GR716-MINI Quick Start Guide
# Table of Contents

1. Introduction ........................................................................................................... 4
   1.1. Overview .................................................................................................... 4
   1.2. References ................................................................................................. 4
2. Overview .................................................................................................................. 5
3. Board Configuration ............................................................................................... 7
   3.1. Overview .................................................................................................... 7
   3.2. Default configuration .................................................................................... 7
   3.3. Power .......................................................................................................... 7
       3.3.1. Alternative power .................................................................................. 7
   3.4. Bootstrap ...................................................................................................... 7
   3.5. Reset ........................................................................................................... 7
   3.6. Break .......................................................................................................... 8
   3.7. Clock .......................................................................................................... 8
   3.8. Pin multiplexing ........................................................................................... 8
   3.9. Analog I/O ................................................................................................. 8
4. Software Development Environment ...................................................................... 9
   4.1. Overview .................................................................................................... 9
5. GRMON3 hardware debugger ............................................................................... 10
   5.1. Overview .................................................................................................... 10
   5.2. Debug links ................................................................................................ 10
       5.2.1. Connecting via the FTDI USB/UART interface .................................... 10
   5.3. First steps ................................................................................................... 10
   5.4. Connecting to the board .............................................................................. 10
   5.5. GR716 License options ............................................................................... 20
   5.6. GR716 specific considerations .................................................................... 21
6. TSIM LEON simulator ........................................................................................... 22
   6.1. Overview .................................................................................................... 22
   6.2. Startup ....................................................................................................... 22
7. Toolchains ............................................................................................................... 25
   7.1. Bare C Cross-Compiler System .................................................................. 25
       7.1.1. Overview ............................................................................................. 25
       7.1.2. Compiling with BCC .......................................................................... 25
   7.1.3. Running and debugging with GRMON3 .................................................. 25
   7.1.4. Running and debugging with TSIM ......................................................... 26
8. Software examples ................................................................................................. 28
   8.1. Overview .................................................................................................... 28
       8.1.1. BCC device driver library ................................................................... 28
       8.1.2. Custom GR716 interface examples ...................................................... 28
   8.2. Environment ................................................................................................ 28
   8.3. IO switch matrix configuration ..................................................................... 28
       8.3.1. Configuring with GRMON3 ................................................................. 28
       8.3.2. Configuring with the CPU .................................................................. 29
   8.4. BCC device drivers ..................................................................................... 30
       8.4.1. GPIO ................................................................................................... 30
       8.4.2. Clock gating unit ................................................................................ 30
   8.4.3. Memory protection unit ........................................................................... 31
   8.4.4. Memory scrubber ................................................................................... 32
   8.4.5. AHB status register ............................................................................... 33
   8.4.6. APBUART .............................................................................................. 34
   8.4.7. SPI master controller ............................................................................. 35
   8.4.8. I2C master controller ............................................................................ 36
   8.4.9. SpaceWire ............................................................................................ 36
   8.4.10. CAN ..................................................................................................... 37
   8.5. GR716 interfaces ......................................................................................... 39
       8.5.1. ADC .................................................................................................... 39
8.5.2. Clock and PLL ................................................................. 40
8.5.3. External SRAM ............................................................... 41
8.6. Boot loader ......................................................................... 42
  8.6.1. The ASW format ......................................................... 42
  8.6.2. Scripts ......................................................................... 42
  8.6.3. ASW image in SPI flash example ................................. 43
8.7. IO switch validation script and configuration ..................... 45
  8.7.1. Scripts Sections .......................................................... 45
  8.7.2. Usage of the pin validation script ................................. 46
8.8. Board IO Configuration ...................................................... 46
  8.8.1. Default IO configuration ............................................. 46
8.9. Resources ........................................................................... 47
9. Expansion boards ................................................................. 48
  9.1. GR716-MINI Carrier Board ............................................. 48
10. Support .................................................................................. 49
A. Assembly drawing ............................................................... 50
B. Default configuration .......................................................... 51
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. References

Table 1.1. References

<table>
<thead>
<tr>
<th>RD</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD-10</td>
<td>Bare C Cross-Compilation System [<a href="https://www.gaisler.com/index.php/products/operating-systems/bcc">https://www.gaisler.com/index.php/products/operating-systems/bcc</a>]</td>
</tr>
</tbody>
</table>

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

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

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.

3.3.1. Alternative power

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

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

Cobham 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.

Cobham 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. Cobham 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](https://www.gaisler.com/index.php/products/debug-tools) and in detail in [RD-4].

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

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

Component                            Vendor
AHB-to-AHB Bridge                    Cobham Gaisler
MIL-STD-1553B Interface              Cobham Gaisler
GRPFW2 SpaceWire Serial Link         Cobham Gaisler
SPI to AHB Bridge                    Cobham Gaisler
I2C to AHB Bridge                    Cobham Gaisler
CAN Controller with DMA              Cobham Gaisler
CAN Controller with DMA              Cobham Gaisler
AHB Debug UART                       Cobham Gaisler
AHB-to-AHB Bridge                    Cobham Gaisler
AHB-to-AHB Bridge                    Cobham Gaisler
AHB-to-AHB Bridge                    Cobham Gaisler
PacketWire Receiver with DMA         Cobham Gaisler
PacketWire Transmitter with DMA      Cobham Gaisler
GRDMAC DMA Controller                Cobham Gaisler
GRDMAC DMA Controller                Cobham Gaisler
GRDMAC DMA Controller                Cobham Gaisler
GRDMAC DMA Controller                Cobham Gaisler
Dual-port SPI Slave                  Cobham Gaisler
LEON3FT SPARC V8 Processor           Cobham Gaisler
AHB-to-AHB Bridge                    Cobham Gaisler
AHB Memory Scrubber                  Cobham Gaisler
AHB-to-AHB Bridge                    Cobham Gaisler
AHB Debug UART                       Cobham Gaisler
AHB Debug UART                       Cobham Gaisler
AHB Debug UART                       Cobham Gaisler
Dual-port AHBl(CPU) On-Chip RAM      Cobham Gaisler
Dual-port AHBl(CPU) On-Chip RAM      Cobham Gaisler
Generic AHB ROM                      Cobham Gaisler
Memory controller with EDAC          Cobham Gaisler
SPI Memory Controller                Cobham Gaisler
SPI Memory Controller                Cobham Gaisler
AHB/APB Bridge                      Cobham Gaisler
AHB/APB Bridge                      Cobham Gaisler
AHB/APB Bridge                      Cobham Gaisler
Memory controller with EDAC          Cobham Gaisler
LEON3 Debug Support Unit             Cobham Gaisler
AHB/APB Bridge                      Cobham Gaisler
AMBA Trace Buffer                    Cobham Gaisler
Multi-processor Interrupt Ctrl.      Cobham Gaisler
Modular Timer Unit                   Cobham Gaisler
Modular Timer Unit                   Cobham Gaisler
GR716 AMBA Protection unit           Cobham Gaisler
Clock gating unit                    Cobham Gaisler
Clock gating unit                    Cobham Gaisler
General Purpose Register             Cobham Gaisler
LEON3 Statistics Unit                Cobham Gaisler
AHB Status Register                  Cobham Gaisler
CCDS TDP / SpaceWire I/F             Cobham Gaisler
General Purpose Register Bank        Cobham Gaisler
General Purpose Register             Cobham Gaisler
GR716 AMBA Protection unit           Cobham Gaisler
GR716 Bandgap                        Cobham Gaisler
GR716 Brownout detector             Cobham Gaisler
GR716 Phase-locked loop              Cobham Gaisler
Generic UART                         Cobham Gaisler
Generic UART                         Cobham Gaisler
Generic UART                         Cobham Gaisler
AHB Status Register                  Cobham Gaisler
ADC / DAC Interface                  Cobham Gaisler
SPI Controller                       Cobham Gaisler
SPI Controller                       Cobham Gaisler
PWM generator                        Cobham Gaisler
General Purpose I/O port             Cobham Gaisler
General Purpose I/O port             Cobham Gaisler
AMBA Wrapper for OC I2C-master       Cobham Gaisler
AMBA Wrapper for OC I2C-master       Cobham Gaisler
```
Use command 'info sys' to print a detailed report of attached cores

```
grmon3> info sys
ahb2ahb0 Cobham Gaisler AHB-to-AHB Bridge
  AHB Master 0
  AHB: 00000000 - 20000000
  AHB: 40000000 - 60000000
  AHB: 80000000 - 90000000
  AHB: F0000000 - 00000000

gr1553b0 Cobham Gaisler MIL-STD-1553B Interface
  AHB Master 1
  APB: 80101000 - 80101100
  IRQ: 4
  Device is disabled

grspw0 Cobham Gaisler GRSPW2 SpaceWire Serial Link
  AHB Master 2
  APB: 80100000 - 80100100
  IRQ: 5
  Device is disabled

spi2ahb0 Cobham Gaisler SPI to AHB Bridge
  AHB Master 3
  APB: 80100400 - 80100410
  IRQ: 62
  Device is disabled

i2c2ahb0 Cobham Gaisler I2C to AHB Bridge
  AHB Master 4
  APB: 80103000 - 80103400
  IRQ: 47
  Device is disabled

grcan0 Cobham Gaisler CAN Controller with DMA
  AHB Master 5
  APB: 80102000 - 80102400
  IRQ: 21
  Device is disabled

grcan1 Cobham Gaisler CAN Controller with DMA
  AHB Master 6
  APB: 80103000 - 80103400
  IRQ: 21
  Device is disabled

ahbuart0 Cobham Gaisler AHB Debug UART
  AHB Master 7
  APB: 8000F000 - 8000F100
  Device is disabled

ahb2ahb1 Cobham Gaisler AHB-to-AHB Bridge
  AHB Master 8
  AHB: 90000000 - A0000000

grpwrx0 Cobham Gaisler PacketWire Receiver with DMA
  AHB Master 9
  APB: 8010E000 - 8010E100
  IRQ: 2
  Device is disabled

grpwx0 Cobham Gaisler PacketWire Transmitter with DMA
  AHB Master 10
  APB: 8010F000 - 8010F100
  IRQ: 3
  Device is disabled

dma0 Cobham Gaisler GRDMAC DMA Controller
  AHB Master 11
  APB: 80106000 - 80106200
  IRQ: 6
  Device is disabled

dma1 Cobham Gaisler GRDMAC DMA Controller
  AHB Master 12
  APB: 80107000 - 80107200
  IRQ: 6
```
Device is disabled
dma2      Cobham Gaisler  GRDMAC DMA Controller
AHB Master 13
APB: 80108000 - 80108200
IRQ:  6
Device is disabled
dma3      Cobham Gaisler  GRDMAC DMA Controller
AHB Master 14
APB: 80109000 - 80109200
IRQ:  6
Device is disabled
spislv0   Cobham Gaisler  Dual-port SPI Slave
AHB Master 15
APB: 8040F000 - 8040F100
IRQ:  62
Device is disabled
cpu0      Cobham Gaisler  LEON3FT SPARC V8 Processor
AHB Master 0
ahb2ahb2  Cobham Gaisler  AHB-to-AHB Bridge
AHB Master 1
AHB: 30000000 - 30100000
AHB: 90000000 - A0000000
memscrub0 Cobham Gaisler  AHB Memory Scrubber
AHB Master 2
AHB: FFF00000 - FFF00100
IRQ:  63
burst length: 8 bytes
ahb2ahb3  Cobham Gaisler  AHB-to-AHB Bridge
AHB Master 0
AHB: 00000000 - 80000000
AHB: 80000000 - 90000000
AHB: F0000000 - 00000000
ahbuart1  Cobham Gaisler  AHB Debug UART
AHB Master 1
APB: 94000000 - 94000100
Baudrate 115200, AHB frequency 20.00 MHz
diram0    Cobham Gaisler  Dual-port AHB/(CPU) On-Chip RAM
AHB: 30000000 - 30100000
APB: 80001000 - 80001100
IRQ:  63
32-bit static ram: 64 KB @ 0x30000000
ilr0      Cobham Gaisler  Dual-port AHB/(CPU) On-Chip RAM
AHB: 31000000 - 31100000
APB: 80008000 - 8000B100
IRQ:  63
32-bit static ram: 128 KB @ 0x31000000
ahbrom0   Cobham Gaisler  Generic AHB ROM
AHB: 00000000 - 00100000
32-bit ROM: 1 MB @ 0x00000000
mcrt0     Cobham Gaisler  Memory controller with EDAC
AHB: 01000000 - 02000000
AHB: 40000000 - 50000000
AHB: 80000000 - 80001000
Device is disabled
spin0     Cobham Gaisler  SPI Memory Controller
AHB: FFF00100 - FFF00200
AHB: 02000000 - 04000000
IRQ:  52
Device is disabled
spin1     Cobham Gaisler  SPI Memory Controller
AHB: FFF00200 - FFF00300
AHB: 04000000 - 06000000
IRQ:  52
Device is disabled
apbmst0   Cobham Gaisler  AHB/APB Bridge
AHB: 80000000 - 80100000
apbmst1   Cobham Gaisler  AHB/APB Bridge
AHB: 80100000 - 80200000
apbmst2   Cobham Gaisler  AHB/APB Bridge
AHB: 80300000 - 80400000
apbmst3   Cobham Gaisler  AHB/APB Bridge
AHB: 80400000 - 80500000
mcrt1     Cobham Gaisler  Memory controller with EDAC
AHB: 51000000 - 52000000
AHB: 50000000 - 51000000
AHB: 80307000 - 80307100
Device is disabled
dsu0      Cobham 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, GRFPFU-lite
icache not implemented
dcache not implemented

apbmst4 Cobham Gaisler AHB/APB Bridge
AHB: 94000000 - 94100000

ahbtrace Cobham Gaisler AMBA Trace Buffer
AHB: 9FF20000 - 9FF40000
Trace buffer size: 128 lines

irqmp0 Cobham Gaisler Multi-processor Interrupt Ctrl.
APB: 80002000 - 80002400
IRQ: 1

qtimer0 Cobham Gaisler Modular Timer Unit
APB: 80000000 - 80001000
IRQ: 9
16-bit scalar, 7 * 32-bit timers, divisor 20

qtimer1 Cobham Gaisler Modular Timer Unit
APB: 80004000 - 80005100
IRQ: 53
16-bit scalar, 7 * 32-bit timers, divisor 20

memprot0 Cobham Gaisler GR716 AMBA Protection unit
APB: 80005000 - 80005200
IRQ: 63
Device is disabled

grcg0 Cobham Gaisler Clock gating unit
APB: 80006000 - 80006100
GRMON enabled clocks during initialization
Index for use in GRMON: 0

grcg1 Cobham Gaisler Clock gating unit
APB: 80007000 - 80007100
GRMON enabled clocks during initialization
Index for use in GRMON: 1

gpreg0 Cobham Gaisler General Purpose Register
APB: 80008000 - 80008100

13stat0 Cobham Gaisler LEON3 Statistics Unit
APB: 80009000 - 80009400
Device is disabled

ahbstat0 Cobham Gaisler AHB Status Register
APB: 8000A000 - 8000A100
IRQ: 63

spwtdp0 Cobham Gaisler CCSDS TDP / SpaceWire I/F
APB: 8000C000 - 8000C200
IRQ: 43
Device is disabled

gpreg1 Cobham Gaisler General Purpose Register Bank
APB: 8000D000 - 8000D100
IRQ: 63

grPLL0 Cobham Gaisler GR716 Phase-locked loop
APB: 8010D000 - 8010D100
IRQ: 63
Device is disabled

uart0 Cobham Gaisler Generic UART
APB: 80300000 - 80300100
IRQ: 24
Baudrate 38461, FIFO debug mode available

uart1 Cobham Gaisler Generic UART
APB: 80301000 - 80301100
IRQ: 25
Device is disabled

uart2 Cobham Gaisler Generic UART
APB: 80302000 - 80302100
IRQ: 42
Device is disabled

uart3 Cobham Gaisler Generic UART
APB: 80303000 - 80303100
IRQ: 44
Device is disabled

uart4 Cobham Gaisler Generic UART
APB: 80304000 - 80304100
IRQ: 45
Device is disabled

uart5 Cobham Gaisler Generic UART
APB: 80305000 - 80305100
IRQ: 46
Device is disabled
**ahbstat1**  Cobham Gaisler  AHB Status Register  
APB: 80306000 - 80306100  
IRQ: 63

**gradcdac0**  Cobham Gaisler  ADC / DAC Interface  
APB: 80308000 - 80308100  
IRQ: 16  
Device is disabled

**spi0**  Cobham Gaisler  SPI Controller  
APB: 80309000 - 80309100  
IRQ: 48  
Device is disabled

**spi1**  Cobham Gaisler  SPI Controller  
APB: 8030A000 - 8030A100  
IRQ: 49  
Device is disabled

**grpwm0**  Cobham Gaisler  PWM generator  
APB: 80310000 - 80310100  
IRQ: 8  
Device is disabled

**gpio0**  Cobham Gaisler  General Purpose I/O port  
APB: 803CC000 - 803C0000  
IRQ: 17

**gpio1**  Cobham Gaisler  General Purpose I/O port  
APB: 803C0000 - 803C0000  
IRQ: 38

**i2cmst0**  Cobham Gaisler  AMBA Wrapper for OC I2C-master  
APB: 8030E000 - 8030E100  
IRQ: 50  
Device is disabled

**i2cmst1**  Cobham Gaisler  AMBA Wrapper for OC I2C-master  
APB: 8030F000 - 8030F100  
IRQ: 51  
Device is disabled

**adc0**  Cobham Gaisler  GR716 Analog-to-Digital Conv  
APB: 80400000 - 80400100  
IRQ: 28  
Device is disabled

**adc1**  Cobham Gaisler  GR716 Analog-to-Digital Conv  
APB: 80401000 - 80401100  
IRQ: 29  
Device is disabled

**adc2**  Cobham Gaisler  GR716 Analog-to-Digital Conv  
APB: 80402000 - 80402100  
IRQ: 30  
Device is disabled

**adc3**  Cobham Gaisler  GR716 Analog-to-Digital Conv  
APB: 80403000 - 80403100  
IRQ: 31  
Device is disabled

**adc4**  Cobham Gaisler  GR716 Analog-to-Digital Conv  
APB: 80404000 - 80404100  
IRQ: 32  
Device is disabled

**adc5**  Cobham Gaisler  GR716 Analog-to-Digital Conv  
APB: 80405000 - 80405100  
IRQ: 33  
Device is disabled

**adc6**  Cobham Gaisler  GR716 Analog-to-Digital Conv  
APB: 80406000 - 80406100  
IRQ: 34  
Device is disabled

**adc7**  Cobham Gaisler  GR716 Analog-to-Digital Conv  
APB: 80407000 - 80407100  
IRQ: 35  
Device is disabled

**dac0**  Cobham Gaisler  GR716 Digital-to-Analog Conv  
APB: 80408000 - 80408100  
IRQ: 26  
Device is disabled

**dac1**  Cobham Gaisler  GR716 Digital-to-Analog Conv  
APB: 80409000 - 80409100  
IRQ: 27  
Device is disabled

**dac2**  Cobham Gaisler  GR716 Digital-to-Analog Conv  
APB: 8040A000 - 8040A100  
IRQ: 36  
Device is disabled

**dac3**  Cobham Gaisler  GR716 Digital-to-Analog Conv  
APB: 8040B000 - 8040B100  
IRQ: 37  
Device is disabled

**i2cslv0**  Cobham Gaisler  I2C Slave  
APB: 8040C000 - 8040C100  
IRQ: 7
i2cslv1 Cobham Gaisler I2C Slave
APB: 80400000 - 80400100
IRQ: 47

gpwm1 Cobham Gaisler PMM generator
APB: 80410000 - 80410100
IRQ: 8
Device is disabled

l3stat1 Cobham Gaisler LEON3 Statistics Unit
APB: 94001000 - 94001400
Device is disabled

gpreg3 Cobham Gaisler General Purpose Register
APB: 94002000 - 94002100

gmon3> info reg
AHB Memory Scrubber
0xff00000 AHB Status register 0x00000000
0xff00004 AHB Failing address register 0x00000000
0xff00008 AHB Error configuration register 0x00000003
0xff00010 Scrubber status register 0x00000000
0xff00014 Scrubber configuration register 0x00000000
0xff00018 Scrubber 1st low address register 0x00000000
0xff0001c Scrubber 1st high address register 0x00000000
0xff00020 Scrubber position register 0x00000000
0xff00024 Scrubber error threshold register 0x00000000
0xff00028 Scrubber initialization data register 0x00000000
0xff0002c Scrubber 2nd low address register 0x00000000
0xfff00030 Scrubber 2nd high address register 0x00000000

Dual-port AHB/(CPU) On-Chip RAM
0x80000000 Control register 0xf0600001
0x80000004 Scrubber data register 0x00000000
0x80000008 Scrubber control 1 register 0x00000000
0x8000000c Scrubber control 2 register 0xff000000

Dual-port AHB/(CPU) On-Chip RAM
0x8000b000 Control register 0x90700001
0x8000b004 Scrubber data register 0x00000000
0x8000b008 Scrubber control 1 register 0x00010000
0x8000b00c Scrubber control 2 register 0xff000000

LEON3 Debug Support Unit
0x90000000 CPU 0 Control register 0x000000ef
0x90000018 CPU 0 Floating-Point State Register 0x00060000
0x90000044 CPU 0 LEON3/4 configuration register 0x06800d9e
0x90000050 CPU 0 Alternative window register 0x001e0000
0x90000058 CPU 0 Trap register 0x00000000

AMBA Trace Buffer
0x9ff20000 Trace buffer control register 0x00008000
0x9ff20004 Trace buffer index register 0x0000007f
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.
0x80000000 Interrupt level register 0x00000000
0x80000004 Interrupt pending register 0x00000000
0x80000008 Interrupt force register 0x00000000
0x80000010 Interrupt status register 0x0c010000
0x80000014 Interrupt error mode status register 0x00000000
0x80000020 Interrupt mask register 0 0x00000000
0x80000024 Interrupt extended ack. register 0 0x00000000
0x80000028 Interrupt boot register 0x00000000
0x8000002c Interrupt interrupt map register 0 0x00010023
0x80000030 Interrupt interrupt map register 1 0x04050607
0x80000034 Interrupt interrupt map register 2 0x08090a0b
0x80000038 Interrupt interrupt map register 3 0x0c0d0e0f
0x8000003c Interrupt interrupt map register 4 0x10111213
0x80000040 Interrupt interrupt map register 5 0x14151617
0x80000044 Interrupt interrupt map register 6 0x18191a1b
0x80000048 Interrupt interrupt map register 7 0x1c1d1e1f
0x8000004c Interrupt interrupt map register 8 0x10111213
0x80000050 Interrupt interrupt map register 9 0x14151617
0x80000054 Interrupt interrupt map register 10 0x18191a1b
0x80000058 Interrupt interrupt map register 11 0x1c1d1e1f
0x8000005c Interrupt interrupt map register 12 0x0c0d0e0f
0x80000060 Interrupt interrupt map register 13 0x02141516
0x80000064 Interrupt interrupt map register 14 0x1718191a
0x80000068 Interrupt interrupt map register 15 0x10111213

Modular Timer Unit
0x80003000 Scalar value register 0x00000000
0x80003004 Scalar reload value register 0x00000000
0x80003008 Configuration register 0x00000000
<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8000300c</td>
<td>Latch configuration register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003010</td>
<td>Timer 0 Value register</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>0x80003014</td>
<td>Timer 0 Reload value register</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>0x80003018</td>
<td>Timer 0 Control register</td>
<td>0x00000043</td>
</tr>
<tr>
<td>0x8000301c</td>
<td>Timer 0 Latch register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003020</td>
<td>Timer 1 Value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003024</td>
<td>Timer 1 Reload value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003028</td>
<td>Timer 1 Control register</td>
<td>0x00000040</td>
</tr>
<tr>
<td>0x8000302c</td>
<td>Timer 1 Latch register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003030</td>
<td>Timer 2 Value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003034</td>
<td>Timer 2 Reload value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003038</td>
<td>Timer 2 Control register</td>
<td>0x00000040</td>
</tr>
<tr>
<td>0x8000303c</td>
<td>Timer 2 Latch register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003040</td>
<td>Timer 3 Value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003044</td>
<td>Timer 3 Reload value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003048</td>
<td>Timer 3 Control register</td>
<td>0x00000040</td>
</tr>
<tr>
<td>0x8000304c</td>
<td>Timer 3 Latch register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003050</td>
<td>Timer 4 Value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003054</td>
<td>Timer 4 Reload value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003058</td>
<td>Timer 4 Control register</td>
<td>0x00000040</td>
</tr>
<tr>
<td>0x8000305c</td>
<td>Timer 4 Latch register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003060</td>
<td>Timer 5 Value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003064</td>
<td>Timer 5 Reload value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003068</td>
<td>Timer 5 Control register</td>
<td>0x00000040</td>
</tr>
<tr>
<td>0x8000306c</td>
<td>Timer 5 Latch register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80003070</td>
<td>Timer 6 Value register</td>
<td>0x0006fc49</td>
</tr>
<tr>
<td>0x80003074</td>
<td>Timer 6 Reload value register</td>
<td>0x0006fc49</td>
</tr>
<tr>
<td>0x80003078</td>
<td>Timer 6 Control register</td>
<td>0x00000040</td>
</tr>
<tr>
<td>0x8000307c</td>
<td>Timer 6 Latch register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004000</td>
<td>Scalar value register</td>
<td>0x00000013</td>
</tr>
<tr>
<td>0x80004004</td>
<td>Scalar reload value register</td>
<td>0x00000013</td>
</tr>
<tr>
<td>0x80004008</td>
<td>Configuration register</td>
<td>0x000001af</td>
</tr>
<tr>
<td>0x8000400c</td>
<td>Latch configuration register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004010</td>
<td>Timer 0 Value register</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>0x80004014</td>
<td>Timer 0 Reload value register</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>0x80004018</td>
<td>Timer 0 Control register</td>
<td>0x00000043</td>
</tr>
<tr>
<td>0x8000401c</td>
<td>Timer 0 Latch register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004020</td>
<td>Timer 1 Value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004024</td>
<td>Timer 1 Reload value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004028</td>
<td>Timer 1 Control register</td>
<td>0x00000040</td>
</tr>
<tr>
<td>0x8000402c</td>
<td>Timer 1 Latch register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004030</td>
<td>Timer 2 Value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004034</td>
<td>Timer 2 Reload value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004038</td>
<td>Timer 2 Control register</td>
<td>0x00000040</td>
</tr>
<tr>
<td>0x8000403c</td>
<td>Timer 2 Latch register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004040</td>
<td>Timer 3 Value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004044</td>
<td>Timer 3 Reload value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004048</td>
<td>Timer 3 Control register</td>
<td>0x00000040</td>
</tr>
<tr>
<td>0x8000404c</td>
<td>Timer 3 Latch register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004050</td>
<td>Timer 4 Value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004054</td>
<td>Timer 4 Reload value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004058</td>
<td>Timer 4 Control register</td>
<td>0x00000040</td>
</tr>
<tr>
<td>0x8000405c</td>
<td>Timer 4 Latch register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004060</td>
<td>Timer 5 Value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004064</td>
<td>Timer 5 Reload value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004068</td>
<td>Timer 5 Control register</td>
<td>0x00000040</td>
</tr>
<tr>
<td>0x8000406c</td>
<td>Timer 5 Latch register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004070</td>
<td>Timer 6 Value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004074</td>
<td>Timer 6 Reload value register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80004078</td>
<td>Timer 6 Control register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8000407c</td>
<td>Timer 6 Latch register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80006000</td>
<td>Unlock register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80006004</td>
<td>Clock enable register</td>
<td>0x7ffeffffff</td>
</tr>
<tr>
<td>0x80006008</td>
<td>Reset register</td>
<td>0x7ffeffffff</td>
</tr>
<tr>
<td>0x80006000</td>
<td>Unlock register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80006004</td>
<td>Clock enable register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80006008</td>
<td>Reset register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x80008000</td>
<td>Boot ROM configuration register</td>
<td>0x000bc001</td>
</tr>
<tr>
<td>0x80008004</td>
<td>Memory test configuration register</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8000a000</td>
<td>Status register</td>
<td>0x00000002</td>
</tr>
<tr>
<td>0x8000a004</td>
<td>Failing address register</td>
<td>0x00000040</td>
</tr>
<tr>
<td>0x8000d000</td>
<td>System IO register for GPIO 0 to 7</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8000d004</td>
<td>System IO register for GPIO 8 to 15</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8000d008</td>
<td>System IO register for GPIO 16 to 23</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8000d00c</td>
<td>System IO register for GPIO 24 to 31</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8000d010</td>
<td>System IO register for GPIO 32 to 39</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x8000d014</td>
<td>System IO register for GPIO 40 to 47</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>
### System IO registers for GPIO 48 to 55
- 0x8000d018: 0x00000000

### System IO registers for GPIO 56 to 63
- 0x8000d01c: 0x00000000

### Pullup registers for GPIO 0 to 31
- 0x8000d020: 0x00000000

### Pullup registers for GPIO 32 to 63
- 0x8000d024: 0x00000000

### Pulldown registers for GPIO 0 to 31
- 0x8000d028: 0xffffffff

### Pulldown registers for GPIO 32 to 63
- 0x8000d02c: 0xffffffff

### IO configuration for LVDS
- 0x8000d030: 0x00888888

### Configuration protection register
- 0x8000d034: 0x00000000

### Config register error interrupt
- 0x8000d038: 0x00000000

### Config register error status
- 0x8000d03c: 0x00000000

### General Purpose Register
- 0x8000e000: 0x00000000

### GR716 AMBA Protection unit
- 0x8010a000: 0x04000000

### Protection Control Register
- 0x8010a004: 0x00000000

### Segment 0 start
- 0x8010a008: 0x00000000

### Segment 0 end
- 0x8010a00c: 0x00000000

### Segment 0 Config register
- 0x8010a014: 0x00000000

### Segment 1 start
- 0x8010a018: 0x00000000

### Segment 1 end
- 0x8010a01c: 0x00000000

### Segment 1 Config register
- 0x8010a024: 0x00000000

### Segment 2 start
- 0x8010a028: 0x00000000

### Segment 2 end
- 0x8010a02c: 0x00000000

### Segment 2 Config register
- 0x8010a034: 0x00000000

### Segment 3 start
- 0x8010a038: 0x00000000

### Segment 3 end
- 0x8010a03c: 0x00000000

### Segment 3 Config register
- 0x8010a040: 0x00000000

### GR716 Phase-locked loop
- 0x8010d000: 0x80000000

### PLL configuration registers
- 0x8010d004: 0x00000002

### PLL status registers
- 0x8010d008: 0x00000300

### PLL clock reference registers
- 0x8010d00c: 0x00000302

### SpaceWire clock reference registers
- 0x8010d010: 0x00000000

### System clock reference registers
- 0x8010d014: 0x00000000

### Select system clock source
- 0x8010d018: 0x00000000

### Clock protection registers
- 0x8010d020: 0x00000000

### PLL clock protection registers
- 0x8010d024: 0x00000000

### Select reference for PWMI clock
- 0x8010d028: 0x00000000

### Select reference for PWMM clock
- 0x8010d02c: 0x00000000

### Generic UART
- 0x80300004: 0x00000000

### UART Status register
- 0x80300008: 0x00000000

### UART Control register
- 0x8030000c: 0x00000000

### UART Scaler register
- 0x80300010: 0x00000000

### AHB Status Register
- 0x80306000: 0x00000000

### Status register
- 0x80306004: 0x00000000

### Failing address register
- 0x80306008: 0x00000000

### ADC / DAC Interface
- 0x80308000: 0x00000000

### Configuration register
- 0x80308004: 0x00000000

### Status register
- 0x80308008: 0x00000000

### ADC data input register
- 0x80308010: 0x00000000

### DAC data output register
- 0x80308014: 0x00000000

### Address input register
- 0x80308020: 0x00000000

### Address output register
- 0x80308024: 0x00000000

### Address direction register
- 0x80308028: 0x00000000

### Data input register
- 0x80308030: 0x00000000

### Data output register
- 0x80308034: 0x00000000

### Data direction register
- 0x80308038: 0x00000000

### General Purpose I/O port
- 0x8030c000: 0x00000000

### I/O port data register
- 0x8030c004: 0x00000000

### I/O port output register
- 0x8030c008: 0x00000000

### I/O port direction register
- 0x8030c00c: 0x00000000

### I/O interrupt mask register
- 0x8030c010: 0x00000000

### I/O interrupt polarity register
- 0x8030c014: 0x00000000

### I/O interrupt edge register
- 0x8030c018: 0x00000000

### I/O capability register
- 0x8030c020: 0x00000000

### I/O interrupt map register
- 0x8030c024: 0x00000000

### I/O interrupt map register 1
- 0x8030c028: 0x00000000

### I/O interrupt map register 2
- 0x8030c02c: 0x00000000

### I/O interrupt map register 3
- 0x8030c030: 0x00000000

### I/O interrupt map register 4
- 0x8030c034: 0x00000000

### I/O interrupt map register 5
- 0x8030c038: 0x00000000

### I/O interrupt map register 6
- 0x8030c03c: 0x00000000

### I/O interrupt map register 7
- 0x8030c040: 0x00000000

### I/O interrupt available register
- 0x8030c044: 0x00000000

### I/O interrupt flag register
- 0x8030c048: 0x00000000

### Input enable register
- 0x8030c04c: 0x00000000

### Pulse Register
- 0x8030c050: 0x00000000

### General Purpose I/O port
- 0x8030d000: 0x00000000

### I/O port data register
- 0x8030d004: 0x00000000

### I/O port output register
- 0x8030d008: 0x00000000

### I/O port direction register
- 0x8030d00c: 0x00000000

### I/O interrupt mask register
- 0x8030d010: 0x00000000

### I/O interrupt polarity register
- 0x8030d014: 0x00000000

### General Purpose I/O port
- 0x8030d000: 0x00000000

### I/O port data register
- 0x8030d004: 0x00000000

### I/O port output register
- 0x8030d008: 0x00000000

### I/O port direction register
- 0x8030d00c: 0x00000000

### I/O interrupt mask register
- 0x8030d010: 0x00000000

### I/O interrupt polarity register
- 0x8030d014: 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  Cobham Gaisler  Generic UART
       APB: 80300000 - 80300100
       IRQ: 24
       Baudrate 38461, FIFO debug mode available

   gptimer0  Cobham 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  HMASK 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  | PMRX     | PacketWire RX Interface | 0          |    0    |   1   |
|   3  | PMTX     | PacketWire TX Interface | 0          |    0    |   1   |
|   4  | FTMCTRL  | Memory ctrl with EDAC | 0          |    0    |   1   |
|   5  | SPI0CTRL | SPI Memory Controller 0 | 0          |    0    |   1   |
|   6  | SPI1CTRL | SPI Memory Controller 1 | 0          |    0    |   1   |
|   7  | SPI2CTRL | SPI Controller 0 | 0         |    0    |   1   |
+-----------------+-------------------+-----------------+-------------------+-------------------+
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.

<table>
<thead>
<tr>
<th>Program version</th>
<th>License</th>
<th>Supported hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRMON3 professional</td>
<td>Professional</td>
<td>• All GR716 systems</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• All LEON/GRLIB systems</td>
</tr>
<tr>
<td>GRMON3 professional</td>
<td>GR716</td>
<td>• All GR716 systems</td>
</tr>
<tr>
<td>GRMON3 evaluation</td>
<td>Evaluation</td>
<td>• No cost</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• No registration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GR716-MINI</td>
</tr>
</tbody>
</table>

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
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
<th>Core name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000000</td>
<td>Memory controller with EDAC</td>
<td>mctrl0</td>
</tr>
<tr>
<td>0x80002000</td>
<td>Multi-processor Interrupt Ctrl</td>
<td>irqmp0</td>
</tr>
<tr>
<td>0x80003000</td>
<td>Modular Timer Unit0</td>
<td>gptimer0</td>
</tr>
<tr>
<td>0x80004000</td>
<td>Modular Timer Unit1</td>
<td>gptimer1</td>
</tr>
<tr>
<td>0x81000000</td>
<td>GRSPW2 Spacewire Link 0</td>
<td>grspw0</td>
</tr>
<tr>
<td>0x81002000</td>
<td>CAN Controller with DMA 0</td>
<td>grcan0</td>
</tr>
<tr>
<td>0x81003000</td>
<td>CAN Controller with DMA 1</td>
<td>grcan1</td>
</tr>
<tr>
<td>0x80300000</td>
<td>Generic APB UART 0</td>
<td>uart0</td>
</tr>
<tr>
<td>0x80301000</td>
<td>Generic APB UART 1</td>
<td>uart1</td>
</tr>
<tr>
<td>0x80302000</td>
<td>Generic APB UART 2</td>
<td>uart2</td>
</tr>
<tr>
<td>0x80303000</td>
<td>Generic APB UART 3</td>
<td>uart3</td>
</tr>
<tr>
<td>0x80304000</td>
<td>Generic APB UART 4</td>
<td>uart4</td>
</tr>
<tr>
<td>0x80305000</td>
<td>Generic APB UART 5</td>
<td>uart5</td>
</tr>
<tr>
<td>0x80306000</td>
<td>AHB status register 0</td>
<td>ahhstatus0</td>
</tr>
<tr>
<td>0x80309000</td>
<td>SPI Controller 0</td>
<td>spi0</td>
</tr>
<tr>
<td>0x8030a000</td>
<td>SPI Controller 1</td>
<td>spi1</td>
</tr>
<tr>
<td>0x8030c000</td>
<td>General purpose I/O port 0</td>
<td>gpio0</td>
</tr>
<tr>
<td>0x8030d000</td>
<td>General purpose I/O port 1</td>
<td>gpio1</td>
</tr>
<tr>
<td>0x80408000</td>
<td>GR716 Digital-to-Analog Converter 0</td>
<td>dac0</td>
</tr>
<tr>
<td>0x80409000</td>
<td>GR716 Digital-to-Analog Converter 1</td>
<td>dac1</td>
</tr>
<tr>
<td>0x8040a000</td>
<td>GR716 Digital-to-Analog Converter 2</td>
<td>dac2</td>
</tr>
<tr>
<td>0x8040b000</td>
<td>GR716 Digital-to-Analog Converter 3</td>
<td>dac3</td>
</tr>
<tr>
<td>0xff000000</td>
<td>SPI Memory Controller 0</td>
<td>spim0</td>
</tr>
<tr>
<td>0xff002000</td>
<td>SPI Memory Controller 1</td>
<td>spim1</td>
</tr>
</tbody>
</table>

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'.
<table>
<thead>
<tr>
<th>Register</th>
<th>Register description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASR16</td>
<td>LEONFT register file prot. reg</td>
<td>0x00000000</td>
</tr>
<tr>
<td>ASR17</td>
<td>Processor config register</td>
<td>0x0000001e</td>
</tr>
<tr>
<td>ASR20</td>
<td>Alternative window register</td>
<td>0x001e0000</td>
</tr>
<tr>
<td>ASR22</td>
<td>Up counter MSB</td>
<td>0x00000000</td>
</tr>
<tr>
<td>ASR23</td>
<td>Up counter LSB</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

tsim> info reg uart0
Generic APB UART 0
0x80300000 UART Data register 0x00000000
0x80300004 UART Status register 0x00000086
0x80300008 UART Control register 0x00000000
0x8030000c UART Scaler register 0x00000000
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
wvmem                      Write word(s) to a virtual 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 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>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-g</code></td>
<td>generate debugging information - recommended for debugging with GDB</td>
</tr>
<tr>
<td><code>-msoft-float</code></td>
<td>emulate floating-point - must be used if no FPU exists in the system</td>
</tr>
<tr>
<td><code>-O2</code></td>
<td>optimize for speed</td>
</tr>
<tr>
<td><code>-Os</code></td>
<td>optimize for size</td>
</tr>
<tr>
<td><code>-Og</code></td>
<td>optimize for debugging experience</td>
</tr>
<tr>
<td><code>-qsvt</code></td>
<td>use the single-vector trap model</td>
</tr>
<tr>
<td><code>-mflat</code></td>
<td>enable flat register window model. The compiler will not emit SAVE and RESTORE instructions.</td>
</tr>
</tbody>
</table>

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
GRMON LEON debug monitor v3.0.15 64-bit pro version

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

grmon3> load hello.elf
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, Cobham 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 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.

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

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`.

```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.

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

int main(void)
{
```
```c
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

```bash
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:

```c
#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

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

#### 8.4.2.3. Run

```bash
gromon3> load bin/clkgate.elf
gromon3> 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.

gromon3>

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

gromon3> 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
 g = 00000000
 en = 0
SEGMENT 1
 start = 00000000
 g = 00000000
 en = 0
SEGMENT 2
 start = 00000000
 g = 00000000
 en = 0
SEGMENT 3
 start = 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>

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.

NOTE: 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 errors.

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
    0x80000000 Interrupt test configuration register  0x00000800
    20:18 vref  0x0 Enable external voltage reference
    17:16 spw  0x0 SpaceWire loopback production test
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

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 ce_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
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.

```
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.

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
```
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> set gpreg1::gpio7::gpio62 1
8.4.6.5.1. Build
8.4.6.5.2. Run
```

```
8.4.6.5.3. 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.
```

```
8.4.7. SPI master controller
8.4.7.1. Preparation
8.4.7.1.1. External SPI bus
8.4.7.2. Build
8.4.7.3. Run
```

```
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 appropriately.

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
```

```
```
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

```bash
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.

```bash
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

```bash
cd $BCC/src/libdrv/examples/grspw
```
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> grcq enable 16 grcq0
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 GRSPW: DMA Started Successfully
Starting Packet processing loop. Enter Command:

grspw> info
grspw: --- 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
grspw: --- 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
grspw: --- receive begin ---
GRSPW0: Prepared 38 RX packet buffers for future reception
grspw: --- 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
grspw: --- receive begin ---
GRSPW0: Received 1 packets
PKT of length 32 bytes, 0x01 0x9b 0x00 0x00 0x00 0x00 0x00 0x00 0x00
X0: scheduling packet on GRSPW0
GRSPW0: Sending 1 packets
PKT of length 32 bytes, 0x01 0x9b 0x00 0x00 0x00 0x00 0x00 0x00 0x00

grspw> q
EXAMPLE COMPLETED.

Program exited normally.
```

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
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.

```c
#include
groen3> grcg enable 5 grcgl
groen3> grcg enable 6 grcgl
#include
grroen3> forward enable uart0
grroen3> load bin/trans.elf
grroen3> run
GRCAN example begin
Found 2 GRCAN cores
grcan0: opened
grcan1: opened
# uart0
groen3> enable 16
groen3> forward enable
grmon3> load bin/trans.elf
groen3> 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:
MG: STD length: 6, id: 0x1
MGDATA: 0x61 0x62 0x63 0x31 0x32 0x33 abc123

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

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

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

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

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

```bash
cd $EXAMPLE/adc
```
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.

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
30:0 cfg 0x0 PLL configuration
0x8010d004 PLL status registers 0x00000002
1 ll 0x1 PLL lost lock
0 cl 0x0 PLL clock lock
NOTE: 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

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bchfile.tcl</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>• Input: Binary data</td>
</tr>
<tr>
<td></td>
<td>• Output: Binary data or ELF</td>
</tr>
<tr>
<td>img-gr716.tcl</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>• Input: ELF file</td>
</tr>
</tbody>
</table>
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 example 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 spin flash load and the TSIM command load.

NOTE: 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.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello.c</td>
<td>Application main() function which prints hello, world.</td>
</tr>
</tbody>
</table>
File | Description
--- | ---
`init60.c` | BCC run-time init hook `__bcc_init60()` which configures
  - UART clock gating
  - UART pins

Makefile | Build script

### 8.6.3.1. Build

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

The make command automatically runs the scripts described above:

```bash
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:

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

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%
```
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.

```bash
$ ./tsim-leon3 -gr716 -bootstrap 0x0000c002 -u 3
... 
```

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.

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

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

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

# 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

|---------------|-----------------------------------------------|--------------------------------------------------------------------------------------------------|
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

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

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

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

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

<table>
<thead>
<tr>
<th>Item</th>
<th>Default</th>
<th>Description of the default configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP1</td>
<td>1-2</td>
<td>Power board via USB</td>
</tr>
</tbody>
</table>

Table B.2. Bootstrap configuration

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

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.