LEON3FT Microcontroller

GR716-MINI

GR716-MINI Quick Start Guide

USER MANUAL

RELEASED SEPTEMBER 2023

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 GR716 MINI Development Board.

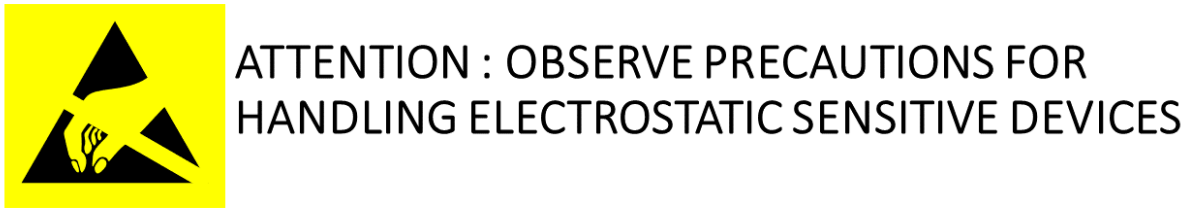
The purpose of this document is to get users quickly started using the board.

For a complete description of the board please refer to the GR716-MINI Development Board User's Manual.

The GR716 system-on-chip is described in the GR716 Data sheet and User's Manual.

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

1.2. Handling



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

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

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

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

1.3. References

Table 1.1. References

RD-1	GR716-MINI Development Board User's Manual
RD-2	GR716 Data sheet and User's Manual [https://www.gaisler.com/doc/gr716/GR716-DS-UM.pdf]
RD-4	GRMON User's Manual [https://www.gaisler.com/doc/grmon3.pdf]
RD-5	TSIM User's Manual [https://gaisler.com/index.php/products/simulators]
RD-10	Bare C Cross-Compilation System [https://www.gaisler.com/index.php/products/operating-systems/bcc]
RD-11	BCC User's Manual [https://www.gaisler.com/doc/bcc2.pdf]

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

2. Overview

The GR716 MINI Development Board provides a comprehensive and rapid software prototyping platform for the GR716 LEON3FT Microcontroller. The miniature 80 pin Expansion connector (bottom side) allow for easy expansion, accessibility and integration. Along with the microcontroller, the subject board supports the following features:

- Processor
  - GR716 microcontroller in CQFP132 Package
- Memory
  - 256 Mbit SPI flash
  - 16 Mbit SRAM
- Power Supply
  - On-board regulator converting from USB 5V to 3.3V
  - Options for single supply operation or individual external supply
- On board 25 MHz crystal
- Interfaces
  - 4 x MMCX coax connectors for 4x ADC or 4x DAC (or combination)
  - Miniature 80 pin Expansion connector (bottom side) with connections for
    - Reset
    - External SpaceWire and system clocks
    - Vref_In and Vref_Buf_out
    - 3 x LVDS RX + 3 x LVDS TX pairs
    - Analog 4 x DAC + 8 x ADC (or use as standard GPIO)
    - 12 GPIO
    - Enables prototyping of SpaceWire, SPI4S, CAN, MIL-1553, PWM, SPI Master/Slave, I2C Master/Slave and UART
  - Debug USB interface via FTDI FT4232 using GRMON3

The board has the dimension of 35 x 50 mm.

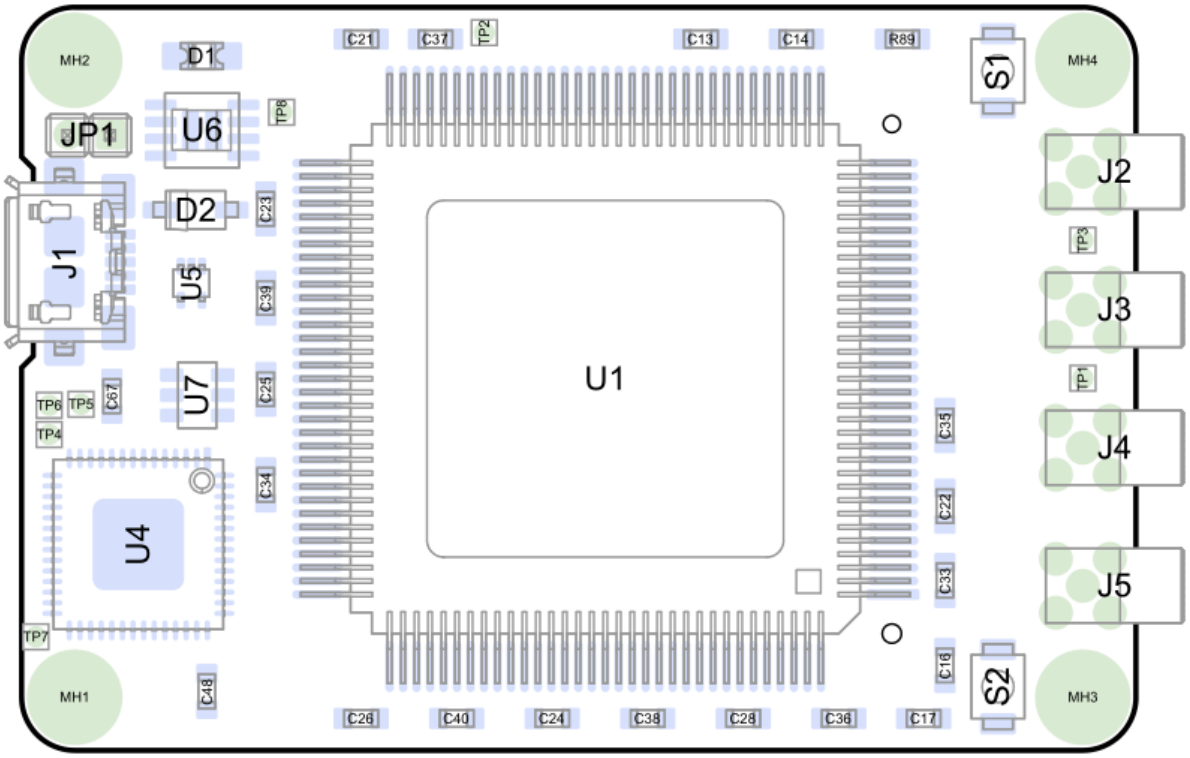


Figure 2.1. GR716-MINI top

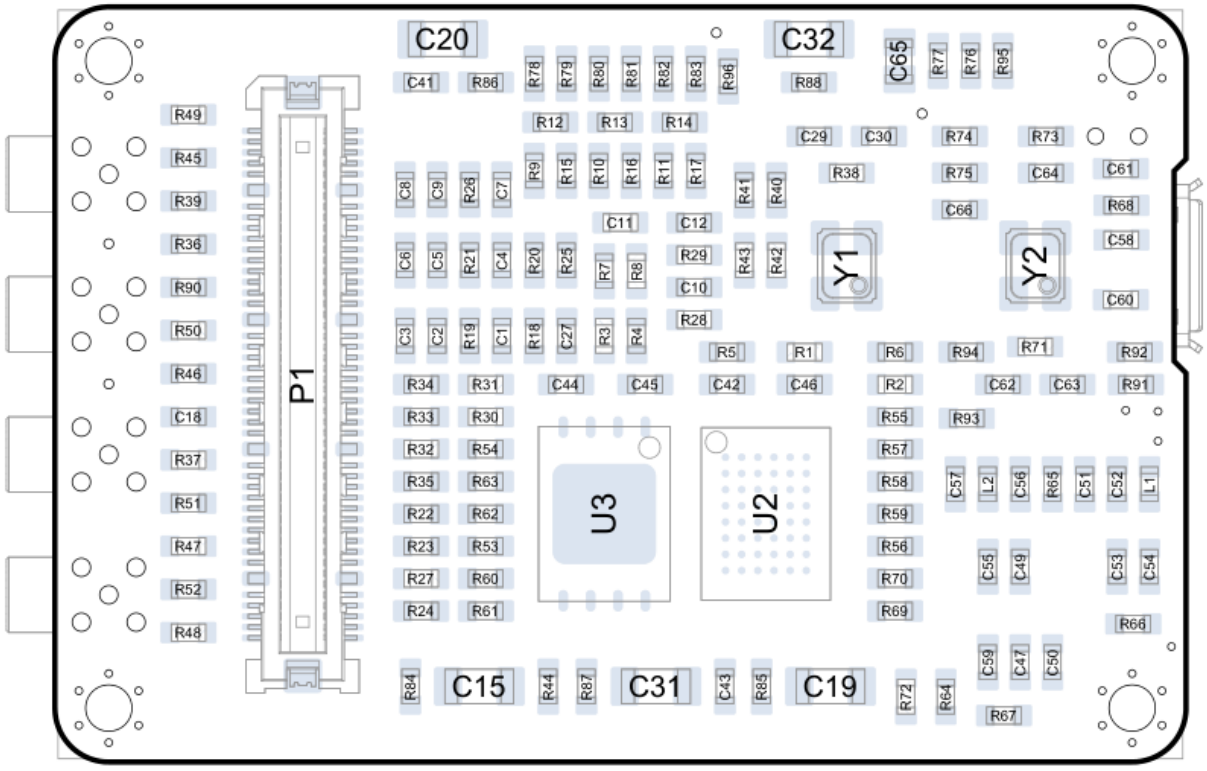


Figure 2.2. GR716-MINI bottom

Table 2.1. Major board items

Item	Description	Section
U1	GR716	[RD-2]
S1	Reset button	3.5
S2	DSU break button	3.6
JP1	Power supply select	3.3
J1	UART to USB Connector	5.2.1
J2, J3	Analog connectors ADC0 and ADC1	3.9, 8.5.1
J4, J5	Analog connectors DAC0 and DAC1	3.9,
D1	USB Power LED	3.3
P1	Mezzanine connector	9, [RD-1]
U3	32 MiB SPI flash	8.6.3
U2	2 MiB SRAM	8.5.3

3. Board Configuration

3.1. Overview

The primary sources of information for board configuration is the GR716-MINI Development Board User's Manual ([RD-1] and the GR716 Data sheet and User's Manual ([RD-2]). Before start using the GR716-MINI, clock sources have to be installed, bootstrap signals need to be set correctly and the desired interfaces have to be enabled.

GR716 shares some of its IO signals due to a limited number of package pins. For that reason, the pin multiplexing has to be set up depending on the desired interfaces and memory configuration.

3.2. Default configuration

This guide provides a default configuration which uses

- SPI flash as boot memory
- UART over FTDI as debug link
- Power supplied via USB

The complete default configuration can be found in Appendix B and GR716-MINI Development Board User's Manual. If this is your first time using the GR716-MINI, please use this configuration as a starting point.

> Default configurations are described in boxes like this one.

3.3. Power

A single supply with a +5V (minimum) / +14V (maximum) is required to power the board. All other board voltages are derived from this input using on-board discrete power circuits. GR716 MINI Development Board can be powered via the USB connector.

> JP1 is installed by default to select supply from the USB connector.

3.3.1. Alternative power

+5V (minimum) / +14V (maximum) can be supplied via the 80-pin expansion connector. JP1 must always be open open when supplying power via the expansoin connector.

3.4. Bootstrap

Bootstrap signals configure the chip at reset and are listed in the section *Bootstrap signals* of GR716 Data sheet and User's Manual. The bootstrap information is stored in a GR716 internal volatile register with reset values sampled from chip pins. It is possible to update the register while the system is running. It is also possible to change the reset values by changing resistors on the board.

> Default configuration is to execute the on-chip boot loader and then start executing from SPI flash. SPI Flash EDAC is disabled.
>
> ---
>
> The ASW container format is not used by default.
>
> ---

3.5. Reset

> The default configuration is to use the internal Power-On-Reset functionality.

It is possible to reset the board by pressing the button S1.

3.6. Break

The default configuration is to start executing from the processor entry point on reset.

It is possible to halt processor execution by pressing the board button S2.

3.7. Clock

Default configuration is to use 25 MHz system and SpaceWire clock. The clocks are derived from the on-board 25 MHz crystal.

3.8. Pin multiplexing

GR716 shares IO between memory and communication interfaces due to limited number of package pins. A complete description is given in the GR716 Data sheet and User's Manual.

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

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

Pin muxing configuration is set in the *System IO configuration* registers described in the section *Configuration Registers* in the GR716 Data sheet and User's Manual.

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

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

For sanity check, the user can validate pin configuration using the validation script described in the section *I/O switch matrix pin validation script* in the GR716 Data sheet and User's Manual. It should be noted that the script can not verify every possible configuration.

3.9. Analog I/O

GR716-MINI has connectors for analog I/O on J2, J3, J4, J5. For more information, see GR716-MINI Development Board User's Manual

4. Software Development Environment

4.1. Overview

Frontgrade Gaisler provides a comprehensive set of software tools to run several different operating systems. The GR716 platform supports the following:

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

Frontgrade Gaisler also provides a set of debug tools. The GR716 platform is supported by the following:

GRMON	Used to run and debug applications on GR716-MINI hardware. See (Chapter 5).
TSIM	Used to run and debug applications on a simulated GR716-MINI. See (Chapter 6).

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

Developer tools are generally provided for both Linux and Windows host operating systems. Frontgrade Gaisler also provides an integrated, easy-to-use solution to help programmers with the task of developing for the LEON. The LEON Integrated Development Environment for Eclipse (LIDE) is an Eclipse plug-in integrating compilers, software and hardware debuggers in a graphical user interface. The plugin makes it possible to cross-compile C and C++ application for LEON, and to debug them on either simulator and target hardware (TSIM or GRMON3).

The recommended method to load software onto a LEON board is by connecting to a debug interface of the board through the GRMON3 hardware debugger (Chapter 5). Execution of programs by a PROM-loaded boot loader is also possible.

5. GRMON3 hardware debugger

5.1. Overview

GRMON3 is a debug monitor used to develop and debug GRLIB/LEON systems. The target system, including the processor and peripherals, is accessed on the AHB bus through a debug-link connected to the host computer. GRMON3 has GDB support which makes C/C++ level debugging possible by connecting GDB to the GRMON3's GDB socket. With GRMON3 one can for example:

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

The first step is to set up a debug link in order to connect to the board. The following section outlines which debug interfaces are available and how to use them on the GR716 MINI Development Board. After that, a basic first inspection of the board is exemplified.

Several of the SoC's peripherals may be clock gated off. GRMON3 will enable all clocks if started with the flag **-cginit**. Within GRMON3, the command **grcg enable all** will have the same effect.

GRMON3 is described on the homepage [https://www.gaisler.com/index.php/products/debug-tools] and in detail in [RD-4].

It is recommended to use version GRMON3 3 or later with GR716.

5.2. Debug links

5.2.1. Connecting via the FTDI USB/UART interface

Please see GRMON User's Manual for instructions how to connect GRMON3 to a board using a serial UART connection. The PC is connected using a USB cable to the J1 connector and then starting GRMON3 with the **-uart PORTNAME** debug-link option and device name. For example:

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

It is recommended to use the GRMON3 command line option **-baud 230400** to increase the AHBUART debug link speed.

5.3. First steps

The previous sections have described which debug-links are available and how to start using them with GRMON3. The subsections below assume that GRMON3, the host computer and the GR716-MINI board have been set up so that GRMON3 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 GRMON3. With the **info sys** command more details about the system is printed and with **info reg** the register contents of the I/O registers can be inspected. Below is a list of items of particular interest:

- AMBA system frequency is printed out at connect, if the frequency is wrong then it might be due to noise in auto detection (small error). See `-freq` flag in the GRMON User's Manual [RD-4].
- Memory location and size configuration is found from the **info sys** output.
- The GR716 has a clock-gating unit which is able to disable/enable clocking and control reset signals. Clocks must be enabled for all cores that LEON software or GRMON3 will be using. The **grcg** command is described in the GRMON User's Manual [RD-4].

5.4. Connecting to the board

In the following example the UART (over USB) debug-link is used to connect to the board. The auto-detected frequency, memory parameters and stack pointer are verified by looking at the GRMON3 terminal output below.

GRMON3 is started with the **-u** and **–cginit 0x00010000** options in order to redirect UART output to the GRMON3 terminal.

```
$ grmon -u -cginit 0x00010000 -uart /dev/ttyUSB0
  GRMON LEON debug monitor v3.0.15 64-bit pro version

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

  Device ID:           0x716
  GRLIB build version: 4204
  Detected system:     GR716
  Detected frequency:  20.0 MHz

  Component                            Vendor
  AHB-to-AHB Bridge                    Frontgrade Gaisler
  MIL-STD-1553B Interface              Frontgrade Gaisler
  GRSPW2 SpaceWire Serial Link         Frontgrade Gaisler
  SPI to AHB Bridge                    Frontgrade Gaisler
  I2C to AHB Bridge                    Frontgrade Gaisler
  CAN Controller with DMA              Frontgrade Gaisler
  CAN Controller with DMA              Frontgrade Gaisler
  AHB Debug UART                       Frontgrade Gaisler
  AHB-to-AHB Bridge                    Frontgrade Gaisler
  PacketWire Receiver with DMA         Frontgrade Gaisler
  PacketWire Transmitter with DMA      Frontgrade Gaisler
  GRDMAC DMA Controller                Frontgrade Gaisler
  GRDMAC DMA Controller                Frontgrade Gaisler
  GRDMAC DMA Controller                Frontgrade Gaisler
  GRDMAC DMA Controller                Frontgrade Gaisler
  Dual-port SPI Slave                  Frontgrade Gaisler
  LEON3FT SPARC V8 Processor           Frontgrade Gaisler
  AHB-to-AHB Bridge                    Frontgrade Gaisler
  AHB Memory Scrubber                  Frontgrade Gaisler
  AHB-to-AHB Bridge                    Frontgrade Gaisler
  AHB Debug UART                       Frontgrade Gaisler
  Dual-port AHB(/CPU) On-Chip RAM      Frontgrade Gaisler
  Dual-port AHB(/CPU) On-Chip RAM      Frontgrade Gaisler
  Generic AHB ROM                      Frontgrade Gaisler
  Memory controller with EDAC          Frontgrade Gaisler
  SPI Memory Controller                Frontgrade Gaisler
  SPI Memory Controller                Frontgrade Gaisler
  AHB/APB Bridge                       Frontgrade Gaisler
  AHB/APB Bridge                       Frontgrade Gaisler
  AHB/APB Bridge                       Frontgrade Gaisler
  AHB/APB Bridge                       Frontgrade Gaisler
  Memory controller with EDAC          Frontgrade Gaisler
  LEON3 Debug Support Unit             Frontgrade Gaisler
  AHB/APB Bridge                       Frontgrade Gaisler
  AMBA Trace Buffer                    Frontgrade Gaisler
  Multi-processor Interrupt Ctrl.      Frontgrade Gaisler
  Modular Timer Unit                   Frontgrade Gaisler
  Modular Timer Unit                   Frontgrade Gaisler
  GR716 AMBA Protection unit           Frontgrade Gaisler
  Clock gating unit                    Frontgrade Gaisler
  Clock gating unit                    Frontgrade Gaisler
  General Purpose Register             Frontgrade Gaisler
  LEON3 Statistics Unit                Frontgrade Gaisler
  AHB Status Register                  Frontgrade Gaisler
  CCSDS TDP / SpaceWire I/F            Frontgrade Gaisler
  General Purpose Register Bank        Frontgrade Gaisler
  General Purpose Register             Frontgrade Gaisler
  GR716 AMBA Protection unit           Frontgrade Gaisler
  GR716 Bandgap                        Frontgrade Gaisler
  GR716 Brownout detector              Frontgrade Gaisler
  GR716 Phase-locked loop              Frontgrade Gaisler
  Generic UART                         Frontgrade Gaisler
  Generic UART                         Frontgrade Gaisler
  Generic UART                         Frontgrade Gaisler
  Generic UART                         Frontgrade Gaisler
  Generic UART                         Frontgrade Gaisler
  Generic UART                         Frontgrade Gaisler
  AHB Status Register                  Frontgrade Gaisler
  ADC / DAC Interface                  Frontgrade Gaisler
  SPI Controller                       Frontgrade Gaisler
  SPI Controller                       Frontgrade Gaisler
  PWM generator                        Frontgrade Gaisler
  General Purpose I/O port             Frontgrade Gaisler
  General Purpose I/O port             Frontgrade Gaisler
  AMBA Wrapper for OC I2C-master       Frontgrade Gaisler
  AMBA Wrapper for OC I2C-master       Frontgrade Gaisler
  GR716 Analog-to-Digital Conv         Frontgrade Gaisler
  GR716 Analog-to-Digital Conv         Frontgrade Gaisler
```

```
GR716 Analog-to-Digital Conv         Frontgrade Gaisler
GR716 Analog-to-Digital Conv         Frontgrade Gaisler
GR716 Analog-to-Digital Conv         Frontgrade Gaisler
GR716 Analog-to-Digital Conv         Frontgrade Gaisler
GR716 Analog-to-Digital Conv         Frontgrade Gaisler
GR716 Analog-to-Digital Conv         Frontgrade Gaisler
GR716 Digital-to-Analog Conv         Frontgrade Gaisler
GR716 Digital-to-Analog Conv         Frontgrade Gaisler
GR716 Digital-to-Analog Conv         Frontgrade Gaisler
GR716 Digital-to-Analog Conv         Frontgrade Gaisler
I2C Slave                            Frontgrade Gaisler
I2C Slave                            Frontgrade Gaisler
PWM generator                        Frontgrade Gaisler
LEON3 Statistics Unit                Frontgrade Gaisler
General Purpose Register             Frontgrade Gaisler

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

grmon3> info sys
ahb2ahb0  Frontgrade Gaisler  AHB-to-AHB Bridge
          AHB Master 0
          AHB: 00000000 - 20000000
          AHB: 40000000 - 60000000
          AHB: 80000000 - 90000000
          AHB: F0000000 - 00000000
gr1553b0  Frontgrade Gaisler  MIL-STD-1553B Interface
          AHB Master 1
          APB: 80101000 - 80101100
          IRQ: 4
          Device is disabled
grspw0    Frontgrade Gaisler  GRSPW2 SpaceWire Serial Link
          AHB Master 2
          APB: 80100000 - 80100100
          IRQ: 5
          Device is disabled
spi2ahb0  Frontgrade Gaisler  SPI to AHB Bridge
          AHB Master 3
          APB: 80104000 - 80104100
          IRQ: 62
          Device is disabled
i2c2ahb0  Frontgrade Gaisler  I2C to AHB Bridge
          AHB Master 4
          APB: 80105000 - 80105100
          IRQ: 47
          Device is disabled
grcan0    Frontgrade Gaisler  CAN Controller with DMA
          AHB Master 5
          APB: 80102000 - 80102400
          IRQ: 21
          Device is disabled
grcan1    Frontgrade Gaisler  CAN Controller with DMA
          AHB Master 6
          APB: 80103000 - 80103400
          IRQ: 21
          Device is disabled
ahbuart0  Frontgrade Gaisler  AHB Debug UART
          AHB Master 7
          APB: 8000F000 - 8000F100
          Device is disabled
ahb2ahb1  Frontgrade Gaisler  AHB-to-AHB Bridge
          AHB Master 8
          AHB: 90000000 - A0000000
grpwrx0   Frontgrade Gaisler  PacketWire Receiver with DMA
          AHB Master 9
          APB: 8010E000 - 8010E100
          IRQ: 2
          Device is disabled
grpwtx0   Frontgrade Gaisler  PacketWire Transmitter with DMA
          AHB Master 10
          APB: 8010F000 - 8010F100
          IRQ: 3
          Device is disabled
dma0      Frontgrade Gaisler  GRDMAC DMA Controller
          AHB Master 11
          APB: 80106000 - 80106200
          IRQ: 6
          Device is disabled
dma1      Frontgrade Gaisler  GRDMAC DMA Controller
          AHB Master 12
          APB: 80107000 - 80107200
          IRQ: 6
          Device is disabled
dma2      Frontgrade Gaisler  GRDMAC DMA Controller
          AHB Master 13
          APB: 80108000 - 80108200
```

```
          IRQ: 6
          Device is disabled
dma3      Frontgrade Gaisler  GRDMAC DMA Controller
          AHB Master 14
          APB: 80109000 - 80109200
          IRQ: 6
          Device is disabled
spislv0   Frontgrade Gaisler  Dual-port SPI Slave
          AHB Master 15
          APB: 8040F000 - 8040F100
          IRQ: 62
          Device is disabled
cpu0      Frontgrade Gaisler  LEON3FT SPARC V8 Processor
          AHB Master 0
ahb2ahb2  Frontgrade Gaisler  AHB-to-AHB Bridge
          AHB Master 1
          AHB: 30000000 - 30100000
          AHB: 90000000 - A0000000
memscrub0 Frontgrade Gaisler  AHB Memory Scrubber
          AHB Master 2
          AHB: FFF00000 - FFF00100
          IRQ: 63
          burst length: 8 bytes
ahb2ahb3  Frontgrade Gaisler  AHB-to-AHB Bridge
          AHB Master 0
          AHB: 00000000 - 80000000
          AHB: 80000000 - 90000000
          AHB: F0000000 - 00000000
ahbuart1  Frontgrade Gaisler  AHB Debug UART
          AHB Master 1
          APB: 94000000 - 94000100
          Baudrate 115200, AHB frequency 20.00 MHz
dlram0    Frontgrade Gaisler  Dual-port AHB(/CPU) On-Chip RAM
          AHB: 30000000 - 30100000
          APB: 80001000 - 80001100
          IRQ: 63
          32-bit static ram: 64 kB @ 0x30000000
ilram0    Frontgrade Gaisler  Dual-port AHB(/CPU) On-Chip RAM
          AHB: 31000000 - 31100000
          APB: 8000B000 - 8000B100
          IRQ: 63
          32-bit static ram: 128 kB @ 0x31000000
ahbrom0   Frontgrade Gaisler  Generic AHB ROM
          AHB: 00000000 - 00100000
          32-bit ROM: 1 MB @ 0x00000000
mctrl0    Frontgrade Gaisler  Memory controller with EDAC
          AHB: 01000000 - 02000000
          AHB: 40000000 - 50000000
          APB: 80000000 - 80000100
          Device is disabled
spim0     Frontgrade Gaisler  SPI Memory Controller
          AHB: FFF00100 - FFF00200
          AHB: 02000000 - 04000000
          IRQ: 52
          Device is disabled
spim1     Frontgrade Gaisler  SPI Memory Controller
          AHB: FFF00200 - FFF00300
          AHB: 04000000 - 06000000
          IRQ: 52
          Device is disabled
apbmst0   Frontgrade Gaisler  AHB/APB Bridge
          AHB: 80000000 - 80100000
apbmst1   Frontgrade Gaisler  AHB/APB Bridge
          AHB: 80100000 - 80200000
apbmst2   Frontgrade Gaisler  AHB/APB Bridge
          AHB: 80300000 - 80400000
apbmst3   Frontgrade Gaisler  AHB/APB Bridge
          AHB: 80400000 - 80500000
mctrl1    Frontgrade Gaisler  Memory controller with EDAC
          AHB: 51000000 - 52000000
          AHB: 50000000 - 51000000
          APB: 80307000 - 80307100
          Device is disabled
dsu0      Frontgrade Gaisler  LEON3 Debug Support Unit
          AHB: 90000000 - 94000000
          AHB trace: 128 lines, 32-bit bus
          CPU0:  win 31, nwp 4, itrace 128, V8 mul/div, REX, lddel 1, GRFPU-lite
                 stack pointer 0x3000fff0
                 icache not implemented
                 dcache not implemented
apbmst4   Frontgrade Gaisler  AHB/APB Bridge
          AHB: 94000000 - 94100000
ahbtrace0 Frontgrade Gaisler  AMBA Trace Buffer
          AHB: 9FF20000 - 9FF40000
          Trace buffer size: 128 lines
```

```
irqmp0    Frontgrade Gaisler  Multi-processor Interrupt Ctrl.
          APB: 80002000 - 80002400
          EIRQ: 1
gptimer0  Frontgrade Gaisler  Modular Timer Unit
          APB: 80003000 - 80003100
          IRQ: 9
          16-bit scalar, 7 * 32-bit timers, divisor 20
gptimer1  Frontgrade Gaisler  Modular Timer Unit
          APB: 80004000 - 80004100
          IRQ: 53
          16-bit scalar, 7 * 32-bit timers, divisor 20
memprot0  Frontgrade Gaisler  GR716 AMBA Protection unit
          APB: 80005000 - 80005200
          IRQ: 63
          Device is disabled
grcg0     Frontgrade Gaisler  Clock gating unit
          APB: 80006000 - 80006100
          GRMON enabled clocks during initialization
          Index for use in GRMON: 0
grcg1     Frontgrade Gaisler  Clock gating unit
          APB: 80007000 - 80007100
          GRMON enabled clocks during initialization
          Index for use in GRMON: 1
gpreg0    Frontgrade Gaisler  General Purpose Register
          APB: 80008000 - 80008100
l3stat0   Frontgrade Gaisler  LEON3 Statistics Unit
          APB: 80009000 - 80009400
          Device is disabled
ahbstat0  Frontgrade Gaisler  AHB Status Register
          APB: 8000A000 - 8000A100
          IRQ: 63
spwtdp0   Frontgrade Gaisler  CCSDS TDP / SpaceWire I/F
          APB: 8000C000 - 8000C200
          IRQ: 43
          Device is disabled
gpreg1    Frontgrade Gaisler  General Purpose Register Bank
          APB: 8000D000 - 8000D100
          IRQ: 63
gpreg2    Frontgrade Gaisler  General Purpose Register
          APB: 8000E000 - 8000E100
memprot1  Frontgrade Gaisler  GR716 AMBA Protection unit
          APB: 8010A000 - 8010A100
          IRQ: 63
bandgap0  Frontgrade Gaisler  GR716 Bandgap
          APB: 8010B000 - 8010B100
          IRQ: 63
bo0       Frontgrade Gaisler  GR716 Brownout detector
          APB: 8010C000 - 8010C100
          IRQ: 63
          Device is disabled
grpll0    Frontgrade Gaisler  GR716 Phase-locked loop
          APB: 8010D000 - 8010D100
          IRQ: 63
uart0     Frontgrade Gaisler  Generic UART
          APB: 80300000 - 80300100
          IRQ: 24
          Baudrate 38461, FIFO debug mode available
uart1     Frontgrade Gaisler  Generic UART
          APB: 80301000 - 80301100
          IRQ: 25
          Device is disabled
uart2     Frontgrade Gaisler  Generic UART
          APB: 80302000 - 80302100
          IRQ: 42
          Device is disabled
uart3     Frontgrade Gaisler  Generic UART
          APB: 80303000 - 80303100
          IRQ: 44
          Device is disabled
uart4     Frontgrade Gaisler  Generic UART
          APB: 80304000 - 80304100
          IRQ: 45
          Device is disabled
uart5     Frontgrade Gaisler  Generic UART
          APB: 80305000 - 80305100
          IRQ: 46
          Device is disabled
ahbstat1  Frontgrade Gaisler  AHB Status Register
          APB: 80306000 - 80306100
          IRQ: 63
gradcdac0 Frontgrade Gaisler  ADC / DAC Interface
          APB: 80308000 - 80308100
          IRQ: 16
spi0      Frontgrade Gaisler  SPI Controller
          APB: 80309000 - 80309100
```

```
          IRQ: 48
          Device is disabled
spi1      Frontgrade Gaisler  SPI Controller
          APB: 8030A000 - 8030A100
          IRQ: 49
          Device is disabled
grpwm0    Frontgrade Gaisler  PWM generator
          APB: 80310000 - 80310100
          IRQ: 8
          Device is disabled
gpio0     Frontgrade Gaisler  General Purpose I/O port
          APB: 8030C000 - 8030D000
          IRQ: 17
gpio1     Frontgrade Gaisler  General Purpose I/O port
          APB: 8030D000 - 8030E000
          IRQ: 38
i2cmst0   Frontgrade Gaisler  AMBA Wrapper for OC I2C-master
          APB: 8030E000 - 8030E100
          IRQ: 50
          Device is disabled
i2cmst1   Frontgrade Gaisler  AMBA Wrapper for OC I2C-master
          APB: 8030F000 - 8030F100
          IRQ: 51
          Device is disabled
adc0      Frontgrade Gaisler  GR716 Analog-to-Digital Conv
          APB: 80400000 - 80400100
          IRQ: 28
          Device is disabled
adc1      Frontgrade Gaisler  GR716 Analog-to-Digital Conv
          APB: 80401000 - 80401100
          IRQ: 29
          Device is disabled
adc2      Frontgrade Gaisler  GR716 Analog-to-Digital Conv
          APB: 80402000 - 80402100
          IRQ: 30
          Device is disabled
adc3      Frontgrade Gaisler  GR716 Analog-to-Digital Conv
          APB: 80403000 - 80403100
          IRQ: 31
          Device is disabled
adc4      Frontgrade Gaisler  GR716 Analog-to-Digital Conv
          APB: 80404000 - 80404100
          IRQ: 32
          Device is disabled
adc5      Frontgrade Gaisler  GR716 Analog-to-Digital Conv
          APB: 80405000 - 80405100
          IRQ: 33
          Device is disabled
adc6      Frontgrade Gaisler  GR716 Analog-to-Digital Conv
          APB: 80406000 - 80406100
          IRQ: 34
          Device is disabled
adc7      Frontgrade Gaisler  GR716 Analog-to-Digital Conv
          APB: 80407000 - 80407100
          IRQ: 35
          Device is disabled
dac0      Frontgrade Gaisler  GR716 Digital-to-Analog Conv
          APB: 80408000 - 80408100
          IRQ: 26
          Device is disabled
dac1      Frontgrade Gaisler  GR716 Digital-to-Analog Conv
          APB: 80409000 - 80409100
          IRQ: 27
          Device is disabled
dac2      Frontgrade Gaisler  GR716 Digital-to-Analog Conv
          APB: 8040A000 - 8040A100
          IRQ: 36
          Device is disabled
dac3      Frontgrade Gaisler  GR716 Digital-to-Analog Conv
          APB: 8040B000 - 8040B100
          IRQ: 37
          Device is disabled
i2cslv0   Frontgrade Gaisler  I2C Slave
          APB: 8040C000 - 8040C100
          IRQ: 7
i2cslv1   Frontgrade Gaisler  I2C Slave
          APB: 8040D000 - 8040D100
          IRQ: 47
grpwm1    Frontgrade Gaisler  PWM generator
          APB: 80410000 - 80410100
          IRQ: 8
          Device is disabled
l3stat1   Frontgrade Gaisler  LEON3 Statistics Unit
          APB: 94001000 - 94001400
          Device is disabled
```

```
gpreg3    Frontgrade Gaisler  General Purpose Register
          APB: 94002000 - 94002100

grmon3> info reg
AHB Memory Scrubber
    0xfff00000  AHB Status register                     0x00000000
    0xfff00004  AHB Failing address register            0x00000000
    0xfff00008  AHB Error configuration register        0x00000003
    0xfff00010  Scrubber status register                0x00000002
    0xfff00014  Scrubber configuration register         0x00000000
    0xfff00018  Scrubber 1st low address register       0x00000000
    0xfff0001c  Scrubber 1st high address register      0x00000007
    0xfff00020  Scrubber position register              0x00000000
    0xfff00024  Scrubber error threshold register       0x00000000
    0xfff00028  Scrubber initialization data register   0x00000000
    0xfff0002c  Scrubber 2nd low address register       0x00000000
    0xfff00030  Scrubber 2nd high address register      0x00000007
Dual-port AHB(/CPU) On-Chip RAM
    0x80001000  Control register                        0xf0060001
    0x80001004  Scrubber data register                  0x00000000
    0x80001008  Scrubber control 1 register             0x00000000
    0x8000100c  Scrubber control 2 register             0x0fff0000
Dual-port AHB(/CPU) On-Chip RAM
    0x8000b000  Control register                        0x90070001
    0x8000b004  Scrubber data register                  0x00000000
    0x8000b008  Scrubber control 1 register             0x00010000
    0x8000b00c  Scrubber control 2 register             0x0fff0000
LEON3 Debug Support Unit
    0x90000024  Debug mode mask register                0x00000001
    0x90000000  CPU 0 Control register                  0x000000ef
    0x90400004  CPU 0 Processor status register         0xf30010e0
    0x90400018  CPU 0 Floating-Point State Register     0x00060000
    0x90400044  CPU 0 LEON3/4 configuration register    0x06800d9e
    0x90400050  CPU 0 Alternative window register       0x001e0000
    0x90400020  CPU 0 Trap register                     0x00000000
AMBA Trace Buffer
    0x9ff20000  Trace buffer control register           0x00008000
    0x9ff20004  Trace buffer index register             0x000007f0
    0x9ff20008  Trace buffer time tag counter           0x00000000
    0x9ff2000c  Trace buffer mst/slv filter register    0x00000000
    0x9ff20010  Trace buffer bp 1 address               0x82b63248
    0x9ff20014  Trace buffer bp 1 mask                  0xb87df614
    0x9ff20018  Trace buffer bp 2 address               0xcf2d3374
    0x9ff2001c  Trace buffer bp 2 mask                  0x5fe0afb8
Multi-processor Interrupt Ctrl.
    0x80002000  Interrupt level register                0x00000000
    0x80002004  Interrupt pending register              0x00000000
    0x80002008  Interrupt force register                0x00000000
    0x80002010  Interrupt status register               0x0c010000
    0x80002018  Interrupt error mode status register    0x00000000
    0x80002040  Interrupt mask register  0              0x00000000
    0x800020c0  Interrupt extended ack. register  0     0x00000000
    0x80002240  Interrupt boot register                 0x00000000
    0x80002300  Interrupt interrupt map register 0      0x00010203
    0x80002304  Interrupt interrupt map register 1      0x04050607
    0x80002308  Interrupt interrupt map register 2      0x08090a0b
    0x8000230c  Interrupt interrupt map register 3      0x0c0d0e0f
    0x80002310  Interrupt interrupt map register 4      0x10111213
    0x80002314  Interrupt interrupt map register 5      0x14151617
    0x80002318  Interrupt interrupt map register 6      0x18191a1b
    0x8000231c  Interrupt interrupt map register 7      0x1c1d1e1f
    0x80002320  Interrupt interrupt map register 8      0x1c1d1e1f
    0x80002324  Interrupt interrupt map register 9      0x1a1b1011
    0x80002328  Interrupt interrupt map register 10     0x12130304
    0x8000232c  Interrupt interrupt map register 11     0x05060708
    0x80002330  Interrupt interrupt map register 12     0x0b0c0d0e
    0x80002334  Interrupt interrupt map register 13     0x02141516
    0x80002338  Interrupt interrupt map register 14     0x1718191a
    0x8000233c  Interrupt interrupt map register 15     0x10111213
Modular Timer Unit
    0x80003000  Scalar value register                   0x00000013
    0x80003004  Scalar reload value register            0x00000013
    0x80003008  Configuration register                  0x0000014f
    0x8000300c  Latch configuration register            0x00000000
    0x80003010  Timer 0 Value register                  0xffffffff
    0x80003014  Timer 0 Reload value register           0xffffffff
    0x80003018  Timer 0 Control register                0x00000043
    0x8000301c  Timer 0 Latch register                  0x00000000
    0x80003020  Timer 1 Value register                  0x00000000
    0x80003024  Timer 1 Reload value register           0x00000000
    0x80003028  Timer 1 Control register                0x00000040
    0x8000302c  Timer 1 Latch register                  0x00000000
    0x80003030  Timer 2 Value register                  0x00000000
    0x80003034  Timer 2 Reload value register           0x00000000
    0x80003038  Timer 2 Control register                0x00000040
```

```
    0x8000303c  Timer 2 Latch register                  0x00000000
    0x80003040  Timer 3 Value register                  0x00000000
    0x80003044  Timer 3 Reload value register           0x00000000
    0x80003048  Timer 3 Control register                0x00000040
    0x8000304c  Timer 3 Latch register                  0x00000000
    0x80003050  Timer 4 Value register                  0x00000000
    0x80003054  Timer 4 Reload value register           0x00000000
    0x80003058  Timer 4 Control register                0x00000040
    0x8000305c  Timer 4 Latch register                  0x00000000
    0x80003060  Timer 5 Value register                  0x00000000
    0x80003064  Timer 5 Reload value register           0x00000000
    0x80003068  Timer 5 Control register                0x00000040
    0x8000306c  Timer 5 Latch register                  0x00000000
    0x80003070  Timer 6 Value register                  0x0006fc49
    0x80003074  Timer 6 Reload value register           0x0006fc49
    0x80003078  Timer 6 Control register                0x00000040
    0x8000307c  Timer 6 Latch register                  0x00000000
Modular Timer Unit
    0x80004000  Scalar value register                   0x00000013
    0x80004004  Scalar reload value register            0x00000013
    0x80004008  Configuration register                  0x000001af
    0x8000400c  Latch configuration register            0x00000000
    0x80004010  Timer 0 Value register                  0xffffffff
    0x80004014  Timer 0 Reload value register           0xffffffff
    0x80004018  Timer 0 Control register                0x00000043
    0x8000401c  Timer 0 Latch register                  0x00000000
    0x80004020  Timer 1 Value register                  0x00000000
    0x80004024  Timer 1 Reload value register           0x00000000
    0x80004028  Timer 1 Control register                0x00000040
    0x8000402c  Timer 1 Latch register                  0x00000000
    0x80004030  Timer 2 Value register                  0x00000000
    0x80004034  Timer 2 Reload value register           0x00000000
    0x80004038  Timer 2 Control register                0x00000040
    0x8000403c  Timer 2 Latch register                  0x00000000
    0x80004040  Timer 3 Value register                  0x00000000
    0x80004044  Timer 3 Reload value register           0x00000000
    0x80004048  Timer 3 Control register                0x00000040
    0x8000404c  Timer 3 Latch register                  0x00000000
    0x80004050  Timer 4 Value register                  0x00000000
    0x80004054  Timer 4 Reload value register           0x00000000
    0x80004058  Timer 4 Control register                0x00000040
    0x8000405c  Timer 4 Latch register                  0x00000000
    0x80004060  Timer 5 Value register                  0x00000000
    0x80004064  Timer 5 Reload value register           0x00000000
    0x80004068  Timer 5 Control register                0x00000040
    0x8000406c  Timer 5 Latch register                  0x00000000
    0x80004070  Timer 6 Value register                  0x00000000
    0x80004074  Timer 6 Reload value register           0x00000000
    0x80004078  Timer 6 Control register                0x00000040
    0x8000407c  Timer 6 Latch register                  0x00000000
Clock gating unit
    0x80006000  Unlock register                         0x00000000
    0x80006004  Clock enable register                   0x00010000
    0x80006008  Reset register                          0x7ffeffff
Clock gating unit
    0x80007000  Unlock register                         0x00000000
    0x80007004  Clock enable register                   0x00000000
    0x80007008  Reset register                          0x003fffff
General Purpose Register
    0x80008000  Boot ROM configuration register         0x000bc001
    0x80008004  Memory test configuration register      0x00000000
AHB Status Register
    0x8000a000  Status register                         0x00000002
    0x8000a004  Failing address register                0x8000a004
General Purpose Register Bank
    0x8000d000  System IO register for GPIO 0 to 7      0x00000000
    0x8000d004  System IO register for GPIO 8 to 15     0x00000000
    0x8000d008  System IO register for GPIO 16 to 23    0x00000000
    0x8000d00c  System IO register for GPIO 24 to 31    0x00000000
    0x8000d010  System IO register for GPIO 32 to 39    0x00000000
    0x8000d014  System IO register for GPIO 40 to 47    0x00000000
    0x8000d018  System IO register for GPIO 48 to 55    0x00000000
    0x8000d01c  System IO register for GPIO 56 to 63    0x00000000
    0x8000d020  Pullup register for GPIO 0 to 31        0x00000000
    0x8000d024  Pullup register for GPIO 32 to 63       0x00000000
    0x8000d028  Pulldown register for GPIO 0 to 31      0xffffffff
    0x8000d02c  Pulldown register for GPIO 32 to 63     0xffffffff
    0x8000d030  IO configuration for LVDS               0x00888888
    0x8000d034  Configuration protection register       0x00000000
    0x8000d038  Config register error interrupt         0x00000000
    0x8000d03c  Config register error status            0x00000000
General Purpose Register
    0x8000e000  Interrupt test configuration register   0x00000000
    0x8000e004  Memory test status register             0x00000000
GR716 AMBA Protection unit
```

```
    0x8010a000  Protection Control Register             0x04000000
    0x8010a004  Segment 0 start                         0x00000000
    0x8010a008  Segment 0 end                           0x00000000
    0x8010a00c  Segment 0 Config register               0x00000000
    0x8010a014  Segment 1 start                         0x00000000
    0x8010a018  Segment 1 end                           0x00000000
    0x8010a01c  Segment 1 Config register               0x00000000
    0x8010a024  Segment 2 start                         0x00000000
    0x8010a028  Segment 2 end                           0x00000000
    0x8010a02c  Segment 2 Config register               0x00000000
    0x8010a034  Segment 3 start                         0x00000000
    0x8010a038  Segment 3 end                           0x00000000
    0x8010a03c  Segment 3 Config register               0x00000000
GR716 Phase-locked loop
    0x8010d000  PLL configuration registers             0x80000000
    0x8010d004  PLL status registers                    0x00000002
    0x8010d008  PLL clock reference registers           0x00000300
    0x8010d00c  SpaceWire clock reference registers     0x00000302
    0x8010d010  MIL-1553B clock reference registers     0x00000000
    0x8010d014  System clock reference registers        0x00000000
    0x8010d018  Select system clock source              0x00000000
    0x8010d020  Clock protection registers              0x0f002519
    0x8010d024  Test clock configuration registers      0x00000000
    0x8010d028  Select reference for PWM0 clock         0x00000000
    0x8010d02c  Select reference for PWM1 clock         0x00000000
Generic UART
    0x80300004  UART Status register                    0x0000008e
    0x80300008  UART Control register                   0x80000003
    0x8030000c  UART Scaler register                    0x00000040
AHB Status Register
    0x80306000  Status register                         0x0000000a
    0x80306004  Failing address register                0x80306004
ADC / DAC Interface
    0x80308000  Configuration register                  0x00000000
    0x80308004  Status register                         0x00000000
    0x80308010  ADC data input register                 0x00000000
    0x80308014  DAC data output register                0x00000000
    0x80308020  Address input register                  0x00000000
    0x80308024  Address output register                 0x00000000
    0x80308028  Address direction register              0x00000000
    0x80308030  Data input register                     0x00000000
    0x80308034  Data output register                    0x00000000
    0x80308038  Data direction register                 0x00000000
General Purpose I/O port
    0x8030c000  I/O port data register                  0x00080001
    0x8030c004  I/O port output register                0x00000000
    0x8030c008  I/O port direction register             0x00000000
    0x8030c00c  I/O interrupt mask register             0x00000000
    0x8030c010  I/O interrupt polarity register         0x00000000
    0x8030c014  I/O interrupt edge register             0x00000000
    0x8030c01c  Capability register                     0x0007041f
    0x8030c020  I/O interrupt map register 0            0x00000000
    0x8030c024  I/O interrupt map register 1            0x00000000
    0x8030c028  I/O interrupt map register 2            0x00000000
    0x8030c02c  I/O interrupt map register 3            0x00000000
    0x8030c030  I/O interrupt map register 4            0x00000000
    0x8030c034  I/O interrupt map register 5            0x00000000
    0x8030c038  I/O interrupt map register 6            0x00000000
    0x8030c03c  I/O interrupt map register 7            0x00000000
    0x8030c040  I/O interrupt available register        0x7fffffff
    0x8030c044  I/O interrupt flag register             0x00000000
    0x8030c048  Input enable register                   0xffffffff
    0x8030c04c  Pulse Register                          0x00000000
General Purpose I/O port
    0x8030d000  I/O port data register                  0x3f000006
    0x8030d004  I/O port output register                0x00000000
    0x8030d008  I/O port direction register             0x00000000
    0x8030d00c  I/O interrupt mask register             0x00000000
    0x8030d010  I/O interrupt polarity register         0x00000000
    0x8030d014  I/O interrupt edge register             0x00000000
    0x8030d01c  Capability register                     0x0007041f
    0x8030d020  I/O interrupt map register 0            0x00000000
    0x8030d024  I/O interrupt map register 1            0x00000000
    0x8030d028  I/O interrupt map register 2            0x00000000
    0x8030d02c  I/O interrupt map register 3            0x00000000
    0x8030d030  I/O interrupt map register 4            0x00000000
    0x8030d034  I/O interrupt map register 5            0x00000000
    0x8030d038  I/O interrupt map register 6            0x00000000
    0x8030d03c  I/O interrupt map register 7            0x00000000
    0x8030d040  I/O interrupt available register        0x7fffffff
    0x8030d044  I/O interrupt flag register             0x00000000
    0x8030d048  Input enable register                   0xffffffff
    0x8030d04c  Pulse Register                          0x00000000
I2C Slave
    0x8040c000  Slave address register                  0x80000050
```

```
    0x8040c004  Control register                        0x0000000c
    0x8040c008  Status register                         0x00000000
    0x8040c00c  Mask register                           0x00000007
    0x8040c014  Transmit register                       0x00000043
I2C Slave
    0x8040d000  Slave address register                  0x80000050
    0x8040d004  Control register                        0x00000006
    0x8040d008  Status register                         0x00000000
    0x8040d00c  Mask register                           0x00000004
    0x8040d014  Transmit register                       0x00000072
General Purpose Register
    0x94002000  Analog test config register 1           0x00000000
    0x94002004  Analog test config register 2           0x00000000
```

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

```
grmon3> info sys uart0 gptimer0
uart0     Frontgrade Gaisler  Generic UART
          APB: 80300000 - 80300100
          IRQ: 24
          Baudrate 38461, FIFO debug mode available
gptimer0  Frontgrade Gaisler  Modular Timer Unit
          APB: 80003000 - 80003100
          IRQ: 9
          16-bit scalar, 7 * 32-bit timers, divisor 20

grmon3> info reg -v ahbstat0
AHB Status Register
    0x8000a000  Status register                         0x00000002
     9     ce                0x0         Correctable error
     8     ne                0x0         New error
     7     hw                0x0         HWRITE on error
     6:3   hm                0x0         HMASTER on error
     2:0   hs                0x2         HSIZE on error

    0x8000a004  Failing address register                0x8000a004
```

The GR716 has a clock-gating unit which can disable and enable clock gating and generate reset signals of certain cores in the SOC. With the GRMON3 **grcg** command the current setting of the clock-gating unit can be inspected and changed, the command line switch `-cginit` also affects the clock-gating unit. See [RD-4] for more information. Below is an example where the first SPI master controller clocks are enabled (not gated). The `APB UART0` clock was enabled on the GRMON3 command line.

```
grmon3> grcg
GRCLKGATE GR716 Primary info:
Unlock register:       0x00000000
Clock enable register: 0x00010000
Reset register:        0x7ffeffff

GR716 Primary decode of values:
+------+----------+---------------------------+----------+---------+-------+
| Gate | Core(s)  | Description               | Unlocked | Enabled | Reset |
+------+----------+---------------------------+----------+---------+-------+
|   0  | SPI2AHB  | SPI to AHB Bridge         |    0     |    0    |   1   |
|   1  | I2C2AHB  | I2C to AHB Bridge         |    0     |    0    |   1   |
|   2  | PWRX     | PacketWire RX Interface   |    0     |    0    |   1   |
|   3  | PWTX     | PacketWire TX Interface   |    0     |    0    |   1   |
|   4  | FTMCTRL  | Memory ctrl with EDAC     |    0     |    0    |   1   |
|   5  | SPIMCTRL | SPI Memory Controller 0   |    0     |    0    |   1   |
|   6  | SPIMCTRL | SPI Memory Controller 1   |    0     |    0    |   1   |
|   7  | SPICTRL  | SPI Controller 0          |    0     |    0    |   1   |
|   8  | SPICTRL  | SPI Controller 1          |    0     |    0    |   1   |
|   9  | I2CMST   | I2C-master 0              |    0     |    0    |   1   |
|  10  | I2CMST   | I2C-master 1              |    0     |    0    |   1   |
|  11  | I2CSLV   | I2C Slave 0               |    0     |    0    |   1   |
|  12  | I2CSLV   | I2C Slave 1               |    0     |    0    |   1   |
|  13  | GRDACADC | ADC / DAC Interface       |    0     |    0    |   1   |
|  14  | GRPWM    | PWM generator 0           |    0     |    0    |   1   |
|  15  | GRPWM    | PWM generator 1           |    0     |    0    |   1   |
|  16  | APBUART  | APB UART 0                |    0     |    1    |   0   |
|  17  | APBUART  | APB UART 1                |    0     |    0    |   1   |
|  18  | APBUART  | APB UART 2                |    0     |    0    |   1   |
|  19  | APBUART  | APB UART 3                |    0     |    0    |   1   |
|  20  | APBUART  | APB UART 4                |    0     |    0    |   1   |
|  21  | APBUART  | APB UART 5                |    0     |    0    |   1   |
|  23  | IOBO     | Enable IO control         |    0     |    0    |   1   |
|  24  | L3STAT   | LEON3 Statistics Unit     |    0     |    0    |   1   |
|  25  | AHBUART  | AHB Debug UART            |    0     |    0    |   1   |
|  26  | MEMPROT  | AMBA Protection unit      |    0     |    0    |   1   |
|  27  | BO       | Brownout detector         |    0     |    0    |   1   |
```

```
|  28  | SPWTDP   | CCSDS TDP                  |    0     |    0    |   1   |
|  29  | SPISLAVE | Dual-port SPI Slave        |    0     |    0    |   1   |
|  30  | FTMCTRL  | Memory ctrl with EDAC      |    0     |    0    |   1   |
+------+----------+----------------------------+----------+---------+-------+

grmon3> grcg enable 7

grmon3> grcg
GRCLKGATE GR716 Primary info:
Unlock register:       0x00000000
Clock enable register: 0x00010080
Reset register:        0x7ffeff7f

GR716 Primary decode of values:
+------+----------+----------------------------+----------+---------+-------+
| Gate | Core(s)  | Description                | Unlocked | Enabled | Reset |
+------+----------+----------------------------+----------+---------+-------+
|   0  | SPI2AHB  | SPI to AHB Bridge          |    0     |    0    |   1   |
|   1  | I2C2AHB  | I2C to AHB Bridge          |    0     |    0    |   1   |
|   2  | PWRX     | PacketWire RX Interface    |    0     |    0    |   1   |
|   3  | PWTX     | PacketWire TX Interface    |    0     |    0    |   1   |
|   4  | FTMCTRL  | Memory ctrl with EDAC      |    0     |    0    |   1   |
|   5  | SPIMCTRL | SPI Memory Controller 0    |    0     |    0    |   1   |
|   6  | SPIMCTRL | SPI Memory Controller 1    |    0     |    0    |   1   |
|   7  | SPICTRL  | SPI Controller 0           |    0     |    1    |   0   |
|   8  | SPICTRL  | SPI Controller 1           |    0     |    0    |   1   |
|   9  | I2CMST   | I2C-master 0               |    0     |    0    |   1   |
|  10  | I2CMST   | I2C-master 1               |    0     |    0    |   1   |
|  11  | I2CSLV   | I2C Slave 0                |    0     |    0    |   1   |
|  12  | I2CSLV   | I2C Slave 1                |    0     |    0    |   1   |
|  13  | GRDACADC | ADC / DAC Interface        |    0     |    0    |   1   |
|  14  | GRPWM    | PWM generator 0            |    0     |    0    |   1   |
|  15  | GRPWM    | PWM generator 1            |    0     |    0    |   1   |
|  16  | APBUART  | APB UART 0                 |    0     |    1    |   0   |
|  17  | APBUART  | APB UART 1                 |    0     |    0    |   1   |
|  18  | APBUART  | APB UART 2                 |    0     |    0    |   1   |
|  19  | APBUART  | APB UART 3                 |    0     |    0    |   1   |
|  20  | APBUART  | APB UART 4                 |    0     |    0    |   1   |
|  21  | APBUART  | APB UART 5                 |    0     |    0    |   1   |
|  23  | IOBO     | Enable IO control          |    0     |    0    |   1   |
|  24  | L3STAT   | LEON3 Statistics Unit      |    0     |    0    |   1   |
|  25  | AHBUART  | AHB Debug UART             |    0     |    0    |   1   |
|  26  | MEMPROT  | AMBA Protection unit       |    0     |    0    |   1   |
|  27  | BO       | Brownout detector          |    0     |    0    |   1   |
|  28  | SPWTDP   | CCSDS TDP                  |    0     |    0    |   1   |
|  29  | SPISLAVE | Dual-port SPI Slave        |    0     |    0    |   1   |
|  30  | FTMCTRL  | Memory ctrl with EDAC      |    0     |    0    |   1   |
+------+----------+----------------------------+----------+---------+-------+
```

5.5. GR716 License options

The GRMON3 evaluation version which is freely available can be used to operate the GR716-MINI board. The evaluation version does not support any other GR716 boards. The evaluation version is limited in certain regards compared with the GRMON3 professional product. GRMON3 can be downloaded from the GRMON3 homepage [RD-4].

The following table summarizes the GRMON3 license options for GR716.

Table 5.1. GRMON3 license options for GR716. All references to GRMON3 is for version 3.0.16 or later.

Program version	License	Supported hardware
GRMON3 professional	Professional	• All GR716 systems • All LEON/GRLIB systems
GRMON3 professional	GR716	• All GR716 systems
GRMON3 evaluation	Evaluation • No cost • No registration	• GR716-MINI

5.6. GR716 specific considerations

Information for users who are familiar with GRMON but new to GR716.

- All GR716 APBUART are clock-gated on power-on. GRMON3 command line option `-u` will not clock-enable the APBUART. To clock enable `uart0`, use:

```
grmon3> grcg enable 16
```

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

```
grmon3> grcg grcg0
grmon3> grcg grcg1
```

- Make sure applications are linked to the on-chip data/instruction SRAM which. Linking and executing in external SRAM is possible but slow.
- The on-chip boot ROM can be engaged from GRMON3 with **go 0**.
- When moving software between boards, note that different APBUART and pins are connected to the FTDI UART channels.

6. TSIM LEON simulator

6.1. Overview

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

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

The following section outlines how to use TSIM to emulate the GR716 MINI Development Board.

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

6.2. Startup

To start TSIM, use the command:

```
tsim-leon3 -gr716
```

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

```
tsim-leon3 -gr716 -mod module.so
```

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

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

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

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

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

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

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

  Register     Register description             Value
```

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

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

tsim> help

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

```
 silent                    Suppress stdout of a command.
 stack                     Show, set or clear initial stack pointer on current or given CPU
 step                      Single step on the current CPU.
 symbols                   Load symbols from file
 symbols list              Show symbols.
 symbols lookup            Lookup a symbol
 thread                    Print thread info or backtrace
 version                   Print the TSIM version and build date
 vwmem                     Write word(s) to a virtual address (and onwards)
 walk                      Print a MMU table walk for the current of given CPUs
 wmem                      Write word(s) to a address (and onwards)
 wmemh                     Write half-word(s) to a address (and onwards)
 wmemb                     Write byte(s) to a address (and onwards)
 xwmem                     Write a word to a address in a given address space

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

User commands:


Type Ctrl-C to interrupt execution

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

7. Toolchains

7.1. Bare C Cross-Compiler System

7.1.1. Overview

The Bare C Cross-Compiler (BCC for short) is a GNU-based cross-compilation system for LEON processors. It allows cross-compilation of C and C++ applications for LEON2, LEON3 and LEON4. This section gives the reader a brief introduction on how to use BCC together with the GR716 MINI Development Board. It will be demonstrated how to build an an example program and run it on the GR716-MINI using GRMON3.

The BCC toolchain includes the GNU C/C++ cross-compiler 7.2.0, GNU Binutils, Newlib embedded C library, the Bare-C run-time system with LEON support and the GNU debugger (GDB). The toolchain can be downloaded from [RD-10] and is available for both Linux and Windows. Further information about BCC can be found in [RD-11].

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

7.1.2. Compiling with BCC

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

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

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

$ sparc-gaisler-elf-gcc -qbsp=gr716 -mcpu=leon3 -O2 -g hello.c -o hello.elf
```

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

Table 7.1. BCC's GCC compiler relevant options

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

It is recommended to use the options

```
-qbsp=gr716 -mcpu=leon3
```

with GR716. For more details, see [RD-10].

7.1.3. Running and debugging with GRMON3

Once your application is compiled, connect to your GR716-MINI with GRMON3. The following log shows how to load and run an application. Note that the console output is redirected to GRMON3 by the use of the `-u` command line switch, so that the application standard output is forwarded to the GRMON3 console.

```
$ grmon -uart /dev/ttyUSB0 -u -cginit 0x00010000
  GRMON LEON debug monitor v3.0.15 64-bit pro version

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

```
grmon3> load hello.elf
  31000000 .text                    23.6kB /  23.6kB   [===============>] 100%
  30005E70 .data                     2.7kB /   2.7kB   [===============>] 100%
  Total size: 26.29kB (803.58kbit/s)
  Entry point 0x31000000
  Image hello.elf loaded

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 GRMON3 console. Compilation symbols are loaded automatically by GRMON3 once you load the application. An example is provided below.

```
grmon3> load hello.elf
  40000000 .text                    23.6kB /  23.6kB   [===============>] 100%
  40005E70 .data                     2.7kB /   2.7kB   [===============>] 100%
  Total size: 26.29kB (806.59kbit/s)
  Entry point 0x40000000
  Image hello.elf loaded

grmon3> bp main
  Software breakpoint 1 at <main>

grmon3> run

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

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 GRMON3 with the `-gdb` command line option and then attach a GDB session to it. For further information see Chapter 3 of [RD-11].

7.1.4. Running and debugging with TSIM

Once your application is compiled, start TSIM with the `-gr716` option. The following log shows how to load and run an application.

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

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


  Program exited normally.
```

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

```
tsim> load hello.elf
  section: .text, addr: 0x31000000, size 22400 bytes
```

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

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


  Program exited normally.
```

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

8. Software examples

This chapter will describe a software example collection applicable for GR716 and GR716-MINI applications. The approach is to provide a starting point for integrating different IO functionality with an application.

8.1. Overview

8.1.1. BCC device driver library

BCC ([RD-10]) includes a device driver library which can be used on different LEON systems. The device driver library comes with its own example collection which is also applicable to GR716.

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

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

8.1.2. Custom GR716 interface examples

A set of examples with GR716 specific functionality is distributed together with this document. It contains programs which which can be used on different LEON systems. The device driver library comes with its own example collection which is also applicable to GR716

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

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

8.2. Environment

All examples are designed to be loaded and run using GRMON3. Any board preparation is described in connection to each example.

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

8.3. IO switch matrix configuration

GR716 external pins can be reprogrammed for different functionality. A complete list of IO to pin routing options and values is available in [RD-2], section named *I/O switch matrix overview*.

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

GPIO	GR716 Data sheet and User's Manual	GRMON3 variable
0..7	`SYS.CFG.GP0`	`gpreg1::gpio0::gpio<N>`
8..15	`SYS.CFG.GP1`	`gpreg1::gpio1::gpio<N>`
16..23	`SYS.CFG.GP2`	`gpreg1::gpio2::gpio<N>`
24..31	`SYS.CFG.GP3`	`gpreg1::gpio3::gpio<N>`
32..39	`SYS.CFG.GP4`	`gpreg1::gpio4::gpio<N>`
40..47	`SYS.CFG.GP5`	`gpreg1::gpio5::gpio<N>`
48..55	`SYS.CFG.GP6`	`gpreg1::gpio6::gpio<N>`
56..63	`SYS.CFG.GP7`	`gpreg1::gpio7::gpio<N>`

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

8.3.1. Configuring with GRMON3

To list the current IO configuration using GRMON3, the **info reg gpreg1** command can be used:

Example 8.1.

```
grmon3> info reg -v gpreg1
  General Purpose Register Bank
      0x8000d000  System IO register for GPIO 0 to 7      0x00000000
      31:28  gpio7             0x0         Functional select for GPIO 7
      27:24  gpio6             0x0         Functional select for GPIO 6
      23:20  gpio5             0x0         Functional select for GPIO 5
      19:16  gpio4             0x0         Functional select for GPIO 4
      15:12  gpio3             0x0         Functional select for GPIO 3
      11:8   gpio2             0x0         Functional select for GPIO 2
       7:4   gpio1             0x0         Functional select for GPIO 1
       3:0   gpio0             0x0         Functional select for GPIO 0

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

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

Example 8.2.

```
grmon3> set gpreg1::gpio4::gpio39 1
grmon3> puts [set gpreg1::gpio4::gpio39]
  1
```

8.3.2. Configuring with the CPU

A C function is provided with the GR716 example collection for the purpose of configuring GR716 IO. It is named `set_pinfunc()` and is located in `common/include/pinhelper.h` and `common/pinhelper.c`.

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

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

Example 8.3.

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

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

Example 8.4.

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

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

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

8.4. BCC device drivers

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

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

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

8.4.1. GPIO

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

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

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

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

8.4.1.1. Preparation

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

8.4.2. Clock gating unit

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

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

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

8.4.2.1. Preparation

none

8.4.2.2. Build

```
cd $BCC/src/libdrv/examples/clkgate
make BSP=gr716
```

8.4.2.3. Run

```
grmon3> load bin/clkgate.elf
grmon3> run

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

clkgate0 configuration:
  enabled=08810060
  disabled=777eff9f

clkgate1 configuration:
  enabled=00000000
  disabled=003fffff
```

```
EXAMPLE END

  Program exited normally.

grmon3>
```

8.4.3. Memory protection unit

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

8.4.3.1. Preparation

The MEMPROT unit needs to be clock enabled. See the GRMON3 transcript below.

8.4.3.2. Build

```
  cd $BCC/src/libdrv/examples/memprot
  make BSP=gr716
```

8.4.3.3. Run

```
grmon3> grcg enable 26 grcg0
grmon3> load bin/memprot.elf
grmon3> run

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

INFO: memprot_dev_count() -> 2

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

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

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

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

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

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

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

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

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

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

example0: writes to 0x80004000 is now expected to trap with tt = 0x2B.
```

```
example0:   HINT: Use the GRMON 'reset' command to disable memory protection
example0: trying to write 0x80004000...

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

grmon3>
```

8.4.4. Memory scrubber

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

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

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

8.4.4.1. Forcing errors

The bus which memscrub0 probes is configurable in GR716. It is controlled by the *"Force Scrubber (FS)"* bit in *"`SYS.CFG.SCFG` - Interrupt test configuration register"*.

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

Clearing and setting FS can be done in GRMON3:

```
grmon3> set gpreg2::scfg::fs 0

grmon3> set gpreg2::scfg::fs 1
```

8.4.4.2. Preparation

None

8.4.4.3. Build

```
cd $BCC/src/libdrv/examples/memscrub
make BSP=gr716
```

8.4.4.4. Run

In the transcript below, `Interrupted!` means that execution was stopped by typing **Ctrl+C** in the the GRMON3 terminal. After the system state has been changed by the user, the GRMON3 command **cont** continues execution.

```
grmon3> set gpreg2::scfg::fs 1
grmon3> info reg -v gpreg2::scfg
  General Purpose Register
      0x8000e000  Interrupt test configuration register   0x00000800
      20:18  vref              0x0         Enable external voltage reference
      17:16  spw               0x0         SpaceWire loopback production test
      15     ll                0x0         LVDS External Loop
      14:13  ls                0x0         LVDS production test enable
      12     le                0x0         Support Locked transfers in SCRUBBER
      11     fs                0x1         Force Scrubber on main bus
      10:9   prot              0x0         Interrupt test protection bits
       8:3   irq               0x0         Generate interrupt
       2     mr                0x0         Disable reset signal from MIL core
       1     we                0x0         Disable reset request
       0     ee                0x0         Disable reset generation

grmon3> load bin/memscrub.elf
grmon3> run

Memory scrubber example.

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

```
example0: open memscrub0 -> OK

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


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

grmon3> set gpreg2::scfg::fs 0

grmon3> cont

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

grmon3> set gpreg2::scfg::fs 1

grmon3> cont
[R] UE detected, addr=30005eb0, rd, mst=2, size=4
[R] UE detected, addr=3000080c, rd, mst=2, size=4
[R] UE detected, addr=30000928, rd, mst=2, size=4
[R] UE detected, addr=30000a48, rd, mst=2, size=4
[R] UE detected, addr=30000b64, rd, mst=2, size=4


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

grmon3>
```

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

```
grmon3> mem ue_count 4
  0x30000b34  0000ea4d                                ...M

grmon3> mem ce_count 4
  0x30000b38  00000000
```

8.4.5. AHB status register

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

8.4.5.1. Forcing errors

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

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

8.4.5.2. Preparation

None

8.4.5.3. Build

```
cd $BCC/src/libdrv/examples/ahbstat
make BSP=gr716
```

8.4.5.4. Run

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

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

```
grmon3> load bin/ahbstat-isr.elf
grmon3> run

EXAMPLE BEGIN
Init with GR716 static tables
ahbstat_dev_count() -> 3
 ahbstat0: addr=8000a000, interrupt=63, device_id=0x052, version=0
 ahbstat1: addr=80306000, interrupt=63, device_id=0x052, version=0
 ahbstat2: addr=fff00000, interrupt=63, device_id=0x057, version=0
```

```
Trig interrupt...
Interrupt condition cleared
User ISR has been called 1 times
 count=1
 last_status=00000302
 last_failing_address=80300010
EXAMPLE END

  Program exited normally.

grmon3>
```

8.4.6. APBUART

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

8.4.6.1. BCC and UART

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

8.4.6.2. The APBUART device driver

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

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

8.4.6.3. Preparation

The following prepares to run the example with `apbuart3` with UART TX output on GPIO62 and UART RX input on GPIO61.

1. Connect a terminal to `apbuart3` and set 115200 BAUD. When using GR716-MINI, and the serial communication program *Minicom*, the following can be used:

   ```
   $ minicom -D /dev/ttyUSB3 -b 115200
   ```
2. Clock enable `apbuart3`.
3. Configure the appropriate GR716 pins for UART functionality. Set `SYS.CFG.GP7.GPIO62 = 0x1` (UART_TX3) and set `SYS.CFG.GP7.GPIO61 = 0x1` (UART_RX3).

The below commands performs step 2 and 3.

```
grmon3> # clock enable UART3
grmon3> grcg enable 19
grmon3> # connect UART3 signals to pins
grmon3> set gpreg1::gpio7::gpio61 1
grmon3> set gpreg1::gpio7::gpio62 1
```

8.4.6.4. Build

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

```
  cd $BCC/src/libdrv/examples/uart
  make BSP=gr716 CFLAGS="-DCFG_UART_INDEX=3 -DCFG_UART_BAUD=115200"
```

8.4.6.5. Run

```
grmon3> # clock enable UART0 for the BCC console output
grmon3> grcg enable 16
grmon3> load bin/uart.elf
grmon3> run

INFO: apbuart_dev_count() -> 6
apbuart0: addr=0x80300000, interrupt=24
apbuart1: addr=0x80301000, interrupt=25
apbuart2: addr=0x80302000, interrupt=42
apbuart3: addr=0x80303000, interrupt=44
apbuart4: addr=0x80304000, interrupt=45
```

```
apbuart5: addr=0x80305000, interrupt=46
INFO: end

  Program exited normally.

grmon3>
```

The following is displayed on the external terminal:

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

8.4.7. SPI master controller

GR716 has two *SPI controllers* (`SPICTRL`) typically named `spi0` and `spi1`. GR716 also has two *SPI memory controllers* (`SPIMCTRL`) typically named `spim0` and `spim1`. These four controllers operate independently.

GR716-MINI has SPI flash memory connected to the *SPI memory controller* `spim0`. The *SPI controllers* (`spi0` and `spi1`) are not connected to any SPI slave devices on the GR716-MINI.

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

8.4.7.1. Preparation

1. An internal loopback connection is setup by the example, effectively connecting `MISO` with `MOSI`. Thus, configuration of external IO is not needed.
2. Clock enable `spi0` with the GRMON3 command **grcg enable 7**.

8.4.7.1.1. External SPI bus

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

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

The IO pins should also be configured approrpiately.

8.4.7.2. Build

```
  cd $BCC/src/libdrv/examples/spi
  make BSP=gr716
```

8.4.7.3. Run

```
grmon3> load bin/spi.elf
grmon3> run

SPI example begin
PASS
SPI example end

  Program exited normally.

grmon3>
```

8.4.8. I2C master controller

GR716 has two *I2C master controllers*, `i2cmst0` and `i2cmst1` which can be connected to individual external I2C buses.

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

GR716-MINI does not have on-board I2C devices but the I2C buses are exported on the expansion connector.

8.4.8.1. Preparation

- Clock enable `i2cmst0` with the GRMON3 commands:

  ```
  grmon3> grcg enable 9
  ```

```
grmon3> grcg enable 10
```

8.4.8.2. Build

```
cd $BCC/src/libdrv/examples/i2cmst
make BSP=gr716
```

8.4.8.3. Run

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

```
grmon3> load bin/scan.elf
grmon3> run

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

INFO: i2cmst_dev_count() -> 2

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

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

 Program exited normally.

grmon3>
```

8.4.9. SpaceWire

The example `term` is an interactive terminal for operating the SpaceWire device driver. It allows monitoring link status, sending and receiving SpaceWire packets, displaying DMA statistics and more.

This example assumes that the GR716 LVDS pins are connected to a SpaceWire connector.

8.4.9.1. Preparation

- Option 1: Connect a SpaceWire loopback plug.
- Option 2: Connect a SpaceWire cable to another SpaceWire device.

Option 1 is used in the transcripts below.

8.4.9.2. Build

```
cd $BCC/src/libdrv/examples/grspw
make BSP=gr716
```

8.4.9.3. Run

APBUART0 debug forwarding is enabled in GRMON. The SpaceWire controller clock gating, PLL and LVDS configuration is done in the example when compiled with `BSP=gr716`.

```
grmon3> grcg enable 16 grcg0
grmon3> forward enable uart0
grmon3> load bin/term.elf
grmon3> run

Found 1 SpaceWire cores
 Initializing SpaceWire device 0
 After Link Start: RUN STATE

Starting SpW DMA channels
Starting GRSPW0: DMA Started Successfully
Starting Packet processing loop. Enter Command:

grspw> info
 grspw0: --- info begin ---
HARDWARE SUPPORT:
 RMAP:                   YES
 RMAP CRC:               YES
 UNALIGNED RX ADR:       YES
```

```
 NUMBER OF PORTS:         2
 DMA CHANNEL COUNT:       1
 HARDWARE VERSION:        0x00290002 GRSPW2
 grspw0: --- info end   ---
grspw> link
 GRSPW0 link/port setup:
 link_ctrl:      0x06
 clkdiv_start:   0x09
 clkdiv_run:     0x03
 link_state:     RUN STATE
 port_cfg:       0
 port_active:    Port0

grspw> r
 grspw0: --- receive begin ---
GRSPW0: Prepared 38 RX packet buffers for future reception
 grspw0: --- receive end   ---
grspw> x 0 1
 X0: scheduling packet on GRSPW0
GRSPW0: Sending 1 packets
 PKT of length 32 bytes, 0x01 0x9b 0x00 0x00 0x00 0x00 0x00 0x00
grspw> r
 grspw0: --- receive begin ---
GRSPW0: Recevied 1 packets
 PKT of length 32 bytes, 0x01 0x9b 0x00 0x00 0x00 0x00 0x00 0x00...
 grspw0: --- receive end   ---
grspw> r
 grspw0: --- receive begin ---
GRSPW0: Prepared 1 RX packet buffers for future reception
 grspw0: --- receive end   ---
grspw> q

EXAMPLE COMPLETED.

  Program exited normally.

grmon3>
```

8.4.10. CAN

`term` is an interactive CAN terminal using the GRCAN driver. It allows for sending and receiving messages on the CAN bus and operating the GRCAN device driver.

The example assumes that the GR716 CAN external signals are connected to CAN transceivers. They could be connected to the same external CAN bus.

8.4.10.1. Build

```
cd $BCC/src/libdrv/examples/grcan
make BSP=gr716
```

If no external CAN bus is available, then the example can be built with

```
make BSP=gr716 CFLAGS=-DCFG_INTLOOPBACK
```

to use on-chip loopback. This may need an external transceiver however.

8.4.10.2. Run

APBUART0 debug forwarding is enabled in GRMON. APBUART0, GRCAN0 and GRCAN1 are clock enabled. Pin multiplexing configuration for GRCAN is done inside the example program.

```
# can
grmon3> grcg enable 5 grcg1
grmon3> grcg enable 6 grcg1
# uart0
grmon3> grcg enable 16
grmon3> forward enable uart0
grmon3> load bin/trans.elf
grmon3> run
GRCAN example begin
Found 2 GRCAN cores

grcan0: opened
grcan1: opened

grcan0: started
grcan1: started
```

```
Starting Packet processing loop. Enter Command:

grcan> help
 help                   - List commands
 start DEV              - Start
 stop DEV               - Stop
 state [DEV]            - Driver state
 status [DEV]           - Get GRCAN core status register
 speed DEV HZ           - Set speed
 afilter DEV MASK CODE  - Set Acceptance filter
 sfilter DEV MASK CODE  - Set Sync Messages RX/TX filters
 q                      - Quit
 x DEV [ID] [STRING]    - Transmit one message
 r [DEV]                - Receive messages
 istxdone [DEV]         - Determine if all TX is done

grcan> x 0
 grcan0: scheduling messages:
MSG: STD length: 6, id: 0x1
MSGDATA: 0x61 0x62 0x63 0x31 0x32 0x33   abc123

grcan> r
 grcan0: --- receive begin ---
 grcan0: --- receive end   ---
 grcan1: --- receive begin ---
MSG[1]: STD length: 6, id: 0x1
MSGDATA[1]: 0x61 0x62 0x63 0x31 0x32 0x33   abc123
 grcan1: --- receive end   ---

grcan> x 1 4 hello
 grcan1: scheduling messages:
MSG: STD length: 5, id: 0x4
MSGDATA: 0x68 0x65 0x6c 0x6c 0x6f   hello

grcan> x 1 5 world
 grcan1: scheduling messages:
MSG: STD length: 5, id: 0x5
MSGDATA: 0x77 0x6f 0x72 0x6c 0x64   world

grcan> x 0 6 sure
 grcan0: scheduling messages:
MSG: STD length: 4, id: 0x6
MSGDATA: 0x73 0x75 0x72 0x65   sure

grcan> r
 grcan0: --- receive begin ---
MSG[1]: STD length: 5, id: 0x4
MSGDATA[1]: 0x68 0x65 0x6c 0x6c 0x6f   hello
MSG[2]: STD length: 5, id: 0x5
MSGDATA[2]: 0x77 0x6f 0x72 0x6c 0x64   world
 grcan0: --- receive end   ---
 grcan1: --- receive begin ---
MSG[1]: STD length: 4, id: 0x6
MSGDATA[1]: 0x73 0x75 0x72 0x65   sure
 grcan1: --- receive end   ---

grcan> istxdone
grcan0: istxdone: YES
grcan1: istxdone: YES

grcan> state
grcan0: STATE_STARTED
grcan1: STATE_STARTED
grcan> q

EXAMPLE COMPLETED.

  Program exited normally.

grmon3>
```

8.4.10.3. CAN bus considerations

There are two GRCAN cores in GR716: grcan0 and grcan1. Both have a normal and redundant buses (bus 0 and bus 1). This means that core grcan0 can use the external bus 0 or bus 1 to send or receive data. Similarly core grcan1 can use bus 0 or 1, provided that there is physical connection between the TX interface 0 to RX interface 0 and TX interface 1 to RX interface 1.

Nominal bus (0) of grcan0 and grcan1 are connected internally. Redundant bus (1) of grcan0 and grcan1 are connected internally. There is a `selection` bit for each GRCAN core which says which bus is to be used as active bus. For example, if `selection` bit is 0 for grcan0, nominal bus (0) is selected for grcan0.

If you select the same type of bus in both core, that is, `selection` bit is 0 for both GRCAN cores, this means that you are connecting both cores internally. But remember that even though you are testing internal loopback, there should be a working connection between the TX and RX of the transceiver: this is for the bus listening of TX.

Usually, for external loopback setup, you need the data to be sent out to the transceiver which is present on the accessory board. There is no CAN transceiver on the GR716-DEV board).

From transceiver TX to receiver of the other transceiver (corresponding to second GRCAN core), and from there to the RX pin on GR716-DEV->separate bus and then to the second core. So it is expected that the two cores select different type of buses to avoid internal loopback.

There is an enable signal in addition to the TX, and RX signals on each GRCAN core. This is used to enable the transceiver present outside on the accessory board. `enable0` is to enable transceiver for CAN A and `enable1` is to enable transceiver for CAN B.

8.5. GR716 interfaces

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

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

8.5.1. ADC

The ADC example takes a set of samples from analog input channel 0 using ADC unit 0 and prints the values.

8.5.1.1. ADC support library

A small support library for the GR716 ADC controller is available in `$EXAMPLE/common/include/adchelper.h` and `$EXAMPLE/common/adchelper.c`, allowing to perform common ADC configuration and operations. It is used in the example.

8.5.1.2. Preparation

Connect an analog input source to the board connector corresponding to ADC input 0.

8.5.1.3. Build

```
cd $EXAMPLE/adc
make
```

8.5.1.4. Run

```
grmon3> load adc.elf
grmon3> run

GR716 ADC example begin
config: adc0 ch0
sdata[ 0]:004f3
sdata[ 1]:004f3
sdata[ 2]:0050a
sdata[ 3]:0050a
sdata[ 4]:0050a
sdata[ 5]:0050a
sdata[ 6]:00509
sdata[ 7]:00509
GR716 ADC example end

  Program exited normally.

grmon3>
```

8.5.2. Clock and PLL

GR716 has flexible clocking and PLL features. Some common features are demonstrated in this example.

The system clock can run in the following modes:

- Normal mode, system input clock is directly derived from input clock. System clock will have the same frequency and duty cycle as the system input clock.
- Generated mode, system clock is derived from a combination of internal PLL and clock dividers.

- Sleep mode, system frequency is divided by a factor N while system is waiting for interrupt.

8.5.2.1. Clock and PLL support library

A support library for the GR716 PLL controller is available in `$EXAMPLE/common/include/clockhelper.h` and `$EXAMPLE/common/clockhelper.c`, allowing to perform common clock and PLL configurations.

It is recommended to refer to the clock distribution scheme figure in GR716 Data sheet and User's Manual when studying the support library.

8.5.2.2. Preparation

Connect the pin EXT_PLL to a clock source. This is part of the GR716-MINI default configuration.

8.5.2.3. Build

```
  cd $EXAMPLE/clock
  make
```

8.5.2.4. Run

```
grmon3> # clock enable APBUART0

grmon3> grcg enable 16

grmon3> # show APBUART0 output on grmon terminal

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

grmon3> # typical grpll0 configuration at power-on

grmon3> info reg -v grpll0
  GR716 Phase-locked loop
      0x8010d000  PLL configuration registers             0x80000000
      31     pd                0x1         PLL power down
       2:0   cfg               0x0         PLL configuration

      0x8010d004  PLL status registers                    0x00000002
       1     ll                0x1         PLL lost lock
       0     cl                0x0         PLL clock lock

      0x8010d008  PLL clock reference registers           0x00000300
       9:8   sel               0x3         PLL Reference Clock

      0x8010d00c  SpaceWire clock reference registers     0x00000302
      23:16  duty              0x0         SpaceWire Reference Clock duty cycle
       9:8   sel               0x3         SpaceWire Reference Clock source
       7:0   div               0x2         SpaceWire Reference Clock Divisor

      0x8010d010  MIL-1553B clock reference registers     0x00000000
      23:16  duty              0x0         MIL-1553B Reference Clock duty cycle
       9:8   sel               0x0         MIL-1553B Reference Clock source
       7:0   div               0x0         MIL-1553B Reference Clock Divisor

      0x8010d014  System clock reference registers        0x00000000
      23:16  duty              0x0         System Reference Clock duty cycle
       9:8   sel               0x0         System Reference Clock source
       7:0   div               0x0         System Reference Clock Divisor

      0x8010d018  Select system clock source             0x00000000
       0     s                 0x0         Select new system clock source
[...]

grmon3> load clock.elf

grmon3> run

GR716 PLL and clock example begin
PLL is currently NOT locked
Enable PLL and select external SpaceWire clock input...
PLL is currently locked
Internal system clock directly from SYS_CLK pin
Configure SpaceWire clock to 100 MHz...
GR716 PLL and clock example end

  Program exited normally.
```

```
grmon3> info reg -v grpll0
  GR716 Phase-locked loop
      0x8010d000  PLL configuration registers             0x00000001
      31     pd                0x0         PLL power down
       2:0   cfg               0x1         PLL configuration

      0x8010d004  PLL status registers                    0x00000003
       1     ll                0x1         PLL lost lock
       0     cl                0x1         PLL clock lock

      0x8010d008  PLL clock reference registers           0x00000100
       9:8   sel               0x1         PLL Reference Clock

      0x8010d00c  SpaceWire clock reference registers     0x00000108
      23:16  duty              0x0         SpaceWire Reference Clock duty cycle
       9:8   sel               0x1         SpaceWire Reference Clock source
       7:0   div               0x8         SpaceWire Reference Clock Divisor

      0x8010d010  MIL-1553B clock reference registers     0x00000000
      23:16  duty              0x0         MIL-1553B Reference Clock duty cycle
       9:8   sel               0x0         MIL-1553B Reference Clock source
       7:0   div               0x0         MIL-1553B Reference Clock Divisor

      0x8010d014  System clock reference registers        0x00000000
      23:16  duty              0x0         System Reference Clock duty cycle
       9:8   sel               0x0         System Reference Clock source
       7:0   div               0x0         System Reference Clock Divisor

      0x8010d018  Select system clock source             0x00000000
       0     s                 0x0         Select new system clock source
[...]

grmon3>
```

When changing frequency of the *Internal System Clock*, the interpretation of scaler/divider registers in various peripheral will change. In particular GPTIMER, APBUART and AHBUART are affected. This may result in (temporary) debug link failure.

8.5.3. External SRAM

GR716-MINI has an on-board external SRAM device connected to the parallel fault-tolerant memory controller (`mctrl0`). This example demonstrates how the external memory can be accessed and also provides a memory test.

8.5.3.1. Preparation

The example is self-contained when executed from GRMON3.

A set of test configuration parameters are available at the top of `memtest.c`. It allows for

- Specifying test region start and length.
- Describing how the program shall behave when a memory inconsistency is found:
  - Print a message
  - Trap
  - Execute a user defined function. This can for example be used to activate an external trig signal.

8.5.3.2. Build

```
cd $EXAMPLE/sram
make
```

8.5.3.3. Run

In the default configuration, 2 MiB of SRAM is tested over and over. A dot (`.`) is printed to the terminal after each 2 MiB test.

```
grmon3> grcg enable 16
grmon3> forward enable uart0
grmon3> load memtest.elf
grmon3> run

GR716 SRAM example begin
Testing memory in range 40000000 .. 401fffff
testname=memtest_seqdata
................................
................................
```

8.6. Boot loader

The GR716 on-chip ROM contains a boot loader which is engaged on power-on. The boot loader can copy a user application image from non-volatile external memory to on-chip RAM and start executing.

8.6.1. The ASW format

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

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

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

8.6.2. Scripts

File	Description
`bchfile.tcl`	Calculates BCH check bytes for binary input data. It write both the input data bytes and the BCH bytes to an output file appropriate for loading into SPI or parallel memory. • Input: Binary data • Output: Binary data or ELF
`img-gr716.tcl`	Tool for transforming an ELF file into an ASW image. By default a binary file ASW image file is created. If the -a option is present, then an ELF file suitable for loading with GRMON3 and TSIM is also created. • Input: ELF file • Output: Binary data or ELF

8.6.2.1. ASW image tool

A tool is provided in `$EXAMPLE/scripts/img-gr716.tcl` for generating ASW image files from application compiled and linked to RAM. The generated ASW image files are compatible with the ASW image format described in [RD-2] and can be loaded into non-volatile memory.

Command line parameter syntax for `img-gr716.tcl` is:

```
img-gr716.tcl [-a ASM_ADDR] [-o OUTFILE] INFILE...
```

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

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

`-a ASM_ADDR` is also optional and can be used to specify the target address of the ASW image in Application Storage Memory. If this option is used, then an additional output file is created, named `<FILE>.elf`. It contains the same data as the `<FILE>` file, but also has ELF information with load address. This allows for loading the image with the GRMON3 commands **load**, **flash load** or **spim flash load** and the TSIM command **load**.

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

8.6.2.2. BCH generation tool

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

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

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

Parameter	Description
`-elf`	Generate an elf output file in addition to the binary output. The ELF file contains target address information for convenient loading with GRMON3 or TSIM.
`-a ADDRESS`	Target address for the `-elf` option. Default is `0`.
`SOURCE`	name of the input binary file
`DEST`	name of the output binary file which will contain the SOURCE binary appended with calculated BCH check bytes.
`SIZE`	target ROM size, in KiB.

8.6.3. ASW image in SPI flash example

This section will describe how to convert an application compiled with BCC into an application image stored in external SPI flash memory for automatic start. The BCC application is linked to on-chip RAM and can also be loaded and executed directly with GRMON3 or TSIM.

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

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

8.6.3.1. Build

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

The make command automatically runs the scripts described above:

```
img-gr716.tcl fixup.elf hello-mini.elf -a 0x02000000 -o hello-mini.elf.img
bchfile.tcl -elf -a 0x02000000 hello-mini.elf.img hello-mini.elf.img.rom-with-bch 16384
```

The following files of interest are generated:

File	Description
`hello-mini.elf`	Application linked for execution in internal RAM. This file can be loaded and run with GRMON3 or TSIM.
`hello-mini.elf.img`	Raw ASW image for loading into non-volatile memory.
`hello-mini.elf.img.rom-with-bch.elf`	Same as above, and also includes BCH checkbytes for use with EDAC enabled memory controller.

The target load sections can be investigated:

Example 8.5.

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

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

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

```
  1 .bch          00001090  02ffef70  02ffef70  00004274  2**0
                  CONTENTS, ALLOC, LOAD, DATA
```

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

8.6.3.2. Load and Run

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

8.6.3.2.1. GRMON3

When using GRMON3 the image can be loaded with:

```
grmon3> spim flash detect
  Got manufacturer ID 0xc2 and device ID 0x2019
  Detected device: Macronix MX25L25635F

grmon3> spim flash load -erase hello-mini.elf.img.rom-with-bch.elf
  02000000 .data                    16.6kB /  16.6kB   [===============>] 100%
  02FFEF70 .bch                      4.1kB /   4.1kB   [===============>] 100%
  Total size: 20.70kB (2.56kbit/s)
  Entry point 0x00000000
  Image hello-mini.elf.img.rom-with-bch.elf loaded

grmon3> verify hello-mini.elf.img.rom-with-bch.elf
  02000000 .data                    16.6kB /  16.6kB   [===============>] 100%
  02FFEF70 .bch                      4.1kB /   4.1kB   [===============>] 100%
  Total size: 20.70kB (64.05kbit/s)
  Entry point 0x00000000
  Image of hello-mini.elf.img.rom-with-bch.elf verified without errors
```

Make sure the bootstrap signals are set for SPI ASW boot (default) and then power-off and power-on the board. The image can also be started by issuing **go 0** in GRMON3.

8.6.3.2.2. TSIM

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

```
$ ./tsim-leon3 -gr716 -bootstrap 0x0000c002 -u 3
...
tsim> load hello-board.elf.img.rom-with-bch.elf
  section: .data, addr: 0x2000000, size 16960 bytes
  section: .bch, addr: 0x2ffef70, size 4240 bytes
  read 5 symbols
```

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

8.6.3.3. Output

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

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

8.7. IO switch validation script and configuration

The IO validation and configuration script `scripts/iomx.tcl` is written in Tcl and contains lists for mapping functional pins to physical pins on the GR716 as described in the section I/O switch matrix overview in the GR716 Data sheet and User's Manual [RD-2].

The intention of the script is to help the user of the GR716 to validate a configuration according to section I/O switch matrix overview in the GR716 Data sheet and User's Manual [RD-2] and to quickly setup IO configuration for the GR716 MINI Development Board.

The IO validation and configuration script prints the following sections and information:

- IO configuration register constants. The IO configuration constants can be used to update the IO registers in order to set the target configuration.
- Pin list information. Lists interface selected per pin.
- Any overlapping interfaces e.g. multiple interfaces using the same external pin.

The script can only detect overlapping interfaces if the interface pin mapping corresponds with an interface on the PCB.

The script will NOT detect erroneous connection on custom boards or interfaces.

8.7.1. Scripts Sections

The script is divided into a number of sections in order to create a valid pin configuration:

- *Interface pin mapping.* This section lists and maps all existing functional pins described in the *I/O switch matrix overview* of the GR716 Data sheet and User's Manual [RD-2].
- *IO configuration.* This section groups and names interfaces pins.

For description of groups see *I/O switch matrix pin validation script* in the GR716 Data sheet and User's Manual [RD-2].

8.7.2. Usage of the pin validation script

The script needs to be modified and loaded into in a Tcl interpreter, for example GRMON3. After the script has been loaded, the Tcl command `gen_config` can be used to generate the external IO configuration register settings. The script is prepared to automatically execute the `gen_config` and print the default configuration.

```
grmon3> source scripts/iomx.tcl
```

8.8. Board IO Configuration

This section describes board IO configurations,

8.8.1. Default IO configuration

The GR716 MINI Development Board is fitted with external SRAM. To use the external SRAM board, the IO configuration registers should be set according to this chapter. The default configuration is printed by the script and named `gr716_mini_1`:

```
//#############################################################################
// C constant (gr716_mini_1)
//#############################################################################

const int gr716_mini_1[8] = {
0x22222222,
0x22222222,
0x00002222,
0x22222220,
0x00000222,
0x00000000,
0x00000220,
0x00000000};

#############################################################################
# Pin list
#############################################################################

pin[0]: mem_addr(0)
pin[1]: mem_addr(1)
pin[2]: mem_addr(2)
pin[3]: mem_addr(3)
pin[4]: mem_addr(4)
pin[5]: mem_addr(5)
pin[6]: mem_addr(6)
pin[7]: mem_addr(7)
pin[8]: mem_addr(8)
pin[9]: mem_addr(9)
pin[10]: mem_addr(10)
```

```
pin[11]: mem_addr(11)
pin[12]: mem_addr(12)
pin[13]: mem_addr(13)
pin[14]: mem_addr(14)
pin[15]: mem_addr(15)
pin[16]: mem_addr(16)
pin[17]: mem_addr(17)
pin[18]: mem_addr(18)
pin[19]: ram_csn(0)
pin[20]: gpio(20)
pin[21]: gpio(21)
pin[22]: gpio(22)
pin[23]: gpio(23)
pin[24]: gpio(24)
pin[25]: mem_data(0)
pin[26]: mem_data(1)
pin[27]: mem_data(2)
pin[28]: mem_data(3)
pin[29]: mem_data(4)
pin[30]: mem_data(5)
pin[31]: mem_data(6)
pin[32]: mem_data(7)
pin[33]: mem_oen
pin[34]: mem_wrn
pin[35]: gpio(35)
pin[36]: gpio(36)
pin[37]: gpio(37)
pin[38]: gpio(38)
pin[39]: gpio(39)
pin[40]: gpio(40)
pin[41]: gpio(41)
pin[42]: gpio(42)
pin[43]: gpio(43)
pin[44]: gpio(44)
pin[45]: gpio(45)
pin[46]: gpio(46)
pin[47]: gpio(47)
pin[48]: gpio(48)
pin[49]: mem_addr(19)
pin[50]: mem_addr(20)
pin[51]: gpio(51)
pin[52]: gpio(52)
pin[53]: gpio(53)
pin[54]: gpio(54)
pin[55]: gpio(55)
pin[56]: gpio(56)
pin[57]: gpio(57)
pin[58]: gpio(58)
pin[59]: gpio(59)
pin[60]: gpio(60)
pin[61]: gpio(61)
pin[62]: gpio(62)
pin[63]: gpio(63)
```

8.9. Resources

Table 8.1. GR716 software examples resources

GR716-AN-0002	External ROM Memory boot option using the GR716 [https://www.gaisler.com/index.php/products/components/gr716]
GR716-AN-0003	SPI Memory boot option using the GR716 [https://www.gaisler.com/index.php/products/components/gr716]
GR716-BP	GR716 Quick Start Guide Board Package, `gr716-examples-<DATE>.zip` [https://www.gaisler.com/index.php/products/components/gr716]

9. Expansion boards

The GR716 MINI Development Board is fitted with an 80-pin expansion connector (bottom side) which provides access to:

- External power supply. See section 5.3.1 for using alternative supply source.
- External reset in signal
- Distribution of system reset from GR716 device
- External system and SpaceWire clock
- Distribution of external voltage reference
- LVDS interface for SpaceWire or SPI4S
- Analog 4xDAC + 8xADC (or use as standard GPIO)
- 12 GPIO (or maximum of 24 GPIO when analog pins are used for GPIO).

The 24 GPIO and 6 LVDS pins available makes it possible to test and prototype the following interfaces and features via the expansion connector:

- SpaceWire via LVDS and/or CMOS pin standard
- SPI4S via LVDS and/or CMOS pin standard
- CAN interface including support for redundant CAN interface
- MIL-1553B interface
- PacketWire interface
- Up to 2 individual SPI masters/slave interface with 4 external slave select for master interface
- Up to 2 individual I2C masters interface
- Up to 2 individual I2C slave interface
- Up to 16 PWM channels
- Up to 4 individual UART interfaces
- Up to 24 individual GPIO pins
- Remote access via SpaceWire, SPI4S, UART and I2C

The pinout for the expansion connector can be found in [RD-1].

9.1. GR716-MINI Carrier Board

The GR716 MINI Development Board expansion connector can be connected to the GR716-MINI Carrier board to:

- Easy connect to/from GPIO signals
- LED for testing GPIO signals

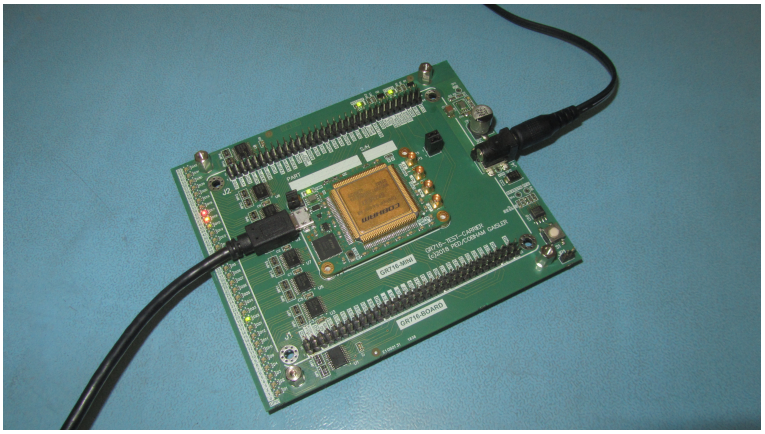


Figure 9.1. GR716-MINI Carrier Board

For more information about the GR716-MINI Carrier Board, contact support@gaisler.com.

10. Support

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

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

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

Appendix A. Assembly drawing

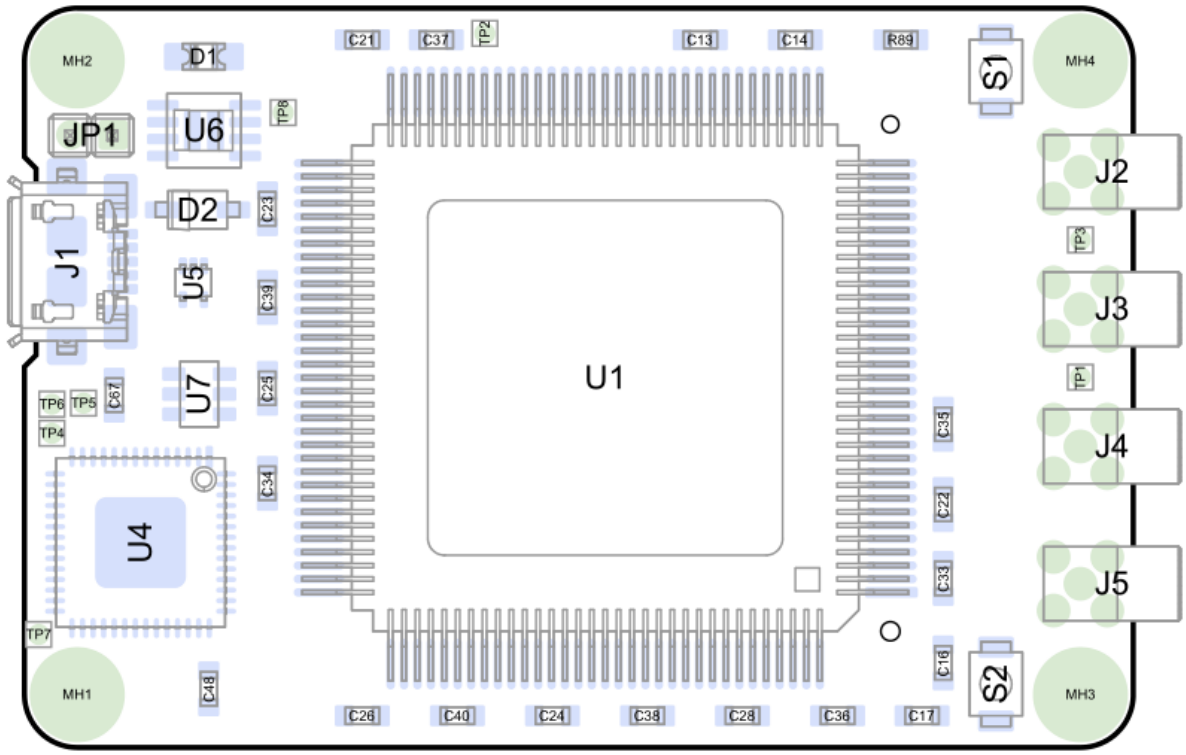


Figure A.1. GR716-MINI top

Appendix B. Default configuration

This appendix describes the jumper and switch positions for the default configuration of the board. For more details, see the section named *Setting Up and Using the Board* in the GR716-MINI Development Board User's Manual [RD-1].

Table B.1. Jumper configuration

Item	Default	Description of the default configuration
JP1	1-2	Power board via USB

Table B.2. Bootstrap configuration

Item	Default	Description of the default configuration
GPIO0	High	Disable EDAC
GPIO17	Low	Enable internal Boot PROM
GPIO62	Low	Disable memory test
GPIO63	Low	Disable redundant memory
DSUTX	High	Enable use of ASW container (TBD)
MOSI	Low	Boot from external SPI memory
SCK	Low	Boot from external SPI memor
SEL	Low	Boot from internal memory
DSUBRE	Low	Processor shall always start

Default bootstrap values can be configured by changing on-board pull up/down resistors. It is also possible to change the bootstrap after power-on by writing the bootstrap configuration registers.

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