NOEL-XCKU-EX Quick Start Guide
# 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. Bare C cross-compiler .................................................................................. 5
      2.4.3. Linux .............................................................................................................. 6
      2.4.4. VxWorks 7 ................................................................................................. 6
      2.4.5. 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 ................................................................................... 8

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/ITAG interface ...................................... 10
      4.4.2. Connecting via the Ethernet debug interfaces ........................................ 10
      4.4.3. Connecting via the serial UART ................................................................. 10
   4.5. First steps .............................................................................................................. 10
   4.6. Connecting to the board ....................................................................................... 10
   4.7. Get system information ....................................................................................... 10
   4.8. Load a RAM application ....................................................................................... 14
   4.9. Debugging with GDB ........................................................................................... 15

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. RISC-V and NOEL-V integration with RTEMS .................................................. 19
      5.6.1. CSRs ............................................................................................................. 19
      5.6.2. Clock tick ...................................................................................................... 19
      5.6.3. Exceptions ................................................................................................... 19
      5.6.4. NOEL-V BSP variants ............................................................................... 19
      5.6.5. Console driver ........................................................................................... 20
      5.6.6. Memory layout ........................................................................................... 20
      5.6.7. Work area ................................................................................................... 20
   5.7. Device tree ............................................................................................................ 20
      5.7.1. Background ................................................................................................. 20
      5.7.2. GRMON ..................................................................................................... 20
   5.8. Compiler options ................................................................................................ 21
   5.9. Building the kernel ............................................................................................... 21
      5.9.1. RTEMS test suite ....................................................................................... 21
5.10. Building the tool chain ................................................................. 21
6. Bare-metal cross-compiler .............................................................. 23
   6.1. Overview .............................................................................. 23
   6.2. Installation ........................................................................ 23
   6.3. Compiling with NCC .............................................................. 23
   6.4. Compiler options ................................................................. 23
   6.5. Multilibs ............................................................................ 23
   6.6. Running and debugging with GRMON ....................................... 24
7. Linux .......................................................................................... 25
   7.1. Overview ............................................................................ 25
   7.2. Step by step instructions ...................................................... 25
8. RTEMS Example applications .......................................................... 29
   8.1. Basic examples .................................................................. 29
      8.1.1. hello ........................................................................... 29
      8.1.2. tasks .......................................................................... 29
      8.1.3. dhrystone ................................................................. 29
      8.1.4. coremark .................................................................. 29
      8.1.5. demo ......................................................................... 30
      8.1.6. Creating a custom application .......................................... 30
   8.2. Driver manager examples ....................................................... 30
      8.2.1. Introduction ................................................................. 30
      8.2.2. Requirements ............................................................. 31
      8.2.3. Build .......................................................................... 31
      8.2.4. Targets ....................................................................... 31
      8.2.5. Comments ................................................................. 31
      8.2.6. Limitations ................................................................. 32
9. Support ....................................................................................... 33
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
- GRMON 3.2.9 evaluation version available at https://www.gaisler.com/grmon.
- Xilinx Vivado Design Suite to program the FPGA ([RD-7]).

1.4. References

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</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 ([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 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. Bare C cross-compiler

NCC is a cross-compilation system for NOEL-V processors. It is based on the GNU compiler tools, the newlib C library and a support library for programming NOEL-V systems. The cross-compiler allows compilation of C and C++ single-threaded applications.
Chapter 6 describes how to use NCC with NOEL-XCKU-EX.

2.4.3. 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 7 for instructions on how to create a Linux image for NOEL-XCKU-EX with Buildroot.

2.4.4. VxWorks 7

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

2.4.5. 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. The bitstream folder contains several bitstreams which represent different configurations of the processor (EX1, EX2, ecc.). Select one of the bitstreams (described in [RD-1]) and follow the instructions below to program the FPGA:

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. Open a terminal in the downloaded folder and issue the following command to launch Vivado:

   ```bash
   vivado -mode tcl -notrace -source doprog.tcl
   ```

4. To program the FPGA with the selected configuration, run in the Vivado console (in this case for EX1):

   ```bash
   doprog EX1
   ```

5. Once the FPGA has been programmed, it is possible to connect to the board using GRMON, using the command:

   ```bash
   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 manager 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-micro J87 USB-JTAG connector and issue the following command:

```
grmon -digilent
```

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 serial UART

Make sure the switch SW12[1] is selecting the UART debug link. Please see GRMON User's Manual for instructions how to connect GRMON to a board using a serial UART connection. The PC is connected using a standard USB cable to serial converter) to the USB-micro J4 USB-UART connector and then starting GRMON without debug-link option (default is UART) or by specifying which PC UART using the `-uart COMPORT_NAME` command line switch. For example:

```
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.
- If the Ethernet debug link is present, one can view and change the EDCL IP using the `edcl` command as described in the GRMON User's Manual [RD-2].

4.6. Connecting to the board

In the following example a JTAG 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 - 80001f00
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: 80000000 - 80000f00
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

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 0x7fffffff
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 0x7fffffff
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: ff000000 - ff4000000
USR: 00000110
L2C: 4-ways, cachesize: 256 kbytes, mtrr: 0, AHB SPLIT support

ahbrom0  Cobham Gaisler  Generic AHB ROM
AHB: 00000000 - 00100000
32-bit ROM: 1 MB @ 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 - e02000000
4 contexts, 32 interrupt sources

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

ahbstat0  Cobham Gaisler  AHB Status Register
APB: 80000000 - 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
adev18  Cobham Gaisler Version and Revision Register
APB: 80000200 - 80000300
gptimer0  Cobham Gaisler Modular Timer Unit
APB: 80000300 - 80000400
IRQ: 2
16-bit scalar, 2 * 32-bit timers, divisor 80
groen3> info reg
GR Ethernet MAC
0x80000500  Control register 0x9d000000
0x80000504  Status register 0x11000000
0x80000508  MAC address MSB 0x00000000
0x8000050c  MAC address LSB 0x00000000
0x80000510  MDIO register 0x7949384a
0x80000514  Tx descriptor register 0x00000000
0x80000518  Rx descriptor register 0x00000000
0x8000051c  EDCL IP register 0xc0a800de
0x80000520  Hash table MSB register 0x00000000
0x80000524  Hash table LSB register 0x00000000
0x80000528  ECDL MAC address MSB 0x00000200
0x8000052c  ECDL MAC address LSB 0x00000009
NOEL-V RISC-V Processor
ISA and extensions 0x8000000000141101
Machine status register (mstatus) 0x0000000a00000000
Machine interrupt pending (mip) 0x0000000000000000
NOEL-V RISC-V Processor
ISA and extensions 0x8000000000141101
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 0x00010001
0x90000044  Debug Module Status 0x00030382
0x90000048  Hart Info 0x00220000
0x90000050  Hart Array Window Select 0x00000000
0x90000054  Hart Array Window 0x00000003
0x90000058  Abstract Control and Status 0x08000004
0x9000005c  Abstract Command 0x00320344
0x90000074  Next Debug Module 0x00000000
0x9000007c  Custom Features 0x00000000
0x90000080  Program Buffer 0 0x0000100f
0x90000084  Program Buffer 1 0x00100073
0x90000088  Program Buffer 2 0x10c2a283
0x9000008c  Program Buffer 3 0x00100073
0x90000090  Program Buffer 4 0x00100073
0x90000094  Program Buffer 5 0x00100073
0x90000098  Program Buffer 6 0x00100073
0x9000009c  Program Buffer 7 0x00100073
0x900000c0  Authentication Data 0x00000000
0x900000c8  Debug Module Control and Status 2 0x00000000
0x90000100  Halt Summary 0 0x00000003
0x9000010c  Custom Features 0 0x00000000
0x90000114  Custom Features 1 0x00000000
0x90000118  Custom Features 2 0x00000000
0x9000011c  Custom Features 3 0x00000000
0x90000120  Custom Features 4 0x00000000
0x90000124  Custom Features 5 0x00000000
0x90000128  Custom Features 6 0x00000000
0x9000012c  Custom Features 7 0x00000000
0x90000130  Custom Features 8 0x00000000
0x90000134  Custom Features 9 0x00000000
0x90000138  Custom Features 10 0x00000000
0x9000013c  Custom Features 11 0x00000000
0x90000140  Custom Features 12 0x00000000
0x90000144  Custom Features 13 0x00000000
0x90000148  Custom Features 14 0x00000000
0x9000014c  Custom Features 15 0x00000000
AMBA Trace Buffer
0xfff00000  Trace buffer control register 0x00000800
0xfff00004  Trace buffer index register 0x000007f0
0xfff00008  Trace buffer time tag counter 0x00000000
0xfff0000c  Trace buffer ms/slv filter register 0x00000000
0xfff00010 Trace buffer bp 1 address 0x00000000
0xfff00014 Trace buffer bp 1 mask 0x00000000
0xfff00018 Trace buffer bp 2 address 0x00000000
0xfff0001c Trace buffer bp 2 mask 0x00000000
L2-Cache Controller
0xfff00000 L2C Control register 0x80040000
0xfff00004 L2C Status register 0x46002103
0xfff00020 L2C Error status/control 0x02000000
RISC-V CLINT
0xe0100000 hart0 msip 0x00000000
0xe0104000 hart0 mtimecmp lo 0xffffffff
0xe0104004 hart0 mtimecmp hi 0xffffffff
0xe0100004 hart1 msip 0x00000000
0xe0104008 hart1 mtimecmp lo 0xffffffff
0xe010400c hart1 mtimecmp hi 0xffffffff
0xe0100008 hart2 msip 0x00000000
0xe0104010 hart2 mtimecmp lo 0xffffffff
0xe0104014 hart2 mtimecmp hi 0xffffffff
0xe010000c hart3 msip 0x00000000
0xe0104018 hart3 mtimecmp lo 0xffffffff
0xe010401c hart3 mtimecmp hi 0xffffffff
0xe010bff8 mtime lo 0x00000000
0xe010bffc mtime hi 0x00000000
RISC-V PLIC
0x84000004 Interrupt source 1 priority 0x00000000
0x84000008 Interrupt source 2 priority 0x00000000
0x8400000c Interrupt source 3 priority 0x00000000
0x84000010 Interrupt source 4 priority 0x00000000
0x84000014 Interrupt source 5 priority 0x00000000
0x84000018 Interrupt source 6 priority 0x00000000
0x8400001c Interrupt source 7 priority 0x00000000
0x84000020 Interrupt source 8 priority 0x00000000
0x84000024 Interrupt source 9 priority 0x00000000
0x84000028 Interrupt source 10 priority 0x00000000
0x8400002c Interrupt source 11 priority 0x00000000
0x84000030 Interrupt source 12 priority 0x00000000
0x84000034 Interrupt source 13 priority 0x00000000
0x84000038 Interrupt source 14 priority 0x00000000
0x8400003c Interrupt source 15 priority 0x00000000
0x84000040 Interrupt source 16 priority 0x00000000
0x84000044 Interrupt source 17 priority 0x00000000
0x84000048 Interrupt source 18 priority 0x00000000
0x8400004c Interrupt source 19 priority 0x00000000
0x84000050 Interrupt source 20 priority 0x00000000
0x84000054 Interrupt source 21 priority 0x00000000
0x84000058 Interrupt source 22 priority 0x00000000
0x8400005c Interrupt source 23 priority 0x00000000
0x84000060 Interrupt source 24 priority 0x00000000
0x84000064 Interrupt source 25 priority 0x00000000
0x84000068 Interrupt source 26 priority 0x00000000
0x8400006c Interrupt source 27 priority 0x00000000
0x84000070 Interrupt source 28 priority 0x00000000
0x84000074 Interrupt source 29 priority 0x00000000
0x84000078 Interrupt source 30 priority 0x00000000
0x8400007c Interrupt source 31 priority 0x00000000
0x84000080 Interrupt source 32 priority 0x00000000
0x84000100 Interrupt Pending bit 0-31 0x00000000
0x84000104 Interrupt Pending bit 32-63 0x00000000
0x84000200 context 0 enable source 0-31 0x00000000
0x84000204 context 0 enable source 32-63 0x00000000
0x84000208 context 1 enable source 0-31 0x00000000
0x8400020c context 1 enable source 32-63 0x00000000
0x84000210 context 2 enable source 0-31 0x00000000
0x84000214 context 2 enable source 32-63 0x00000000
0x84000218 context 3 enable source 0-31 0x00000000
0x8400021c context 3 enable source 32-63 0x00000000
0x84200000 context 0 priority threshold 0x00000000
0x84200004 context 1 priority threshold 0x00000000
0x84200008 context 2 priority threshold 0x00000000
0x8420000c context 3 priority threshold 0x00000000
AHB Status Register
0x800000f0 Status register 0x00000000
0x800000f4 Failing address register 0x00000000
General Purpose I/O port
0x80000400 I/O port data register 0x00000000
0x80000404 I/O port output register 0x00000000
0x80000408 I/O port direction register 0x00000000
0x8000040c I/O interrupt mask register 0x00000000
0x80000410 I/O interrupt polarity register 0x00000000
0x80000414 I/O interrupt edge register 0x00000000
0x80000418 Capability register 0x00000000
Generic UART
0x80000010 UART Status register 0x00000000
0x80000014 UART Control register 0x80040000
0x80000018 UART Status register 0x46002103
0x80000020 UART Error status/control 0x02000000
www.cobhamaes.com/gaisler
NOEL-XCKU-EX-QSG
December 2020, Version 1.2
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.

4.8. Load a RAM application

An application linked to RAM can be loaded directly with the load command and run with the run command. The dtb command is used to load a device tree description for the board. In the example below, the file board.dtb should be changed into the name of the target board .dtb file.
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 \texttt{-gdb} flag or enter the \texttt{gdb} command in GRMON.

\begin{verbatim}
grmon3> gdb
   Started GDB service on port 2222.
\end{verbatim}

GDB is included with the RTEMS toolchain as \texttt{riscv-rtems5-gdb}:

\begin{verbatim}
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 \textit{"help"}.
Type \textit{"apropos word"} to search for commands related to \textit{"word"}.
(gdb)
\end{verbatim}

Specify the filename of the image to debug using the \texttt{file} command:

\begin{verbatim}
(gdb) file /home/user/riscv/demo/hello/hello.exe
Reading symbols from /home/user/riscv/demo/hello/hello.exe...
\end{verbatim}

Connect to GRMON using \texttt{target extended-remote}:

\begin{verbatim}
(gdb) target extended-remote :2222
Remote debugging using :2222
0x0000000000000000 in ?? ()
\end{verbatim}

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

\begin{verbatim}
(gdb) load
Loading section .start, size 0x4c lma 0x40000000
Loading section .text, size 0x132e8 lma 0x40000004c
Loading section .rodata, size 0x120a0 lma 0x40013338
Loading section .data2, 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 0x8 lma 0x40027000
Start address 0x40000000, load size 158864
Transfer rate: 62 KB/sec, 7943 bytes/write.
\end{verbatim}

RTEMS images expect register \texttt{a1} to contain a pointer to a device tree description. This can be set up with the \texttt{dtb} command provided by GRMON. Use the \texttt{mon} prefix to execute a command in GRMON and load the device tree using \texttt{dtb}:

\begin{verbatim}
(gdb) mon dtb noel-xilinx-kcu105.dtb
DTB will be loaded to the stack
\end{verbatim}

Use the \texttt{break} command to insert a breakpoint at the \textit{Init} function:
(gdb) break Init
Breakpoint 1 at 0x40000170: file test.c, line 13.

The program can now be executed using the `run` command. GDB should break the execution once the program reaches the `Init` function.

(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("\000");

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.

(gdb) cont
Continuing.

hello, world

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

*** FATAL ***
fatal source: 5 (RTEMS_FATAL_SOURCE_EXIT)
fatal code: 0 (0x00000000)
RTEMS version: 5.0.0.94bdcc95daf258a8d09b1e63bf4180b7b249677
RTEMS tools: 9.3.0 20200312 (RTEMS 5, RSB 5 {3bd1fd4899b}, Newlib 7947581)
executing 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/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 32-bit and 64-bit 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 rv32i, rv32im, rv32ima, rv32imafd, rv64im, rv64ima and rv64imafd.
  - Uni-processor and SMP kernels available.
  - RTEMS POSIX support
- **NOEL-V BSP**
  - Console driver for APBUART
  - Interrupt controller (PLIC and CLINT)
  - Clock driver via CLINT mt ime
- **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-1.0.4/bin has to be added to the PATH variable as below:

```
$ cd /opt
$ tar xf rtems-noel-1.0.4.tar.bz2
$ export PATH=$PATH:/opt/rtems-noel-1.0.4/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:

```
$ cd /opt/rtems-noel-1.0.4/examples/hello
$ make
riscv-rtems5-gcc --pipe -march=rv64ima -mabi=lp64 \ 
-B/opt/rtems-noel-1.0.4/kernel/riscv-rtems5/noel64ima/lib \ 
-socs bsp_specs -qrtems -o test.o -o test.o
```

The default load address is at the start of the RAM, i.e. 0x00000000.

See Chapter 8 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.

```
$ grmon -digilent -u
GRMON debug monitor v3.2.9 64-bit version
grmon3> dtb board.dtb
DTB will be loaded to the stack
grmon3> load hello.exe
GRMON debug monitor v3.2.9 64-bit version
Welcome to GRMON
grmon3> run
hello, world
CPU 0: Forced into debug mode
0x0001607c: 00100073  ebreak  <_CPU_Fatal_halt+36>
CPU 1: Interrupted!
0x00011018: 10500073  wfi     <_CPU_Thread_Idle_body+0>
```

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.

```
grmon3> load hello.exe
Total size: 188.86kB (1.12Mbit/s)
Entry point 0x00000000
Image hello.exe loaded
grmon3> bp Init
Software breakpoint 1 at <Init>
grmon3> run
Breakpoint 1 hit
0x00000118: 1141  addi    sp, sp, -16  <Init+0>
grmon3> inst 5
TIME   L  P  ADDRESS           INSTRUCTION        RESULT              SYMBOL
593654  1  M  0000705a  jalr    ra, a5             [000000000000705C]  _Thread_Handler+0x4e
593658  1  M  00007684  ld      t1, 272(a0)        [0000000000023128]  _Thread_Entry_adaptor_numeric+0x0
593660  0  M  00007688  ld      a0, 280(a0)        [0000000000023128]  _Thread_Entry_adaptor_numeric+0x4
593660  1  M  0000768c  jalr    zero, t1           [000000000000768E]  _Thread_Entry_adaptor_numeric+0x8
593695  0  M  00000118  addi    sp, sp, -16        [   BREAKPOINT   ]  Init+0x0
```
Alternatively you can run GRMON with the -gdb command line option and then attach a GDB session to it.

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

5.6.1. 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

5.6.2. 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 chose to not use the kernel clock service, in which case the the mtime interrupt will not be enabled.

5.6.3. Exceptions

The only RISC-V exceptions handled by RTEMS is the mtime interrupt exception and external interrupts. 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 a ebreak.

5.6.4. 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.

The full list of BSP variants provided with the tool chain is:

- noel32i
- noel32im
- noel32imafd
- noel32imafd_smp
- noel32ima_smp
BSP variants suffixed with _smp have SMP enabled in the kernel.

5.6.5. 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.

5.6.6. Memory layout

All NOEL-V RTEMS BSP variants link the full application to RAM. The link address is the first address of RAM: 0x00000000. ROM is not used. MMU or PMP is not used by RTEMS.

5.6.7. Work area

The NOEL-V RTEMS BSP variants tries to detect the amount of RAM and sizing of the workspace (heap) at runtime. 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 a total of 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. `sp` is normally initialized by GRMON when using the `run` command.

5.7. Device tree

5.7.1. 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-1.0.4`, 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.

5.7.2. 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 `dtb` command.

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

Example 5.2. load and run a RAM image which expects a device tree

```
grmon3> dtb noel-xilinx-kcu105.dtb
grmon3> load myprogram.elf
grmon3> run
```

5.7.2.1. 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 (.).
2. Tell GRMON about the device tree blob .dtb file with the dtb command.
   `grmon3> dtb the.dtb`

3. Load the application ELF file.
   `grmon3> load hello.exe`

4. Start execution with the GRMON command run.
   `grmon3> run`

### 5.8. 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-1.0.4/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**

- `-g` generate debugging information - must be used for debugging with GDB
- `-msoft-float` emulate floating-point - must be used if no FPU exists in the system
- `-march=rv64ima` generate code with mul/div and atomic instructions
- `-O2` or `-O3` optimize code maximum performance and minimal code size

### 5.9. Building the kernel

The source code for the RTEMS NOEL-V BSPs is available in the archive named `rtems-noel-1.0.4-src.tar.bz2`.

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

**Example 5.3.**

```bash
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="noel32imafd_smp noel64imafd_smp" 
made -j 4
make install
```

#### 5.9.1. 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.

### 5.10. Building the tool chain

The host tools can be built using the rtems-source-builder, as described at [RD-4]. A git patch file is included in the binary distribution which adds additional multilibs. 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 5.4.

```bash
export THE_PREFIX=/opt/my-rtems-noel
export THE_RSB_REPO="git://git.rtems.org/rtems-source-builder.git"
export THE_RSB_COMMIT=5.1
export THE_RSB_PATCHES=/opt/rtems-noel-1.0.4/0001-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. Bare-metal cross-compiler

6.1. Overview

The Bare C Cross-Compiler (NCC) is a GNU-based cross-compilation system for NOEL-V processors. It allows cross-compilation of C and C++ single-threaded applications. This section gives the reader a brief introduction on how to use NCC together with the NOEL-XCKU-EX design. It will be demonstrated how to build an an example program and run it on the NOEL-XCKU-EX using GRMON.

The NCC toolchain includes the GNU C/C++ cross-compiler 10.2.0, GNU Binutils, Newlib embedded C library, the Bare-C run-time system with NOEL-V support and the GNU debugger (GDB). The toolchain can be downloaded from [RD-1] and is available for Linux host.

6.2. Installation

Extract the toolchain and add the bin directory to PATH. For example:

```
$ cd /opt
$ tar xf ncc-1.0.0-gcc.tar.bz2
$ PATH=$PATH:/opt/ncc-1.0.0-gcc/bin
```

The rest of this chapter assumes that the toolchain has been installed and that riscv-gaisler-elf-gcc is available in the PATH environment variable.

6.3. Compiling with NCC

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

```
$ cat hello.c
#include <stdio.h>
int main(void)
{
    printf("hello, world\n");
    return 0;
}
$ riscv-gaisler-elf-gcc -O2 -g hello.c -o hello.elf
```

It creates a program with the following characteristics:

- RV64IM instruction set
- linked to address 0.
- UART, TIMER, PLIC is probed using AMBA Plug&Play.

To build the same example targeting a 32-bit NOEL-V processor, use:

```
$ riscv-gaisler-elf-gcc -march=rv32im -mabi=ilp32 -O2 -g hello.c -o hello.elf
```

6.4. Compiler options

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-g</td>
<td>generate debugging information - recommended for debugging with GDB</td>
</tr>
<tr>
<td>-march</td>
<td>Selects instruction set for code generation. See Section 6.5.</td>
</tr>
<tr>
<td>-mabi</td>
<td>Select ABI.</td>
</tr>
<tr>
<td>-O2</td>
<td>optimize for speed</td>
</tr>
<tr>
<td>-Os</td>
<td>optimize for size</td>
</tr>
<tr>
<td>-Og</td>
<td>optimize for debugging experience</td>
</tr>
</tbody>
</table>

6.5. Multilibs

The available multilibs are:
Directory             Options
------------------------------------------------------
rv32i/ilp32           -march=rv32i     -mabi=ilp32
rv32im/ilp32          -march=rv32im    -mabi=ilp32
rv32ima/ilp32         -march=rv32ima   -mabi=ilp32
rv32imafd/ilp32d      -march=rv32imafd -mabi=ilp32d
rv64ima/lp64          -march=rv64ima   -mabi=lp64
rv64imafd/lp64d       -march=rv64imafd -mabi=lp64d
------------------------------------------------------
The default multilib when (no march or mabi) is used corresponds to -march=rv64im -mabi=ilp64.

6.6. Running and debugging with GRMON

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

$ grmon -ftdi -u
GRMON3 LEON debug monitor v3.2.9 professional 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

grmon3> load hello.elf
00000000 .text                     23.6kB /  23.6kB   [===============>] 100%
00005E70 .data                      2.7kB /   2.7kB   [===============>] 100%
Total size: 26.29kB (803.58kbit/s)
Enter point 0x00000000
Image hello.elf loaded

grmon3> run
hello, world

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

grmon3> load hello.elf
00000000 .text                     23.6kB /  23.6kB   [===============>] 100%
00005E70 .data                      2.7kB /   2.7kB   [===============>] 100%
Total size: 26.29kB (806.59kbit/s)
Enter point 0x00000000
Image hello.elf loaded

grmon3> bp main
Software breakpoint 1 at <main>

grmon3> run
CPU 0:  Program exited normally.

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

7.1. 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.

7.2. 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 noel-buildroot.tar.gz

Go into the directory:

cd noel-buildroot

Generate the default config:

make noel32_defconfig

or

make noel64_defconfig

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

make xconfig
For a text only configuration dialog use:

make menuconfig

You can also configure the Linux kernel:

make linux-xconfig

or

make linux-menuconfig

![Linux kernel configuration dialog](image)

**Figure 7.2. Linux kernel configuration dialog**

See the Buildroot user manual for more information on how to configure your system (https://buildroot.org)

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 NOEL 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
```

```
40000000 .text 58.5KB / 58.5KB [===============>] 100%
4000FF00 .rodata 2.9KB / 2.9KB [===============>] 100%
40010000 .data 512B [===============>] 100%
40200000 .payload 12.7MB / 12.7MB [===============>] 100%
Total size: 12.80MB (97.78Mbit/s)
Entry point 0x40000000
Image noelv-buildroot/output/images/fw_payload.elf loaded
```

Load the DTB using the `dtb` command:

```
grmon3> dtb noel-xilinx-kcu105.dtb
DTB will be loaded to the stack
```
Start the image using the `run` command:
```
grmon3> run
```

**OpenSBI v0.8**

```
Platform Name : noel-xilinx-kcu105
Platform Features : timer, mfdeleg
Platform HART Count : 4
Boot HART ID : 0
Boot HART ISA : rv64imafdsu
BOOT HART Features : pmp, scounteren, mcounteren
BOOT HART PMP Count : 16
Firmware Base : 0x40000000
Firmware Size : 140 KB
Runtime SBI Version : 0.2
```

```
MIDELEG : 0x0000000000000222
MEDELEG : 0x000000000000b109
```

```
Linux version 5.7.19
(gcc version 9.3.0 (Buildroot 2020.08-6-gb7b5a7c2d6),
GNU ld (GNU Binutils) 2.33.1) #15 SMP Tue Oct 27 15:10:53 CET 2020
```

```
[    0.000000] OF: fdt: Ignoring memory range 0x40000000 - 0x40200000
[    0.000000] printk: bootconsole [sbi0] enabled
[    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 0x7b1fb000-0x7f1fb000] (64MB)
[    0.000000] SBI specification v0.2 detected
[    0.000000] SBI implementation ID=0x1 Version=0x8
[    0.000000] SBI v0.2 TIME extension detected
[    0.000000] SBI v0.2 IPI extension detected
[    0.000000] SBI v0.2 RFENCE extension detected
[    0.000000] riscv: ISA extensions acim
[    0.000000] riscv: ELF capabilities acim
[    0.000000] percpu: Embedded 17 pages/cpu s31912 r8192 d29528 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] Built 1 zonelists, mobility grouping on. Total pages: 258055
[    0.000000] Virtual kernel memory layout:
[    0.000000]       fixmap : 0xffffffcefee00000 - 0xffffffceff000000   (2048 kB)
[    0.000000]       pci io : 0xffffffceff000000 - 0xffffffcf00000000   (  16 MB)
[    0.000000]      vmemmap : 0xffffffff00000000 - 0xfffffffffffffff    (4095 MB)
[    0.000000]       lowmem : 0xffffffde00000000 - 0xffffffdf00000000   (65535 MB)
[    0.000000]       cma-reserved: 0x80000000 - 0xffffffffffffffff   (16384 KB)
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=4, Nodes=1
[    0.000000] Virtual kernel memory layout:
[    0.000000]       fixmap : 0xffffffffcfe000000 - 0xffffffffcfe0000000000 (2048 KB)
[    0.000000]       pci io : 0xffffffffcfe000000 - 0xffffffffcfe00000000000 ( 16 MB)
[    0.000000]       vmemmap : 0xffffffff00000000 - 0xfffffffffffffff    (4095 MB)
[    0.000000]       lowmem : 0xffffffff00000000 - 0xffffffff00000000000 (65535 MB)
[    0.000000]       cma-reserved: 0x80000000 - 0xffffffffffffffff   (16384 KB)
```

```
[    0.000066] sched_clock: 64 bits at 50MHz, resolution 20ns, wraps every 439804651100ns
[    0.003704] Console: colour dummy device 80x25
[    0.005332] Calibrating delay loop (skipped), value calculated using timer frequency,
100.00 BogomIPS (ip)=200000
```

```
[    0.008901] pid_max: default: 32768 minimum: 301
[    0.012378] Mount-cache hash table entries: 2048 (order: 2, 16384 bytes, linear)
[    0.015177] Mountpoint-cache hash table entries: 2048 (order: 2, 16384 bytes, linear)
```

**NOEL-XCKU-EX-QSG**  
December 2020, Version 1.2  
www.cobhamaes.com/gaisler
rcu: Hierarchical SRCU implementation.
smp: Bringing up secondary CPUs ...
smp: Brought up 1 node, 4 CPUs
devtmpfs: initialized
random: get_random_u32 called from bucket_table_alloc.isra.0+0x74/0x1e4 with crng_init=0
clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 764501785100000 ns
tmpfs: initialized
clocksource: Switched to clocksource riscv_clocksource
TCP established hash table entries: 8192 (order: 4, 65536 bytes, linear)
TCP bind hash table entries: 8192 (order: 6, 262144 bytes, linear)
TCP: Hash tables configured (established 8192 bind 8192)
UDP hash table entries: 512 (order: 3, 49152 bytes, linear)
NET: Registered protocol family 1
RPC: Registered named UNIX socket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
PCI: CLS 0 bytes, default 64
workingset: timestamp_bits=62 max_order=18 bucket_order=0
NFS: Registering the id_resolver key type
Key type id_resolver registered
Key type id_legacy registered
nfs4filelayout_init: NFSv4 File Layout Driver Registering...
9p: Installing v9fs 9p2000 file system support
NET: Registered protocol family 38
Block layer SCSI generic (bsg) driver version 0.4 loaded (major 252)
io scheduler mq-deadline registered
io scheduler kyber registered
Serial: GRLIB APBUART driver
80000100.uart: ttyGR0 at MMIO 0x80000100 (irq = 2, base_baud = 625000) is a GRLIB/APBUART
printk: console [ttyGR0] enabled
printk: bootconsole [sbi0] disabled
grlib-apbuart at 0x80000100, irq 2
loop: module loaded
libphy: Fixed MDCO Bus: probed
libphy: grtech-mdio: probed
ehci-hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
ehci-pci: EHCI PCI platform driver
ehci-platform: EHCI generic platform driver
ohci-hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
ohci-pci: OHCI PCI platform driver
ohci-platform: OHCI generic platform driver
usbcore: registered new interface driver uasbus
usbcore: registered new interface driver ushbid
usbhid: USB HID core driver
NET: Registered protocol family 10
Segment Routing with IPv6
Netfilter IPv4 and IPv6 over IPv6 tunneling driver
NET: Registered protocol family 17
Safenet: Installing SAFNet 9P2000 support
Key type dns_resolved registered
Freeing unused kernel memory: 668K
Run /init as init process
random: dd: uninitialized urandom read (512 bytes read)
random: crng init done
Starting syslogd: OK
Starting klogd: OK
Starting init: OK
Welcome to Buildroot
You can now log into the system using root as username.
8. RTEMS Example applications

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

8.1. Basic examples

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

Example 8.1.

```
$ cd /opt/rtems-noel-1.0.4/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-1.0.4/kernel/share/rtems5/make/README.

8.1.1. hello

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

Example 8.2.

```
$ cd /opt/rtems-noel-1.0.4/examples/hello
$ make
[...]
[...]
grmon3> dtb noel-xilinx-kcu105.dtb
grmon3> load hello.exe
grmon3> run
hello, world
```

8.1.2. tasks

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

Example 8.3.

```
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
```

8.1.3. dhrystone

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

8.1.4. 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 8.4.
$ ./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
coremak.h: OK

The output binary is named coremark.exe.

8.1.5. 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 8.5.

```
grmon3> run

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

RTEMS Shell on /dev/console. Use 'help' to list commands.
SHLL [/] # toggledled
toggledled: [0|1|2]
SHLL [/] # toggledled 0
Toggling LED0
SHLL [/] # toggledled 2
Toggling LED2
SHLL [/] # task
```

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>SHED</th>
<th>PRI</th>
<th>STATE</th>
<th>MODES</th>
<th>EVENTS</th>
<th>WAITINFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0a010001</td>
<td>UI1</td>
<td>UPD</td>
<td>1</td>
<td>EV</td>
<td>P:T:nA</td>
<td>NONE</td>
<td></td>
</tr>
<tr>
<td>0x0a010002</td>
<td>JOB0</td>
<td>UPD</td>
<td>120</td>
<td>READY</td>
<td>P:T:nA</td>
<td>NONE</td>
<td></td>
</tr>
<tr>
<td>0x0a010003</td>
<td>JOB1</td>
<td>UPD</td>
<td>140</td>
<td>TIME</td>
<td>P:T:nA</td>
<td>NONE</td>
<td></td>
</tr>
<tr>
<td>0x0a010004</td>
<td>SHLL</td>
<td>UPD</td>
<td>100</td>
<td>READY</td>
<td>P:T:nA</td>
<td>NONE</td>
<td></td>
</tr>
</tbody>
</table>

```
SHLL [/] # top
```

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

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>RPRI</th>
<th>CPRI</th>
<th>TIME</th>
<th>TOTAL</th>
<th>CURRENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0a010002</td>
<td>JOB0</td>
<td>120</td>
<td>120</td>
<td>45.178679</td>
<td>43.266</td>
<td>41.197</td>
</tr>
<tr>
<td>0x0a010001</td>
<td>IDLE</td>
<td>255</td>
<td>255</td>
<td>25.374620</td>
<td>24.300</td>
<td>22.077</td>
</tr>
<tr>
<td>0x0a010001</td>
<td>UI1</td>
<td>1</td>
<td>1</td>
<td>0.033000</td>
<td>0.031</td>
<td>0.000</td>
</tr>
<tr>
<td>0x0a010003</td>
<td>JOB1</td>
<td>140</td>
<td>140</td>
<td>9.936269</td>
<td>9.515</td>
<td>17.218</td>
</tr>
<tr>
<td>0x0a010005</td>
<td>CPit</td>
<td>100</td>
<td>100</td>
<td>2.108507</td>
<td>2.019</td>
<td>0.109</td>
</tr>
</tbody>
</table>

8.1.6. 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.

8.2. Driver manager examples

8.2.1. Introduction

The directory examples/drvmgr contains RTEMS sample applications demonstrating how to use the RTEMS driver manager together with NOEL-V. The driver manager is compatible with 32-bit and 64-bit NOEL-V systems.
The driver manager is a device driver API software abstraction which allows developing peripheral drivers independent of bus attachment, host controller and CPU architecture. This abstraction allows sharing the same driver implementation for GRLIB peripheral among the LEON 32-bit architecture, NOEL-V 32-bit architecture and NOEL-V 64-bit architecture.

At the time of writing, the current driver manager device driver implementations are being adapted to the NOEL-V systems and is progressing together with GRLIB hardware updates for endian and bus widths.

The following driver manager device drivers are fully supported on NOEL-V:

- GRGPIO
- GPTIMER
- AHBSTAT
- GRETH (32-bit NOEL-V)

### 8.2.2. Requirements

The NOEL-V RTEMS distribution 1.0.4 is required and `riscv-rtems5-gcc` should be available in PATH.

### 8.2.3. Build

To build all examples for all targets, use:

```
make
```

The example binaries will be placed inside the `bin` directory.

### 8.2.4. Targets

Examples will be built automatically for the following BSP variants:

- Single processor BSP variants
  - noel32i
  - noel32im
  - noel32imafd
  - noelf6im
  - noelf6imafd

- Symmetric multiprocessor BSP variants:
  - noel32imafd_smp
  - noel32ima_smp
  - noelf6imafd_smp
  - noelf6ima_smp

Individual BSP:s can be selected by setting the `BPS` Make variable, for example:

```
make BPS="noel32imafd noel32im"
```

Individual examples can be built by giving the build target on the command line:

```
make rtems-gpio
```

The executables will be stored in the root samples directory. When building individual examples it is possible to control the behaviour by setting the following variables.

```
CFLAGS      - Override common compilation flags
CPUFLAGS    - Override the hardware specific compilation flags
```

Most of the samples include `config.c` which configures drivers and help setting up networking. Network interfaces are assigned a MAC and IP address according to `networkconfig.h`.

### 8.2.5. Comments

`rtems-cdtest` is a C++ application that tests exception handling.
rtems-ttcp is a network test program. It implements the receiver part of the TTCP test program. The transmission part would typically execution on your host machine.

rtems-shell is an demonstration program for the RTEMS shell. Type help at the prompt to see the available commands.

config.c configures driver resources, initializes the Driver Manager and BSP Networking Stack. config_*.c is the subsystem configurations.

### 8.2.6. Limitations

The RTEMS TCP/IP network stack provided is considered experimental for 64-bit NOEL-V.
9. 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.

There is also an open forum available at https://grlib.community.