

To debug the compiled program you can insert break points, step and continue directly from the GRMON console. Compilation symbols are loaded automatically by GRMON once you load the executable. An example is provided below.

gomon3> load hello.exe
[...]
Total size: 188.86kB (1.12Mbit/s)
Entry point 0x40000000
Image hello.exe loaded

gomon3> bp Init
Software breakpoint 1 at <Init>

gomon3> run

Breakpoint 1 hit
0x40000118: 1141 addi sp, sp, -16 <Init+0>

gomon3> inst 5

```
TIME  L  P  ADDRESS  INSTRUCTION           RESULT  SYMBOL
593654  1  M  4000075a  jair ra, a5  [0000000040000005] _Thread_Handler+0x4e
593658  1  M  40000768  ld  t1, 272(a0)  [0000000040000018] _Thread_Entry_adaptor_numeric+0x0
593660  0  M  40000768  ld  a0, 280(a0)  [00000000400023128] _Thread_Entry_adaptor_numeric+0x4
593660  1  M  40000768  jair zero, t1  [0000000040007668E] _Thread_Entry_adaptor_numeric+0x8
593695  0  M  40000118  addi sp, sp, -16  [ BREAKPOINT ] Init+0x0
```

gomon3> step
0x4000013c: 4002e537 lui a0, 0x4001e  <Init+0>

gomon3> inst 5

```
TIME  L  P  ADDRESS  INSTRUCTION           RESULT  SYMBOL
593660  0  M  40000768  ld  a0, 280(a0)  [00000000400023128] _Thread_Entry_adaptor_numeric+0x4
593660  1  M  40000768  jair zero, t1  [0000000040007668E] _Thread_Entry_adaptor_numeric+0x8
593695  0  M  40000118  ebreak  [ BREAKPOINT ] Init+0x0
```
593728 0 M 40000118 addi sp, sp, -16 [0000000040029D70] Init+0x0
593729 0 M 4000011a sd s0, 0(sp) [0000000040029D70] Init+0x2

grmon3> reg
a0: 000000000040023128    t0: 7F7F7F7F80800000    s0: 0000000040021A48
a1: 0000000000000000    t1: 0000000000000000    s1: 0000000000000000
a2: 0000000000000000    t2: 8080808080808080    s2: 0000000000000000
a3: 0000000000000000    t3: 0000000000000000    s3: 0000000000000000
a4: 0000000000000000    t4: 0000000000000000    s4: 0000000000000000
a5: 0000000000000000    t5: 0000000000000000    s5: 0000000000000000
a6: 00000000000021DAE500    t6: 0000000000000000    s6: 0000000000000000
a7: 00000000000021DAE500    t7: 0000000000000000    s7: 0000000000000000
sp: 0000000000000000    t8: 0000000000000000    s8: 0000000000000000
mstatus: 0000000000000000    mip: 000    mie: 800    s10: 0000000000000000
ra: 0000000000000000    t9: 0000000000000000    s9: 0000000000000000
<Thread_Handler+0x50>
ra: 0000000004000000    t0: 7F7F7F7F80800000    s0: 0000000040021A48
ra: 0000000000000000    t1: 0000000000000000    s1: 0000000000000000
ra: 0000000000000000    t2: 8080808080808080    s2: 0000000000000000
ra: 0000000000000000    t3: 0000000000000000    s3: 0000000000000000
ra: 0000000000000000    t4: 0000000000000000    s4: 0000000000000000
ra: 0000000000000000    t5: 0000000000000000    s5: 0000000000000000
ra: 00000000000021DAE500    t6: 0000000000000000    s6: 0000000000000000
ra: 00000000000021DAE500    t7: 0000000000000000    s7: 0000000000000000
sp: 0000000000000000    t8: 0000000000000000    s8: 0000000000000000
mstatus: 0000000000000000    mip: 000    mie: 800    s10: 0000000000000000
ra: 0000000000000000    t9: 0000000000000000    s9: 0000000000000000
<Thread_Handler+0x50>

grmon3> cont
hello, world
Forced into debug mode
0x4000c2ba: 9002  ebreak  <_CPU_Fatal_halt+36>

grmon3>

Alternatively you can run GRMON with the --gdb command line option and then attach a GDB session to it.

5.6. Running with the Spike RISC-V ISA Simulator

The NOEL-V BSP is compatible with Spike by means of HTIF driver support. HTIF and APBUART support on the target is determined at run-time using the device tree. Below is an example on how to use Spike to run the same binary as was previously used in the GRMON example.

Example 5.2.
$ spike -m0x40000000:0x10000000 --isa=RV64IMA hello.exe
hello, world

5.7. RISC-V and NOEL-V integration with RTEMS

5.7.1. CSRs

RTEMS RISC-V executes in machine privilege mode only. The following is the set of CSRs which are accessed by the kernel:

- mcause
- mepc
- mie
- mstatus
- mtvec

In particular, PMP is not used.

5.7.2. Clock tick

mtime is used for the RTEMS kernel clock service. It relies on the core local interrupt controller (clint).

Note that an RTEMS application can choose to not use the kernel clock service, and thus not the mtime interrupt.

5.7.3. Exceptions

The only RISC-V exception handled by RTEMS is the mtime interrupt exception. All other exceptions (interrupt and non-interrupt) will result in a kernel fatal. The fatal handler will print the current processor state and then terminate execution.
Terminating execution is performed on NOEL-V by executing an `ebreak` and on HTIF systems by generating a dedicated HTIF command. The NOEL-V RTEMS BSP determines the method to terminate at run-time if support for both HTIF is enabled (default).

5.7.4. NOEL-V BSP variants

The NOEL-V RTEMS BSP variants are similar to the RTEMS mainline BSP variants for RISC-V (rv64imac, etc) available in the kernel source tree directory `bsps/riscv/riscv/config`.

5.7.5. Console driver

NOEL-V BSP variants include support for the GRLIB APBUART device which is used as RTEMS console. The GRLIB driver `apbuart_termios.c` is used. That is, the NOEL-V BSP and the LEON BSP:s use the same console driver. Polling mode is used by default and the kernel can optionally be configured for interrupt console UART.

5.7.6. Memory layout

All NOEL-V RTEMS BSP variants link the full application to RAM. The link address is the first address of RAM: `0x40000000`. ROM is not used. MMU is not used.

5.7.7. Work area

The NOEL-V RTEMS BSP variants tries to detect the amount of RAM and sizing of the workspace (heap) at run-time. This is done by investigating the stack pointer (`sp`) at entry to kernel.

- If `sp` equals `0` at entry to the kernel, then the BSP assumes that the 12 MiB RAM is available.
- If `sp` is not equal to `0` at entry to the kernel, then the BSP assumes that `sp` points to the top of RAM.

In both cases, the workspace (heap) is configured to use all RAM space ranging from end of the image to the end of RAM.

5.7.8. Symmetric Multiprocessing

RTEMS on NOEL-V supports Symmetric Multiprocessing (SMP). This works out-of-the-box with NOEL-V and no RTEMS changes are needed for SMP, except for giving the option `--enable-smp` when configuring the kernel. The NOEL-V BSP variants in the binary kernel distribution are compiled with SMP support.

5.8. Device tree

5.8.1. Background

RTEMS relies on a device tree description of the target system to operate. It is used for locating peripheral devices and other hardware configuration. On entry to the kernel, RTEMS assumes that a pointer to the device tree is available in register `a1`. The RTEMS init code copies the device tree from the location pointed to by `a1` to a private buffer in RAM where it is later parsed during device discovery.

When building an RTEMS application with `rtems-noel-1.0.3`, a device tree is not included in the link image. The benefit of this is that the same application binary can be used on different systems.

5.8.2. GRMON

GRMON is responsible for preparing the device tree binary file (.dtb) in RAM and pointing to it with `a1`.

Preparing and defining the device tree with GRMON is easiest done using the provided GRMON Tcl script `noel-xilinx-kcu105.tcl` (Xilinx KCU105).

Then use the command `run` as normal to start an RTEMS application, or any other application expecting a device tree:

\textit{Example 5.3. load and run a RAM image which expects a device tree}

```
grmon3> batch noel-xilinx-kcu105.tcl
grmon3> load myprogram.elf
```
5.8.2.1. Details

The following describes how to manually prepare GRMON and the processor for executing an RTEMS application. It replicates the steps taken by the `run` command after it has been patched.

- The below procedure may change or may not be needed in future versions of GRMON or the RTEMS BSP.

1. Use the device tree compiler (`dtc`) to generate a device tree blob (.dtb) from a device tree source file (.dts).
   
   ```
   $ dtc the.dts > the.dtb
   ```

2. Load the device tree blob (.dtb) to the top of target RAM using the GRMON command.

   ```
   grmon3> bload the.dtb $dtbaddr
   ```

3. Set the stack pointer (`sp`) to the address of the device tree blob. This sets the upper RAM address limit for the RTEMS workspace.

   ```
   grmon3> stack $dtbaddr
   ```

4. Load the application ELF file.

   ```
   grmon3> load hello.exe
   ```

5. Initialize `a1` to the address of the device tree blob. This makes the RTEMS kernel aware of the device tree.

   ```
   grmon3> reg a1 $dtbaddr
   ```

6. Start execution with the GRMON command `run`.

   ```
   grmon3> run
   ```

5.8.3. Spike

When running an application with Spike, the simulator prepares the device tree in a simulated ROM and points to it before entering the kernel entry point in RAM. No external `.dtb` file is required.

Section 5.6 describes how to run a NOEL-V RTEMS application with Spike.

5.9. Compiler options

The build examples above use Makefile fragments available in the tool chain installation directory and is provided by the RTEMS kernel. An overview of this is given in the text file `rtems-noel-1.0.3/kernel/share/rtems5/make/README`.

All GCC compiler options are described in the GCC User's Manual. Some of the commonly used options are repeated below:

*Table 5.1. Common GCC options for rtems-noel*

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-g</code></td>
<td>generate debugging information - must be used for debugging with GDB</td>
</tr>
<tr>
<td><code>-msoft-float</code></td>
<td>emulate floating-point - must be used if no FPU exists in the system</td>
</tr>
<tr>
<td><code>-march=rv64ima</code></td>
<td>generate code with mul/div and atomic instructions</td>
</tr>
<tr>
<td><code>-O2</code> or <code>-O3</code></td>
<td>optimize code maximum performance and minimal code size</td>
</tr>
</tbody>
</table>

5.10. Building the kernel

The source code for the RTEMS NOEL-V BSPs is available in the archive named `rtems-noel-1.0.3-src.tar.bz2`.

To build the kernel, first extract the NOEL-V RTEMS kernel source archive, and then use the following commands:

*Example 5.4.*

```sh
export RISCV_CONSOLE_MAX_NS16550_DEVICES=0
```
export RISCV_ENABLE_HTIF_SUPPORT=1
export THE_KERNPREFIX=/opt/my-rtems-noel

cd <kernel-source-dir>
./bootstrap
mkdir -p build
cd build

.../configure \
--prefix=$THE_KERNPREFIX \ 
--target=riscv-rtems5 \ 
--enable-smp \ 
--enable-tests \ 
--enable-posix=yes \ 
--enable-rtemsbsp="noel64ima noel64imac noel64imafd noel64imafdc"
make --j 4
make install

5.10.1. RTEMS test suite

Giving the kernel configure option --enable-tests will build the RTEMS kernel test suite, consisting of over 600 tests, together with the kernel. Most tests run correctly on NOEL-V. Cobham Gaisler is currently analyzing if failing tests can be explained by general RTEMS issues, RISC-V issues in RTEMS, or because of the NOEL-V integration.

5.11. Building the tool chain

The host tools can be built using the rtems-source-builder, as described at [RD-4]. RTEMS for NOEL-V uses GCC multilibs for -march=rv64ima and -march=rv64imafd. A git patch file is included in the binary distribution which adds this. rtems-noel tools were built using this patch applied on top of rtems-source-builder. The commit hash is specified in the README of the binary tool chain distribution.

An example on how to build the tool chain is provided below.

Example 5.5.

export THE_PREFIX=/opt/my-rtems-noel
export THE_RSB_REPO="git://git.rtems.org/rtems-source-builder.git"
export THE_RSB_COMMIT=3bd11fd4
export THE_RSB_PATCHES=/opt/rtems-noel-1.0.3/0001-noel-multilibs-for-gcc-9.3.0.patch

git clone $THE_RSB_REPO rsb
pushd rsb
git checkout -b noel $THE_RSB_COMMIT
git apply $THE_RSB_PATCHES --check
git am $THE_RSB_PATCHES
popd

pushd rsb/rtems
../source-builder/sb-set-builder --prefix=$THE_PREFIX 5/rtems-riscv
popd
6. Linux

6.1. Overview

A Linux image can be easily created with the help of the Buildroot tool [RD-6]. It automatically builds a toolchain and includes a wide range of user selectable software packages. This chapter will show how to get started with Buildroot and how to load and execute a Linux image on hardware using GRMON.

Cobham Gaisler provides a BSP for NOEL-XCKU-EX which is included in the Buildroot version downloadable from https://www.gaisler.com/noel-xcku. The Buildroot BSP contains additional driver support for Linux and NOEL-V platform support for OpenSBI.

6.2. Step by step instructions

Download the Buildroot distribution with the NOEL-V BSP from https://www.gaisler.com/noel-xcku and extract it.

tar xf noelv-buildroot.tar.gz

Go into the directory:

cd noelv-buildroot

Generate the default config:

make noelv_defconfig

You can now make any changes you want to the configuration. For example, you could include additional software packages.

make xconfig

![Buildroot configuration dialog](image)

*Figure 6.1. Buildroot configuration dialog*

For a text only configuration dialog use:

make menuconfig
You can also configure the Linux kernel:

```
make linux-xconfig
or
make linux-menuconfig
```

The image is then created by running `make` (this will take a while depending on the number of software packages selected):

```
make
```

The main output will be the file `output/images/fw_payload.elf` which can be loaded onto the NOELV system using GRMON.

The dependency handling for the OpenSBI package is not working correctly. If the image does not start or does not include recent changes to the configuration, try deleting the `output/build/opensbi-*` directory and rebuild the image.

Start `grmon` with `-u -nb` to forward the UART output and not break on page faults. Then load the image:

```
grmon3> load output/images/fw_payload.elf
```

Start the image using the `run` command:

```
grmon3> run
```

```
```

You can also configure the Linux kernel:

```
make linux-xconfig
or
make linux-menuconfig
```

The image is then created by running `make` (this will take a while depending on the number of software packages selected):

```
make
```

The main output will be the file `output/images/fw_payload.elf` which can be loaded onto the NOELV system using GRMON.

The dependency handling for the OpenSBI package is not working correctly. If the image does not start or does not include recent changes to the configuration, try deleting the `output/build/opensbi-*` directory and rebuild the image.

Start `grmon` with `-u -nb` to forward the UART output and not break on page faults. Then load the image:

```
grmon3> load output/images/fw_payload.elf
```

Start the image using the `run` command:

```
grmon3> run
```

```
```

The dependency handling for the OpenSBI package is not working correctly. If the image does not start or does not include recent changes to the configuration, try deleting the `output/build/opensbi-*` directory and rebuild the image.

Start `grmon` with `-u -nb` to forward the UART output and not break on page faults. Then load the image:

```
grmon3> load output/images/fw_payload.elf
```

Start the image using the `run` command:

```
grmon3> run
```

```
```

You can also configure the Linux kernel:

```
make linux-xconfig
or
make linux-menuconfig
```

The image is then created by running `make` (this will take a while depending on the number of software packages selected):

```
make
```

The main output will be the file `output/images/fw_payload.elf` which can be loaded onto the NOELV system using GRMON.

The dependency handling for the OpenSBI package is not working correctly. If the image does not start or does not include recent changes to the configuration, try deleting the `output/build/opensbi-*` directory and rebuild the image.

Start `grmon` with `-u -nb` to forward the UART output and not break on page faults. Then load the image:

```
grmon3> load output/images/fw_payload.elf
```

Start the image using the `run` command:

```
grmon3> run
```

```
```

The dependency handling for the OpenSBI package is not working correctly. If the image does not start or does not include recent changes to the configuration, try deleting the `output/build/opensbi-*` directory and rebuild the image.

Start `grmon` with `-u -nb` to forward the UART output and not break on page faults. Then load the image:

```
grmon3> load output/images/fw_payload.elf
```

Start the image using the `run` command:

```
grmon3> run
```

```
```
Platform HART Features : RV64ADFIMSU
Platform Max HARTs     : 4
Current Hart           : 0
Firmware Base          : 0x40000000
Firmware Size          : 104 KB
Runtime SBI Version    : 0.2

PMP0: 0x0000000040000000-0x000000004001ffff (A)
PMP1: 0x0000000000000000-0x00000001ffffffff (A,R,W,X)

[    0.000000] OF: fdt: Ignoring memory range 0x40000000 - 0x40200000
[    0.000000] Linux version 5.4.23 (user@host.gaisler.com) (gcc version 8.3.0 (Buildroot 2020.02))
#3 SMP Wed Jun 24 16:09:52 CEST 2020
[    0.000000] earlycon: sbi0 at I/O port 0x0 (options '')
[    0.000000] printk: bootconsole [sbi0] enabled
[    0.000000] initrd not found or empty - disabling initrd
[    0.000000] Zone ranges:
[    0.000000]   DMA32    [mem 0x0000000040200000-0x000000007fffffff]
[    0.000000]   Normal   empty
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x0000000040200000-0x000000007fffffff]
[    0.000000] Initmem setup node 0 [mem 0x0000000040200000-0x000000007fffffff]
[    0.000000] software IO TLB: mapped [mem 0x7b1fc000-0x7f1fc000] (64MB)
[    0.000000] elf_hwcap is 0x1105
[    0.000000] percpu: Embedded 17 pages/cpu s30680 r8192 d30760 u69632
[    0.000000] Built 1 zonelists, mobility grouping on.  Total pages: 258055
[    0.000000] Kernel command line: earlycon=sbi console=ttyGR0,115200
[    0.000000] Dentry cache hash table entries: 131072 (order: 8, 1048576 bytes, linear)
[    0.000000] Inode-cache hash table entries: 65536 (order: 7, 524288 bytes, linear)
[    0.000000] Sorting __ex_table...
[    0.000000] mem auto-init: stack:off, heap alloc:off, heap free:off
[    0.000000] Memory: 951532K/1046528K available (8311K kernel code, 377K rwdata, 1927K init, 305K bss, 94996K reserved, 0K cma-reserved)
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=2, Nodes=1
[    0.000000] rcu: Hierarchical RCU implementation.
[    0.000000] rcu:     RCU restricting CPUs from NR_CPUS=8 to nr_cpu_ids=2.
[    0.000000] rcu: RCU calculated value of scheduler-enlistment delay is 25 jiffies.
[    0.000000] rcu: Adjusting geometry for rcu_fanout_leaf=16, nr_cpu_ids=2
[    0.000000] nr_irqs: 0, nr_irqs: 0, preallocated irqs: 0
[    0.000000] plic: mapped 31 interrupts with 2 handlers for 4 contexts.
[    0.000000] riscv_timer_init_dt: Registering clocksource cpuid [0] hartid [0]
[    0.000000] clocksource: Switched to clocksource riscv_clocksource
[    0.000000] NET: Registered protocol family 2
[    0.000000] TCP established hash table entries: 8192 (order: 5, 131072 bytes, linear)
[    0.000000] TCP: Hash tables configured (established 8192 bind 8192)
[    0.000000] UDP hash table entries: 512 (order: 2, 16384 bytes, linear)
[    0.000000] UDP-Lite hash table entries: 512 (order: 2, 16384 bytes, linear)
[    0.000000] rpc: Registered protocol family 1
[    0.000000] nfs4: Registered protocol family 1
[    0.000000] rpcid: Registered named UNIX socket transport module.
[    0.000000] rpc: Registered udp transport module.
[    0.000000] nfs4: Registered tcp transport module.
[    0.000000] pkcs11: PKCS#11 transport module.
[    0.000000] nfs: Registered NFSv4.1 backchannel transport module.
[    0.000000] nfs: Registered NFSv4.2 backchannel transport module.
[    0.000000] nfs: Registered NM client transport module.
[    0.000000] nfsd: Registered NFS v4.1 transparent backchannel transport module.
[    0.000000] nfsd: Registered NFS v4.2 transparent backchannel transport module.
[    0.000000] nfsd: Registered NM client transport module.
Starting syslogd: OK
Starting klogd: OK
Running sysctl: OK
Saving random seed: OK
Starting network: OK

Welcome to Buildroot
buildroot login:

You can now log into the system using root as username.
7. RTEMS Example applications

This section describes examples included in the distribution directory named `examples`.

To build an example, enter the source directory and issue `make`.

Example 7.1.

```
$ cd /opt/rtems-noel-1.0.3/examples/hello
$ make
```

Load and run the examples as described in Section 5.5.

Most of the examples described below use Make script fragments available in the tool chain installation directory and is provided by the RTEMS kernel. An overview of this is given in the text file `rtems-noel-1.0.3/kernel/share/rtems5/make/README`.

7.1. hello

Prints `hello, world` to the console. It can be used as a minimal starting point for custom applications.

Example 7.2.

```
$ cd /opt/rtems-noel-1.0.3/examples/hello
$ make
[...]
$ grmon -digilent -u
[...]
grmon3> batch /opt/rtems-noel-1.0.3/grmon/noel-xilinx-kcu105.tcl
grmon3> load hello.exe
grmon3> run
hello, world
```

7.2. tasks

Demonstrates the use of multiple tasks and the RTEMS directive `rtems_clock_get_tod()`.

Example 7.3.

```
grmon3> run

*** CLOCK TICK TEST ***
TA1  - rtems_clock_get_tod -  09:00:00   12/31/1988
TA2  - rtems_clock_get_tod -  09:00:00   12/31/1988
TA3  - rtems_clock_get_tod -  09:00:00   12/31/1988
TA1  - rtems_clock_get_tod -  09:00:04   12/31/1988
TA2  - rtems_clock_get_tod -  09:00:09   12/31/1988
TA1  - rtems_clock_get_tod -  09:00:09   12/31/1988
TA3  - rtems_clock_get_tod -  09:00:14   12/31/1988
TA1  - rtems_clock_get_tod -  09:00:14   12/31/1988
TA2  - rtems_clock_get_tod -  09:00:19   12/31/1988
TA1  - rtems_clock_get_tod -  09:00:19   12/31/1988
```

7.3. dhrystone

This directory contains the Dhrystone benchmark source code and Make script.

7.4. coremark

The CoreMark benchmark program from EEMBC. See the file `coremark/README.NOEL-V` for information on how to set the build parameters.

Build for NOEL-V by entering the coremark directory and run the script named `build.sh`.

Example 7.4.
$ ./build.sh
riscv-rtems5-gcc [...]
Link performed along with compile
md5sum -c coremark.md5
core_list_join.c: OK
core_main.c: OK
core_matrix.c: OK
core_state.c: OK
core_util.c: OK
coremark.h: OK

The output binary is named coremark.exe.

7.5. demo

An interactive terminal application which can toggle LEDs and display run-time information.

The following are some of the commands available in the examples/demo application:

- **top**: list the tasks
- **toggled N**: toggle LED N (0,1,2)
- **help**: list all available commands

See the file demo/README for more details.

Example 7.5.

```
grmon3> run

Hint: Test the commands 'toggled' and 'top'

RTEMS Shell on /dev/console. Use 'help' to list commands.
SHLL [/] # toggled
  toggled: [0][1][2]
SHLL [/] # toggled 0
  Toggling LED0
SHLL [/] # toggled 2
  Toggling LED2
SHLL [/] # task
ID   NAME     SHED PRI STATE  MODES    EVENTS WAITINFO
--------------------------------------------------------------------------------------------------
0a010001 UI1  UPD    1 EV     P:T:nA   NONE
0a010002 JOB0  UPD  120 READY  P:T:nA   NONE
0a010003 JOB1  UPD  140 TIME   P:T:nA   NONE
0a010004 SHLL  UPD  100 READY  P:T:nA   NONE

SHLL [/] # top
ENTER:Exit SPACE:Refresh S:Scroll A:All <>:Order +/-:Lines

Uptime: 1m44.420027          Period: 3.620000
Tasks:    6  Load Average:   75.699%  Load:   77.922%  Idle:   22.077%
Mem: Wks:   24K free  80K used  Heap:  239M free  10K used  88K stack

ID   | NAME      | RRPI | CPRI | TIME      | TOTAL    | CURRENT
---------------------------------^--
0xa010002 | JOB0  | 120  | 120  | 45.178679 | 43.266   | 41.197
0x0100001 | IDLE  | 255  | 255  | 25.374620 | 24.300   | 22.077
0xa010001 | UI1   | 1    | 1    | 0.033000  | 0.031    | 0.109
0xa010003 | JOB1  | 140  | 140  | 21.788981 | 20.866   | 19.396
0xa010005 | CPlt  | 100  | 100  | 9.936269  | 9.515    | 17.218
0xa010004 | SHLL  | 100  | 100  | 2.108507  | 2.019    | 0.109
```

7.6. Creating a custom application

The simplest way to create a custom application is to copy the hello directory used in the example above and modify the source code. New source code files can be added to the Makefile variable CSRCS.
8. Support

For support contact the Cobham Gaisler 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.