# NOEL-PF-EX

FRONTGRADE Gaisler

USER MANUAL

RELEASED MAY 2023

# NOEL-PF-EX Quick Start Guide



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

# Table of Contents

# 1. Introduction

## 1.1. Overview

This document is a quick start guide for the NOEL-PF-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-PF-EX web page: https://www.gaisler.com/NOEL-PF.

## 1.3. Prerequisites

To use the provided bitstream, the user needs:

- PolarFire FPGA Splash Kit
- GRMON 3.2.9 evaluation version available at https://www.gaisler.com/grmon.
- Microchip FlashPro Express to program the FPGA ([RD-7]).

## 1.4. References

*Table 1.1. References*

| | |
|---|---|
| RD-1 | NOEL-PF-EX User's Manual [https://www.gaisler.com/NOEL-PF] |
| RD-2 | GRMON User's Manual [https://www.gaisler.com/doc/grmon3.pdf] |
| RD-3 | RTEMS homepage [https://www.rtems.org] |
| RD-4 | RTEMS User Manual [https://docs.rtems.org/branches/master/user/index.html] |
| RD-5 | MPF300-SPLASH-KIT User Guide [https://www.microsemi.com/existing-parts/parts/150866#resources] |
| RD-6 | Buildroot homepage [https://www.buildroot.com] |
| RD-7 | Microchip FlashPro Express [https://www.microsemi.com/product-directory/programming/4977-flashpro] |

# 2. Overview

## 2.1. Boards

The NOEL-PF-EX design can be used with the PolarFire FPGA Splash Kit ([RD-5]).

## 2.2. Design summary

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

- Frontgrade Gaisler NOEL RISC-V RV32G processor
- RISC-V Debug module
- L2 cache with 256 KiB in 4 ways
- Memory controller and 1 GiB SDRAM.
- APBUART serial interface
- GRLIB AMBA AHB bus controller
- JTAG and UART debug link
- AHB bus trace
- 10-pin GPIO controller

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

## 2.3. Processor features

- 32-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-PF-EX.

The recommended method to load software onto NOEL-PF-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-PF-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-PF-EX BSP for Buildroot which provides the necessary driver support.

See Chapter 7 for instructions on how to create a Linux image for NOEL-PF-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-PF-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-PF-EX.

# 3. Board Configuration

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

Please see MPF300-SPLASH-KIT User Guide for a detailed legend of the reference designators.

## 3.1. Debug connectors

- `J1`: USB JTAG/UART interface via FTDI with mini-USB connector. See (Chapter 4).

## 3.2. LEDs

- `LED[1..4]`: Connected to GPIO outputs [0..3] (active LOW).
- `LED[5]`: When OFF the UART interface in J1 is configured as debug link. When ON the UART interface is configured as console UART.
- `LED[6]`: When OFF indicates that the CPU is in error mode.
- `LED[7]`: When ON indicates that the memory controller calibration is complete and the FPGA design has access to the on-board SDRAM.
- `LED[8]`: When OFF indicates that the reset is asserted.

## 3.3. Push buttons

- `SW[3]`: Connected to DSUBREAK signal. Push to break software execution.
- `SW[4..6]`: Connected to GPIO inputs [4..6] .

## 3.4. DIP switches

- `DIP[1]`acts as select signal for the UART interface. When "ON" if selects the UART debug link. When "OFF" it selects the console UART.
- `DIP[2..4]`: Connected to GPIO outputs [7..9].

## 3.5. Memories

The LEON-PF-EX has 1 GiB of SDRAM available on the on-chip bus.

## 3.6. Programming the FPGA

The bitstream folder contains several FPExpress programming job files (.job) which represent different configurations of the processor (EX1,EX2,EX3 and EX4). Select one of the bitstreams (described in [RD-1]. and follow the instructions below to program the FPGA:

1. Connect the PC and the board using a standard micro-USB cable into the connector `USB-JTAG J1`.
2. Launch FlashPro Express.
3. Open a new job project and select the provided Programming Job File (.job).
4. Click on the "RUN" button and wait until the action is complete.
5. Once the FPGA has been programmed, it is possible to connect to the board through GRMON, as described in Chapter 4.

# 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-PF-EX design. After that, a basic first inspection of the board is exemplified.

GRMON is described on the homepage [https://www.gaisler.com/index.php/products/debug-tools] and in detail in the GRMON User Manual [RD-2].

## 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-PF-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. FTDI USB/JTAG interface

Please see GRMON User's Manual for how to set up the required FTDI driver software. Then connect the PC and the board using a standard USB cable into the USB-micro J1 USB-JTAG connector and issue the following command:

```
grmon -ftdi
```

### 4.4.2. 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 -digilent
```

### 4.4.3. 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
```

### 4.4.4. Connecting via the serial UART

Make sure the switch DIP[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-mini J1 USB-JTAG 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-PF-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 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 -ftdi
  GRMON debug monitor v3.2.8.3-101-gf451057 64-bit internal version

  Copyright (C) 2020 Frontgrade Gaisler - All rights reserved.
  For latest updates, go to http://www.gaisler.com/
  Comments or bug-reports to support@gaisler.com

  This internal version will expire on 27/11/2021
```

```
Parsing -ftdi

Commands missing help:

JTAG chain (1): MPF300T
  Device ID:           0x291
  GRLIB build version: 4261
  Detected frequency:  50.0 MHz

  Component                            Vendor
  NOEL-V RISC-V Processor              Frontgrade Gaisler
  AHB Debug UART                       Frontgrade Gaisler
  JTAG Debug Link                      Frontgrade Gaisler
  L2-Cache Controller                  Frontgrade Gaisler
  Generic AHB ROM                      Frontgrade Gaisler
  AHB/APB Bridge                       Frontgrade Gaisler
  RISC-V CLINT                         Frontgrade Gaisler
  RISC-V PLIC                          Frontgrade Gaisler
  AHB-to-AHB Bridge                    Frontgrade Gaisler
  RISC-V Debug Module                  Frontgrade Gaisler
  AMBA Trace Buffer                    Frontgrade Gaisler
  PolarFire FDDR4 Controller           Microsemi Corporation
  Version and Revision Register        Frontgrade Gaisler
  AHB Status Register                  Frontgrade Gaisler
  General Purpose I/O port             Frontgrade Gaisler
  Modular Timer Unit                   Frontgrade Gaisler
  Generic UART                         Frontgrade Gaisler

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

grmon3>
```

## 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      Frontgrade Gaisler  Xilinx MIG DDR3 Controller
            AHB: 40000000 - 80000000
            SDRAM: 1024 Mbyte

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

```
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> dtb board.dtb
  DTB will be loaded to the stack

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`:

```
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:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

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:

```
(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`:

```
(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.

```
(gdb) load
Loading section .start, size 0x4c lma 0x40000000
Loading section .text, size 0x132e8 lma 0x4000004c
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 .rtemsroset, 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 `dtb` command provided by GRMON. Use the `mon` prefix to execute a command in GRMON and load the device tree using `dtb`:

```
(gdb) mon dtb noel-pf.dtb
DTB will be loaded to the stack
```

Use the `break` command to insert a breakpoint at the `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("");
```

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: 5.0.0.94bddc9c5daf258a8d0981e63bf4180b7b249677
RTEMS tools: 9.3.0 20200312 (RTEMS 5, RSB 5 (3bd11fd4898b), 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]. Frontgrade 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-PF-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-PF-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
  - 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`
- GCC 9.3.0

## 5.3. Install toolchain and kernel

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

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 \
      -specs bsp_specs -qrtems -c test.c -o test.o
riscv-rtems5-gcc --pipe -march=rv64ima -mabi=lp64 \
      -B/opt/rtems-noel-1.0.4/kernel/riscv-rtems5/noel64ima/lib \
      -specs bsp_specs -qrtems -Wl,--gc-sections test.o   -o hello.exe
```

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-PF-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
          00000000 .start             76B              [===============>] 100%
          0000004C .text            98.6kB /  98.6kB   [===============>] 100%
          00018AB0 .rodata          83.5kB /  83.5kB   [===============>] 100%
          0002D8E0 .sdata2            48B              [===============>] 100%
          0002D910 .eh_frame           4B              [===============>] 100%
          0002D918 .init_array         8B              [===============>] 100%
          0002D920 .fini_array         8B              [===============>] 100%
          0002D928 .rtemsroset       112B              [===============>] 100%
          0002D998 .data             2.3kB /   2.3kB   [===============>] 100%
          0002E2C0 .htif             4.0kB /   4.0kB   [===============>] 100%
          00030000 .sdata            208B              [===============>] 100%
  Total size: 188.86kB (1.12Mbit/s)
  Entry point 0x00000000
  Image hello.exe loaded

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>

grmon3>
```

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)        [0000000000000118]  _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


grmon3> step
  0x4000013c: 4002e537  lui     a0, 0x4002e  <Init+0>


grmon3> inst 5
 TIME     L  P  ADDRESS           INSTRUCTION        RESULT              SYMBOL
  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  ebreak                     [   BREAKPOINT   ]  Init+0x0
  593728  0  M  00000118  addi    sp, sp, -16        [0000000000029D70]  Init+0x0
  593729  0  M  0000011a  sd      s0, 0(sp)          [0000000000029D70]  Init+0x2


grmon3> reg
       a0: 0000000040023128    t0: 7F7F7F7FFFFFFFFF    s0: 0000000040021A48
       a1: 0000000000000000    t1: 0000000040000118    s1: 0000000000000000
       a2: 000000004001ECC8    t2: FFFFFFFFFFFFFFFF    s2: 0000000000000000
       a3: 000000000A010001    t3: 0000000000000000    s3: 0000000000000000
       a4: 0000000000000000    t4: 0000000000000002    s4: 0000000000000000
       a5: 0000000040007684    t5: 00000000000021ED    s5: 0000000000000000
       a6: 0000000021DAE500    t6: FFFFFFFFFFFFFFF0    s6: 0000000000000000
       a7: 0000000021DAE500                            s7: 0000000000000000
```

```
                             tp: 0000000000000000    s8: 0000000000000000
       sp: 0000000000029D70    gp: 0000000000020800    s9: 0000000000000000
                                                      s10: 0000000000000000
  mstatus: 0000000A00000008   mip: 000     mie: 800   s11: 0000000000000000

       ra: 000000000000705C                           <_Thread_Handler+0x50>
       pc: 000000000000011A   sd      s0, 0(sp)       <Init+0x2>


grmon3> cont

hello, world

  Forced into debug mode
  0x0000c2ba: 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. 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 `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`
- `noel64im`
- `noel64imafd`
- `noel64imafd_smp`
- `noel64ima_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 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 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-pf.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 ().

   ```
   $ dtc board.dts > board.dtb
   ```
2. Tell GRMON about the 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.*

```
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"
make -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. Frontgrade 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 additinoal 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.*

```
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-PF-EX design. It will be demonstrated how to build an an example program and run it on the NOEL-PF-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 6.1. NCC's GCC compiler relevant options*

| | |
|---|---|
| `-g` | generate debugging information - recommended for debugging with GDB |
| `-march=` | Selects instruction set for code generation. See Section 6.5. |
| `-mabi=` | Select ABI. |
| `-O2` | optimize for speed |
| `-Os` | optimize for size |
| `-Og` | optimize for debugging experience |

## 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-PF-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 Frontgrade 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)
  Entry point 0x00000000
  Image hello.elf loaded

grmon3> run
hello, world

  CPU 0:  Program exited normally.
```

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)
  Entry point 0x00000000
  Image hello.elf loaded

grmon3> bp main
  Software breakpoint 1 at <main>

grmon3> run

  CPU 0:  breakpoint 1 hit
          0x00001928: b0102000  addi    sp, sp, -16    <main+0>

grmon3> step
  0x40001928: b0102000  mov  0, %i0  <main+4>

grmon3> step
  0x4000192c: 11100017  sethi  %hi(0x40005C00), %o0  <main+8>

grmon3> cont
hello, world

  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.

Frontgrade Gaisler provides a BSP for NOEL-PF-EX which is included in the Buildroot version downloadable from https://www.gaisler.com/NOEL-PF. 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-PF 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
```

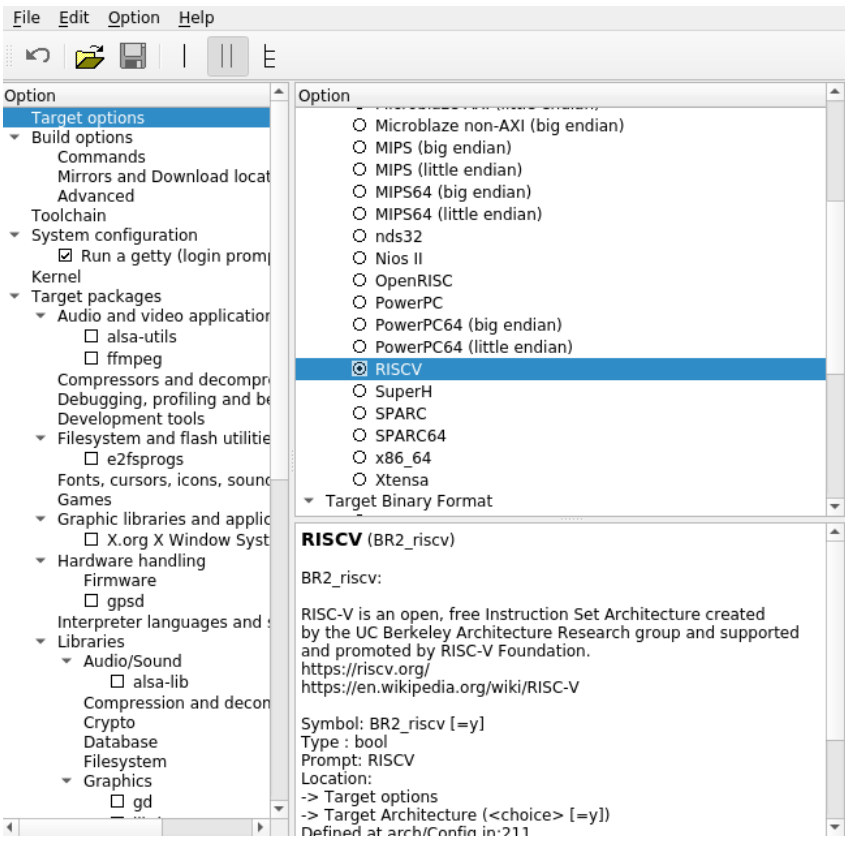


*Figure 7.1. Buildroot configuration dialog*

For a text only configuration dialog use:

```
make menuconfig
```

You can also configure the Linux kernel:

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

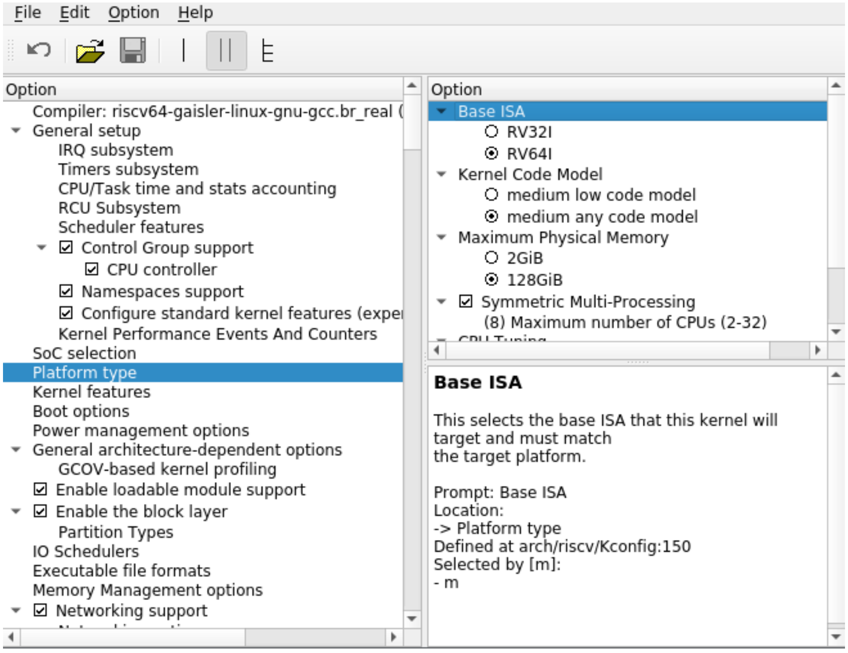


*Figure 7.2. Linux kernel configuration dialog*

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%
          4000F000 .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-pf.dtb
DTB will be loaded to the stack
```

Start the image using the `run` command:

```
grmon3> run

OpenSBI v0.8
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name       : noel-xcku-ex4
Platform Features   : timer,mfdeleg
Platform HART Count : 4
Boot HART ID        : 0
Boot HART ISA       : rv64imafdcsu
BOOT HART Features  : pmp,scounteren,mcounteren
BOOT HART PMP Count : 16
Firmware Base       : 0x0
Firmware Size       : 144 KB
Runtime SBI Version : 0.2

MIDELEG : 0x0000000000000222
MEDELEG : 0x000000000000b109
[    0.000000] Linux version 5.10.25 (@0cae9dc1dffa) (riscv64-gaisler-linux-gnu-gcc.br_real (Buildroot 2020.08-3286-g4cec7096
[    0.000000] OF: fdt: Ignoring memory range 0x0 - 0x200000
[    0.000000] earlycon: sbi0 at I/O port 0x0 (options '')
[    0.000000] printk: bootconsole [sbi0] enabled
[    0.000000] Zone ranges:
[    0.000000]   DMA32    [mem 0x0000000000200000-0x000000003fffffff]
[    0.000000]   Normal   empty
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x0000000000200000-0x000000003fffffff]
[    0.000000] Initmem setup node 0 [mem 0x0000000000200000-0x000000003fffffff]
[    0.000000] software IO TLB: mapped [mem 0x000000003b1fb000-0x000000003f1fb000] (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] SBI v0.2 HSM extension detected
[    0.000000] riscv: ISA extensions adfim
[    0.000000] riscv: ELF capabilities adfim
[    0.000000] percpu: Embedded 17 pages/cpu s32360 r8192 d29080 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: 934504K/1046528K available (9844K kernel code, 4297K rwdata, 4096K rodata, 243K init, 346K bss, 112024
[    0.000000] Virtual kernel memory layout:
[    0.000000]       fixmap : 0xffffffcefee00000 - 0xffffffceff000000   (2048 kB)
[    0.000000]       pci io : 0xffffffceff000000 - 0xffffffcf00000000   (  16 MB)
[    0.000000]      vmemmap : 0xffffffcf00000000 - 0xffffffcfffffffff   (4095 MB)
[    0.000000]      vmalloc : 0xffffffd000000000 - 0xffffffdfffffffff   (65535 MB)
[    0.000000]       lowmem : 0xffffffe000000000 - 0xffffffe03fe00000   (1022 MB)
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=4, Nodes=1
[    0.000000] rcu: Hierarchical RCU implementation.
[    0.000000] rcu:  RCU restricting CPUs from NR_CPUS=8 to nr_cpu_ids=4.
[    0.000000] rcu:  RCU debug extended QS entry/exit.
[    0.000000]  Tracing variant of Tasks RCU enabled.
[    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=4
[    0.000000] NR_IRQS: 64, nr_irqs: 64, preallocated irqs: 0
[    0.000000] riscv-intc: 64 local interrupts mapped
[    0.000000] plic: interrupt-controller@f8000000: mapped 31 interrupts with 4 handlers for 16 contexts.
[    0.000000] random: get_random_bytes called from start_kernel+0x388/0x528 with crng_init=0
[    0.000000] riscv_timer_init_dt: Registering clocksource cpuid [0] hartid [0]
[    0.000000] clocksource: riscv_clocksource: mask: 0xffffffffffffffff max_cycles: 0xb8812736b, max_idle_ns: 440795202655 ns
[    0.000088] sched_clock: 64 bits at 50MHz, resolution 20ns, wraps every 4398046511100ns
[    0.004232] Console: colour dummy device 80x25
[    0.006201] Calibrating delay loop (skipped), value calculated using timer frequency.. 100.00 BogoMIPS (lpj=200000)
[    0.010085] pid_max: default: 32768 minimum: 301
[    0.013665] Mount-cache hash table entries: 2048 (order: 2, 16384 bytes, linear)
[    0.016578] Mountpoint-cache hash table entries: 2048 (order: 2, 16384 bytes, linear)
[    0.042477] rcu: Hierarchical SRCU implementation.
[    0.053153] smp: Bringing up secondary CPUs ...
[    0.087982] smp: Brought up 1 node, 4 CPUs
[    0.099642] devtmpfs: initialized
[    0.120315] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 7645041785100000 ns
```

```
[    0.124252] futex hash table entries: 1024 (order: 4, 65536 bytes, linear)
[    0.135249] NET: Registered protocol family 16
[    0.383759] vgaarb: loaded
[    0.390060] SCSI subsystem initialized
[    0.399045] usbcore: registered new interface driver usbfs
[    0.401965] usbcore: registered new interface driver hub
[    0.404588] usbcore: registered new device driver usb
[    0.420272] clocksource: Switched to clocksource riscv_clocksource
[    0.593782] NET: Registered protocol family 2
[    0.604431] tcp_listen_portaddr_hash hash table entries: 512 (order: 2, 20480 bytes, linear)
[    0.608316] TCP established hash table entries: 8192 (order: 4, 65536 bytes, linear)
[    0.612508] TCP bind hash table entries: 8192 (order: 6, 262144 bytes, linear)
[    0.620885] TCP: Hash tables configured (established 8192 bind 8192)
[    0.627483] UDP hash table entries: 512 (order: 3, 49152 bytes, linear)
[    0.631704] UDP-Lite hash table entries: 512 (order: 3, 49152 bytes, linear)
[    0.638468] NET: Registered protocol family 1
[    0.651689] RPC: Registered named UNIX socket transport module.
[    0.654220] RPC: Registered udp transport module.
[    0.657014] RPC: Registered tcp transport module.
[    0.659604] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    0.662208] PCI: CLS 0 bytes, default 64
[    7.384815] workingset: timestamp_bits=62 max_order=18 bucket_order=0
[    7.514220] NFS: Registering the id_resolver key type
[    7.516526] Key type id_resolver registered
[    7.517905] Key type id_legacy registered
[    7.521251] nfs4filelayout_init: NFSv4 File Layout Driver Registering...
[    7.526503] fuse: init (API version 7.32)
[    7.531432] 9p: Installing v9fs 9p2000 file system support
[    7.538908] NET: Registered protocol family 38
[    7.541328] Block layer SCSI generic (bsg) driver version 0.4 loaded (major 251)
[    7.543794] io scheduler mq-deadline registered
[    7.545614] io scheduler kyber registered
[    8.568597] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
[    8.585747] Serial: GRLIB APBUART driver
[    8.589558] fc001000.uart: ttyGR0 at MMIO 0xfc001000 (irq = 2, base_baud = 6250000) is a GRLIB/APBUART
[    8.593800] printk: console [ttyGR0] enabled
[    8.593800] printk: console [ttyGR0] enabled
[    8.598150] printk: bootconsole [sbi0] disabled
[    8.598150] printk: bootconsole [sbi0] disabled
[    8.606431] grlib-apbuart at 0xfc001000, irq 2
[    8.617301] [drm] radeon kernel modesetting enabled.
[    8.804665] loop: module loaded
[    8.819065] libphy: Fixed MDIO Bus: probed
[    8.832895] vcan: Virtual CAN interface driver
[    8.836297] CAN device driver interface
[    9.038755] libphy: greth-mdio: probed
[   12.101012] e1000e: Intel(R) PRO/1000 Network Driver
[   12.160500] e1000e: Copyright(c) 1999 - 2015 Intel Corporation.
[   12.171283] ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
[   12.180419] ehci-pci: EHCI PCI platform driver
[   12.188829] ehci-platform: EHCI generic platform driver
[   12.193589] ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
[   12.198158] ohci-pci: OHCI PCI platform driver
[   12.201784] ohci-platform: OHCI generic platform driver
[   12.211752] usbcore: registered new interface driver uas
[   12.217027] usbcore: registered new interface driver usb-storage
[   12.223879] mousedev: PS/2 mouse device common for all mice
[   12.236438] usbcore: registered new interface driver usbhid
[   12.240478] usbhid: USB HID core driver
[   12.254464] NET: Registered protocol family 10
[   12.275687] Segment Routing with IPv6
[   12.279385] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[   12.293971] NET: Registered protocol family 17
[   12.297454] can: controller area network core
[   12.302364] NET: Registered protocol family 29
[   12.305670] can: raw protocol
[   12.307598] can: broadcast manager protocol
[   12.310633] can: netlink gateway - max_hops=1
[   12.318451] 9pnet: Installing 9P2000 support
[   12.322446] Key type dns_resolver registered
[   12.327778] debug_vm_pgtable: [debug_vm_pgtable         ]: Validating architecture page table helpers
[   12.353613] Freeing unused kernel memory: 240K
[   12.382397] Run /init as init process
[   14.247683] random: dd: uninitialized urandom read (512 bytes read)
[   17.992829] random: crng init done
Starting syslogd: OK
Starting klogd: OK
Running sysctl: OK
Saving random seed: OK
Starting network: OK
Starting sshd: OK

Welcome to Buildroot
buildroot login:
```

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
[...]
$ grmon -digilent -u
[...]
grmon3> dtb noel-pf.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
coremark.h: OK
```

The output binary is named `coremark.exe`.

### 8.1.5. 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
  - noel64im
  - noel64imafd
- Symmetric multiprocessor BSP variants:
  - noel32imafd_smp
  - noel32ima_smp

- noel64imafd_smp
- noel64ima_smp

Individual BSP:s can be selected by setting the BSPS 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 demonstraion program for the RTEMS shell. Type `help` at the prompt to see the available commands.

`config.c` onfigures 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 Frontgrade 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.

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