Quick Start Guide for GR712RC-BOARD
# Table of Contents

1. Introduction ..................................................................................................................... 4  
   1.1. Overview .................................................................................................................. 4  
   1.2. References ............................................................................................................. 4  
2. Board Configuration ....................................................................................................... 5  
   2.1. Overview ................................................................................................................. 5  
   2.2. Clock Sources ....................................................................................................... 5  
   2.3. I/O Switch Matrix ................................................................................................. 6  
   2.4. UART ...................................................................................................................... 7  
   2.5. PROM ...................................................................................................................... 7  
3. Software Development Environment ............................................................................. 8  
   3.1. Overview ................................................................................................................. 8  
   3.2. Boot Loaders ......................................................................................................... 8  
   3.3. Software Drivers .................................................................................................. 9  
4. GRMON hardware debugger ......................................................................................... 10  
   4.1. Overview ................................................................................................................. 10  
   4.2. Debug-link alternatives ....................................................................................... 10  
      4.2.1. Connecting via the FTDI USB/JTAG interface ............................................ 10  
      4.2.2. Connecting via SpaceWire RMAP interface .............................................. 10  
   4.3. First steps ............................................................................................................... 10  
   4.4. Connecting to the board ...................................................................................... 11  
5. TSIM LEON simulator .................................................................................................. 18  
   5.1. Overview ................................................................................................................. 18  
   5.2. Startup .................................................................................................................... 18  
6. Toolchains ..................................................................................................................... 21  
   6.1. Bare C Cross-Compiler System ............................................................................ 21  
      6.1.1. Overview ......................................................................................................... 21  
      6.1.2. Compiling with BCC ..................................................................................... 21  
      6.1.3. Running and debugging with GRMON ......................................................... 21  
      6.1.4. Running and debugging with TSIM ............................................................. 22  
   6.2. RTEMS Real Time Operating System ................................................................. 23  
      6.2.1. Overview ......................................................................................................... 23  
      6.2.2. Installing RCC .............................................................................................. 23  
      6.2.3. Building an RTEMS sample application ..................................................... 23  
      6.2.4. Running and debugging with GRMON ......................................................... 24  
   6.3. VxWorks ................................................................................................................ 25  
      6.3.1. Overview ......................................................................................................... 25  
   6.4. MKPROM2 ............................................................................................................. 25  
      6.4.1. Overview ......................................................................................................... 25  
      6.4.2. Usage of MKPROM2 ..................................................................................... 25  
7. Frequently Asked Questions / Common Mistakes / Know Issues ................................ 27  
   7.1. GR712RC ................................................................................................................ 27  
      7.1.1. Clock gating ..................................................................................................... 27  
      7.1.2. GRMON issues .............................................................................................. 27  
      7.1.3. GPIO controller does not remember interrupt requests ............................... 27  
      7.1.4. Multiprocessor & legacy support .................................................................. 27  
      7.1.5. Inter-processor interrupts ............................................................................. 27  
      7.1.6. Interrupt considerations ................................................................................ 27  
      7.1.7. GRMON Debug Link Limitations .................................................................. 28  
      7.1.8. MIL-1553 ........................................................................................................ 28  
      7.1.9. CAN multiplexing ........................................................................................... 28  
      7.1.10. Concurrent CAN and Ethernet ..................................................................... 29  
      7.1.11. Hardware behavior at CPU reset and power management ....................... 29  
   7.2. GR712RC-BOARD ................................................................................................ 30  
      7.2.1. Clock problems .............................................................................................. 30  
      7.2.2. Switch Matrix Configuration Problems ....................................................... 30
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.3.</td>
<td>GPIO used as configuration at reset</td>
<td>30</td>
</tr>
<tr>
<td>7.2.4.</td>
<td>SDRAM configuration</td>
<td>30</td>
</tr>
<tr>
<td>8.</td>
<td>Support</td>
<td>31</td>
</tr>
</tbody>
</table>
1. Introduction

1.1. Overview

This document is a quick start guide for the GR712RC Development Board. The purpose of this document is to get users quickly started using the board.


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

| RD-6 | RTEMS homepage [https://www.rtems.org] |
| RD-7 | LEON/ERC32 RTEMS Cross Compilation System (RCC) [https://www.gaisler.com/index.php/products/operating-systems/rtems] |
| RD-9 | Cobham Gaisler RTEMS driver documentation [https://gaisler.com/anonftp/rcc/doc] |
| RD-10 | Bare C Cross-Compilation System [https://www.gaisler.com/index.php/products/operating-systems/bcc] |
| RD-12 | VxWorks 7 SPARC architectural port and BSP [https://www.gaisler.com/index.php/products/operating-systems/vxworks-7] |

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

2.1. Overview

The primary source of information for board configuration is the GR712RC Development Board User Manual. The board requires some hardware configuration to fit with the customer requirements. In particular, the number of the GR712RC-BOARD’s processor I/O pins limits the simultaneously available connections to external interfaces. To overcome this limitation, the SoC features an internal switch matrix, and a set of jumpers must be configured accordingly to route the signals to the appropriate headers on the board. The internal switch matrix is configured by enabling the respective interfaces via software. Additionally, clock selection might need to be configured by a set of jumpers and possibly the insertion of custom oscillators.

![GR712RC-BOARD default configuration as delivered](image)

2.2. Clock Sources

The minimum requirement in order for the board to work and to be able to connect to it, is that the clock sources are properly configured. The 80 MHz oscillator in socket X2 provided by default with the board is connected to
the system clock input through the JP84 jumper in the default configuration 2-3. The on-board soldered 48 MHz oscillator can be used instead by positioning the JP84 jumper on pins 1-2. Alternatively a custom oscillator can be installed in X2.

The SpaceWire clock is, by default, driven by an on board additional 100 MHz oscillator. If the user wants to use the system clock configured in the paragraph above as the source of the SpaceWire clock, then jumper JP88 must be inserted and the oscillator in socket X5 must be removed.

Refer to Section 2.14 of [RD-1] for further information about oscillators and clock inputs and more information about the system and SpaceWire clock.

Once the external clock sources are selected, further clock configuration can be done in software. The SpaceWire external clock source can be used as 1X, 2X or 4X, or the external system clock can be used in its place. This selection is done by configuring the SoC's General Purpose Register (GPREG). At reset the 1X SpaceWire clock received from the board is used internally.

For in depth information about configuring the SpaceWire and MIL-STD-1553 clocks through the GPREG, please refer to Chapter 3 and Chapter 13 of [RD-2].

### 2.3. I/O Switch Matrix

To overcome the limitation on the number of SoC pins, an internal switch matrix selects the input/output signals to connect to the pad. Additionally the chip I/O pins are connected to the board’s I/O ports through an array of jumpers. One UART and two SpaceWire interfaces are routed independently of the internal switch matrix and the jumpers JP3 through JP66. In the default position A of jumpers JP3 through JP66, all multiplexed switch matrix signals are connected to the board's GPIO pins.

Six basic example configurations are provided to respond to typical use cases, as seen in Table 2.1. To use one of these configurations, the user has to insert jumpers JP3 through JP66 in the position described in the table. Refer to [RD-1] and GR712RC Development Board Schematic for more information on signal and GPIO configuration.

#### Table 2.1. Typical configurations

<table>
<thead>
<tr>
<th>Cfg. description</th>
<th>I/O enabled</th>
<th>Jumper position</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU for GEO applications</td>
<td>UART0, UART1, UART2, UART3, UART4, UART5 SpaceWire-0, SpaceWire-1, SpaceWire-2, SpaceWire-3, SpaceWire-4, SpaceWire-5 Mil-Std-1553-A, Mil-Std-1553-B SPI I2C</td>
<td>B</td>
</tr>
<tr>
<td>CPU for TMTC applications</td>
<td>UART0, UART1, UART2, UART3 SpaceWire-0, SpaceWire-1, SpaceWire-2, SpaceWire-3 SDRAM with optional Reed-Solomon CCSDS/ECSS TC &amp; TM</td>
<td>C</td>
</tr>
<tr>
<td>CPU for LEO applications</td>
<td>UART0, UART1, UART2, UART3, UART4, UART5 SpaceWire-0, SpaceWire-1 SDRAM with optional Reed-Solomon ASCS16 CAN-A, CAN-B SLINK I2C</td>
<td>D</td>
</tr>
<tr>
<td>Instrument Controller, type A</td>
<td>UART0, UART1, UART2, UART3, UART4, UART5 SpaceWire-0, SpaceWire-1 SDRAM with optional Reed-Solomon CAN-A, CAN-B SLINK I2C</td>
<td>E</td>
</tr>
</tbody>
</table>
Once the board's jumpers are properly connected, the internal switch matrix must be driven by a set of enabling conditions. It is important to note that to obtain a proper functioning system, the I/O interfaces of the required configurations have to be enabled or clock ungated by software. See Chapter 2 and Table 9 of [RD-2] for further details on the switch matrix.

The I/O matrix is not limited to these pre-defined configurations. Jumpers can be custom configured according to the user requirements. See Section 2.4 of [RD-1] for further details.

### 2.4. UART

Jumpers JP1 and JP2 are used to select the output standard of the UART0 and UART1 interfaces between RS232 and RS422, and to route the signals to the J1 and J16 connectors respectively. In the default configuration the interfaces are connected to the J1 connectors UART-0 and UART-1 using the RS232 standard. While UART0 is not affected by the internal switch matrix, UART1 Rx is multiplexed and JP3 must be set to 3-4 in order to use it. Refer to the GR712RC Development Board Schematic for more information on how to configure UART0 and UART1 to use the RS422 standard.

### 2.5. PROM

The PROM width and PROM EDAC conditions are set by the state of the GPIO[3] and GPIO[1] pins at power up of the Processor. These pins are provided with pull-down resistors to set the default mode to 8 bit with no EDAC. If EDAC operation of the Flash PROM is desired, then jumper JP85 should be installed, to pull-up GPIO[1].
3. Software Development Environment

3.1. Overview

Cobham Gaisler provides a comprehensive set of software tools to run several different operating systems. The GR712RC 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

- **RTEMS**
  a hard Real Time Operating System. Cobham Gaisler provides RCC, a toolchain to develop and compile RTEMS applications specifically for the LEON

- **Linux**
  the open source operating system. Board Support Packages and tools to ease the compilation and deployment of the kernel are provided

- **VxWorks**
  an embedded real-time operating system developed by WindRiver. Cobham Gaisler provides a LEON architectural port (HAL) and a Board Support Package (BSP) in full source code

Cobham Gaisler also provides a set of debug tools. The GR712RC platform is supported by the following:

- **GRMON**
  Used to run and debug applications on GR712RC-BOARD hardware. See (Chapter 4).

- **TSIM**
  Used to run and debug applications on a simulated GR712RC-BOARD. See (Chapter 5).

TSIM is mainly used when no hardware is available. However, TSIM also provides faster than realtime simulation and can 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 GRMON).

The recommended method to load software onto a LEON board is by connecting to a debug interface of the board through the GRMON hardware debugger (Chapter 4). Execution of programs by a PROM-loaded boot loader is also possible.

3.2. Boot Loaders

Cobham Gaisler provides three boot loaders for the ERC32, LEON2, LEON3 and LEON4 processors listed below for more information. The boot loaders covers different use cases and requirements on software quality level. The boot loaders are all capable of booting all the supported Operating Systems provided by Cobham Gaisler.

- **MKPROM2**
  MKPROM2 is a free open-source boot loader supporting a minimal system initialization, extraction of a single ROM application image into main memory and booting it. No system self-tests are performed by MKPROM2.

- **GR712RC Boot SW**
  The GR712RC Boot SW was specifically developed for the ESA JUICE satellite and will be used in several of its GR712RC based payloads on board following the requirements of ESA's Flight Computer Initialisation Sequence requirement document.
  
  It supports initialization, self-tests, a SpaceWire remote terminal as Standby Mode and CRC checking application loader handing over a boot report. The software was developed according to ESA's software engineering standards ECSS-E-ST-40C and ECSS-Q-ST-80C, software criticality category B, reviewed successfully by ESA and third party (ISV&V).

- **GRBOOT**
  The GRBOOT boot loader software is based on the GR712RC Boot SW using the same ECSS software engineering standards previously used to guarantee a high reliability for flight. By isolating mission and device specific parts into BSPs and generalizing the implementation, GRBOOT provides similar a reusable feature set for systems based on LEON3/4FT processor devices acting as either payload or OBC.
One or more application images can be located in parallel flash or SPI flash. Multiprocessor application booting is supported.

GRBOOT is available for GR712RC and GR740 based systems together with the appropriate quality proofs, documentation and test suites. A version without references to the ESA requirements documents is also available.

u-boot

Currently u-boot for the GR712RC Development Board is not provided by Cobham Gaisler.

Table 3.1. Boot Loader feature table

<table>
<thead>
<tr>
<th>Feature</th>
<th>MKPROM2</th>
<th>GR712RC Boot SW</th>
<th>GRBOOT</th>
</tr>
</thead>
</table>
| Supported processors                | • Most LEON | • GR712RC | • GR712RC • GR740
|                                      |         |                 | Additional system support in progress. |
| Processor self-tests                 | No      | Yes             | Yes    |
| Memory self-tests                    | No      | Yes             | Yes    |
| Application storage memory           | • PROM  | MRAM            | • PROM  • FLASH  • MRAM  • SPI flash |
|                                      | • FLASH |                 |        |
|                                      | • MRAM  |                 |        |
| Number of application images         | 1       | 2               | Unlimited |
| Standby Mode                         | No      | Yes             | Prepared for user extensions. |
|                                      |         | PUS over SpaceWire | PUS over SpaceWire in development. |
| Validation and unit test suite       | No      | Yes             | Yes    |
| Documentation covering SW requirements, design and quality | No | Yes | Yes |
| Compatible standards                 | None    | TEC-SWS/10-373, ECSS-E-70-41A | SAVOIR-GS-002 |

3.3. Software Drivers

The operating system environments include software drivers for most I/O units of the GR712RC. Cobham Gaisler license low-level software drivers listed below together with the infrastructure for qualification on GR712RC-BOARD. The drivers have been qualified for the JUICE GR712-DPU board. For more information please contact sales@gaisler.com.

- SpaceWire controller with DMA
- UART
- SPI master controller
- GPIO
- Timer
- AHB Status Register
- Clock Gating Unit
4. GRMON hardware debugger

4.1. Overview

GRMON 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. GRMON has GDB support which makes C/C++ level debugging possible by connecting GDB to the GRMON's GDB socket. With GRMON one can for example:

- Inspect 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 GR712RC Development Board. After that, a basic first inspection of the board is exemplified.

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

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

GR712RC can be used with GRMON version 2 or later. It is recommended to use version GRMON 3 or later with GR712RC.

4.2. Debug-link alternatives

4.2.1. Connecting via the FTDI USB/JTAG interface

Please see GRMON User’s Manual for how to set up the required FTDI driver software. Then connect the PC and the board using a standard USB cable into the USB-mini J12 USB-JTAG connector and issue the following command:

```
grmon -ftdi
```

4.2.2. Connecting via SpaceWire RMAP interface

GRMON has support for connecting to boards with SpaceWire interfaces as long as the SpaceWire has RMAP and automatic link start. An Ethernet to SpaceWire bridge (GRESB) is required to tunnel SpaceWire packets from the Ethernet network over to SpaceWire.

Please see the [RD-4] for information about connecting through a GRESB and optional parameters. Connect the GRESB SpW0 connector and the GR712RC-BOARD’s J3 (SPW-0) or J4 (SPW-1) connector, then issue the following command:

```
grmon -gresb
```

4.3. First steps

The previous sections have described which debug-links are available and how to start using them with GRMON. The subsections below assume that GRMON, the host computer and the GR712RC-BOARD board have been set up so that GRMON can connect to the board.

When connecting to the board for the first time it is recommended to get to know the system by inspecting the current configuration and hardware present using GRMON. With the `info sys` command more details about the system is printed and with `info reg` the register contents of the I/O registers can be inspected. Below is a list of items of particular interest:

- AMBA system frequency is printed out at connect, if the frequency is wrong then it might be due to noise in auto detection (small error). See `-frequ` flag in the GRMON User's Manual [RD-4].
• Memory location and size configuration is found from the `info sys` output. If the board has both SRAM and SDRAM interfaces, SDRAM can be mapped at the SRAM base address using the `--nosram` option of GRMON. See the GRMON User’s Manual [RD-4] for further details.
• The GR712RC 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 GRMON will be using. The `grcg` command is described in the GRMON User’s Manual [RD-4].

4.4. Connecting to the board

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

daniel@daniel:~$ grmon -ftdi
GRMON2 LEON debug monitor v2.0.35 professional version
Copyright (C) 2012 Aeroflex Gaisler - All rights reserved.
For latest updates, go to http://www.gaisler.com/
Comments or bug-reports to support@gaisler.com

Parsing -ftdi
Commands missing help:
datacache

JTAG chain (1): GR712RC
Detected system: GR712RC
Detected frequency: 80 MHz

<table>
<thead>
<tr>
<th>Component</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEON3-FT SPARC V8 Processor</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>LEON3-FT SPARC V8 Processor</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>JTAG Debug Link</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>GR Ethernet MAC</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>SatCAN controller</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>GRSPW2 SpaceWire Serial Link</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>GRSPW2 SpaceWire Serial Link</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>GRSPW2 SpaceWire Serial Link</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>AMBA Wrapper for Core1553BRM</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>CCSDS Telecommand Decoder</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>CCSDS Telemetry Encoder</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>SLINK Master</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>Memory controller with EDAC</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>AHB/APB Bridge</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>LEON3 Debug Support Unit</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>AHB/APB Bridge</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>OC CAN AHB interface</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>Generic FT AHB SRAM module</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>Generic UART</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>Multi-processor Interrupt Ctrl.</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>Modular Timer Unit</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>SPI Controller</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>CAN Bus multiplexer</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>General Purpose Register</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>ASCS Master</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>General Purpose I/O port</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>General Purpose I/O port</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>AMBA Wrapper for OC I2C-master</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>Clock gating unit</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>AHB Status Register</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>Generic UART</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>Generic UART</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>Generic UART</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>Generic UART</td>
<td>Aeroflex Gaisler</td>
</tr>
<tr>
<td>Timer Unit with Latches</td>
<td>Aeroflex Gaisler</td>
</tr>
</tbody>
</table>

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

grmon2> info sys
cpu0  Aeroflex Gaisler  LEON3-FT SPARC V8 Processor
       AHB Master 0

cpu1  Aeroflex Gaisler  LEON3-FT SPARC V8 Processor
       AHB Master 1
ahbjtag0  Aeroflex Gaisler  JTAG Debug Link  
AHB Master 2  

greth0  Aeroflex Gaisler  GR Ethernet MAC  
AHB Master 3  
APB: 80000E00 - 80000F00  
IRQ: 14  

satcan0  Aeroflex Gaisler  SatCAN controller  
AHB Master 4  
AHB: FFF20000 - FFF20100  
IRQ: 14  

grspw0  Aeroflex Gaisler  GRSPW2 SpaceWire Serial Link  
AHB Master 5  
APB: 80100800 - 80100900  
IRQ: 22  
Number of ports: 1  

grspw1  Aeroflex Gaisler  GRSPW2 SpaceWire Serial Link  
AHB Master 6  
APB: 80100900 - 80100A00  
IRQ: 23  
Number of ports: 1  

grspw2  Aeroflex Gaisler  GRSPW2 SpaceWire Serial Link  
AHB Master 7  
APB: 80100A00 - 80100B00  
IRQ: 24  
Number of ports: 1  

grspw3  Aeroflex Gaisler  GRSPW2 SpaceWire Serial Link  
AHB Master 8  
APB: 80100B00 - 80100C00  
IRQ: 25  
Number of ports: 1  

grspw4  Aeroflex Gaisler  GRSPW2 SpaceWire Serial Link  
AHB Master 9  
APB: 80100C00 - 80100D00  
IRQ: 26  
Number of ports: 1  

grspw5  Aeroflex Gaisler  GRSPW2 SpaceWire Serial Link  
AHB Master 10  
APB: 80100D00 - 80100E00  
IRQ: 27  
Number of ports: 1  

b1553brm0  Aeroflex Gaisler  AMBA Wrapper for Core1553BRM  
AHB Master 11  
AHB: FFF00000 - FFF01000  
IRQ: 14  

grtc0  Aeroflex Gaisler  CCSDS Telecommand Decoder  
AHB Master 12  
AHB: FFF10000 - FFF10100  
IRQ: 14  

grtm0  Aeroflex Gaisler  CCSDS Telemetry Encoder  
AHB Master 13  
APB: 80000B00 - 80000C00  
IRQ: 29  

addev14  Aeroflex Gaisler  SLINK Master  
AHB Master 14  
APB: 80000800 - 80000900  
IRQ: 13  

mcrl0  Aeroflex Gaisler  Memory controller with EDAC  
AHB: 00000000 - 20000000  
AHB: 20000000 - 40000000  
AHB: 40000000 - 80000000  
AHB: 80000000 - 80000100  
8-bit prom @ 0x00000000  
32-bit static ram: 1 * 8192 kbyte @ 0x40000000  
32-bit sram: 2 * 128 Mbyte @ 0x60000000  
Col 10, cas 2, ref 7.8 us  

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

dsu0  Aeroflex Gaisler  LEON3 Debug Support Unit  
AHB: 90000000 - AD000000  
AHB trace: 256 lines, 32-bit bus  
CPU0: win 8, hwbp 2, ltrace 256, V8 mul/div, srmmu, lddel 1, GRFPFU  
stack pointer 0xa40'fff0  
icache 4 * 4 kB, 16 B/line 1ru  
dcache 4 * 4 kB, 16 B/line 1ru  
CPU1: win 8, hwbp 2, ltrace 256, V8 mul/div, srmmu, lddel 1, GRFPFU  
stack pointer 0xa40'fff0  
icache 4 * 4 kB, 32 B/line 1ru  
dcache 4 * 4 kB, 16 B/line 1ru  

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

occ0  Aeroflex Gaisler  OC CAN AHB interface  
AHB: FFF30000 - FFF31000  
IRQ: 5
cores: 2

ahbram0 Aeroflex Gaisler Generic FT AHB SRAM module
AHB: A0000000 - A0100000
APB: 80100000 - 80100100
32-bit static ram; 256 kB @ 0xa0000000

uart0 Aeroflex Gaisler Generic UART
APB: 80000100 - 80000200
IRQ: 2
Baudrate 38461

irqmp0 Aeroflex Gaisler Multi-processor Interrupt Ctrl.
APB: 80000200 - 80000300
EIRQ: 12

aptimer0 Aeroflex Gaisler Modular Timer Unit
APB: 80000300 - 80000400
IRQ: 8
16-bit scalar, 4 * 32-bit timers, divisor 48

spi0 Aeroflex Gaisler SPI Controller
APB: 80000400 - 80000500
IRQ: 13
FIFO depth: 16, no slave select lines
Maximum word length: 32 bits
Controller index for use in GRMON: 0

adev25 Aeroflex Gaisler CAN Bus multiplexer
APB: 80000500 - 80000600

grgprg0 Aeroflex Gaisler General Purpose Register
APB: 80000600 - 80000700

adev27 Aeroflex Gaisler ASCS Master
APB: 80000700 - 80000800
IRQ: 16

gpio0 Aeroflex Gaisler General Purpose I/O port
APB: 80000900 - 80000A00

gpio1 Aeroflex Gaisler General Purpose I/O port
APB: 80000A00 - 80000B00

i2cmst0 Aeroflex Gaisler AMBA Wrapper for OC I2C-master
APB: 80000C00 - 80000D00
IRQ: 28

grcg0 Aeroflex Gaisler Clock gating unit
APB: 80000D00 - 80000E00
GRMON did NOT enable clocks during initialization

ahbstat0 Aeroflex Gaisler AHB Status Register
APB: 80000F00 - 80001000
IRQ: 1

uart1 Aeroflex Gaisler Generic UART
APB: 80100100 - 80100200
IRQ: 17
Baudrate 38461

uart2 Aeroflex Gaisler Generic UART
APB: 80100200 - 80100300
IRQ: 18
Baudrate 38461

uart3 Aeroflex Gaisler Generic UART
APB: 80100300 - 80100400
IRQ: 19
Baudrate 38461

uart4 Aeroflex Gaisler Generic UART
APB: 80100400 - 80100500
IRQ: 20
Baudrate 38461

uart5 Aeroflex Gaisler Generic UART
APB: 80100500 - 80100600
IRQ: 21
Baudrate 38461

grtimer0 Aeroflex Gaisler Timer Unit with Latches
APB: 80100600 - 80100700
IRQ: 7
8-bit scalar, 2 * 32-bit timers, divisor 48

grmon2> info reg
GR Ethernet MAC
0x800000e0 Control register 0x04000080
0x800000e4 Status register 0x0000000a
0x800000e8 MAC address MSB 0x00000012
0x800000ec MAC address LSB 0x10844440
0x80000100 MDIO register 0x7849084a
0x80000140 Tx descriptor register 0x10004000
0x80000180 Rx descriptor register 0x80000000
0x800001c0 EDCL IP register 0x00000000

GRSPW2 SpaceWire Serial Link
0x80100000 Control register 0xa8010002
0x80100004 Status/Interrupt-source 0x00600000
0x80100008 Node address 0x0000000e
0x8010000c Clock divisor 0x00000000
0x80100010 Destination key 0x00000000
CCSDS Telemetry Encoder

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000b00</td>
<td>DMA control register</td>
</tr>
<tr>
<td>0x80000b04</td>
<td>DMA status register</td>
</tr>
<tr>
<td>0x80000b08</td>
<td>DMA length register</td>
</tr>
<tr>
<td>0x80000b0c</td>
<td>DMA descriptor pointer register</td>
</tr>
<tr>
<td>0x80000b14</td>
<td>DMA revision register</td>
</tr>
<tr>
<td>0x80000b80</td>
<td>Control register</td>
</tr>
<tr>
<td>0x80000b84</td>
<td>Status register</td>
</tr>
<tr>
<td>0x80000b88</td>
<td>Configuration register</td>
</tr>
<tr>
<td>0x80000b90</td>
<td>Physical layer register</td>
</tr>
<tr>
<td>0x80000b94</td>
<td>Coding sub-layer register</td>
</tr>
<tr>
<td>0x80000b98</td>
<td>Attached Synchronization Marker</td>
</tr>
<tr>
<td>0x80000ba0</td>
<td>All frames generation register</td>
</tr>
<tr>
<td>0x80000ba4</td>
<td>Master frame generation register</td>
</tr>
<tr>
<td>0x80000ba8</td>
<td>Idle frame generation register</td>
</tr>
<tr>
<td>0x80000bdc</td>
<td>OCF register</td>
</tr>
</tbody>
</table>

Memory controller with EDAC

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000000</td>
<td>Memory config register 1</td>
</tr>
<tr>
<td>0x80000004</td>
<td>Memory config register 2</td>
</tr>
<tr>
<td>0x80000008</td>
<td>Memory config register 3</td>
</tr>
</tbody>
</table>

LEON3 Debug Support Unit

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x90000024</td>
<td>Debug mode mask register</td>
</tr>
<tr>
<td>0x90000000</td>
<td>CPU 0 Control register</td>
</tr>
<tr>
<td>0x90100000</td>
<td>CPU 1 Control register</td>
</tr>
</tbody>
</table>

Generic FT AHB SRAM module

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80100000</td>
<td>Configuration Register</td>
</tr>
</tbody>
</table>

Generic UART

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000104</td>
<td>UART Status register</td>
</tr>
<tr>
<td>0x80000108</td>
<td>UART Control register</td>
</tr>
<tr>
<td>0x80000110</td>
<td>UART Scaler register</td>
</tr>
</tbody>
</table>

Multi-processor Interrupt Ctrl.

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000200</td>
<td>Interrupt level register</td>
</tr>
<tr>
<td>0x80000204</td>
<td>Interrupt pending register</td>
</tr>
<tr>
<td>0x80000210</td>
<td>Interrupt status register</td>
</tr>
<tr>
<td>0x80000220</td>
<td>Interrupt mask register 0</td>
</tr>
<tr>
<td>0x80000224</td>
<td>Interrupt mask register 1</td>
</tr>
<tr>
<td>0x80000280</td>
<td>Interrupt force register 0</td>
</tr>
<tr>
<td>0x80000284</td>
<td>Interrupt force register 1</td>
</tr>
</tbody>
</table>

Modular Timer Unit

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000300</td>
<td>Scalar value register</td>
</tr>
<tr>
<td>0x80000304</td>
<td>Scalar reload value register</td>
</tr>
<tr>
<td>0x80000308</td>
<td>Configuration register</td>
</tr>
<tr>
<td>0x80000310</td>
<td>Timer 0 Value register</td>
</tr>
<tr>
<td>0x80000314</td>
<td>Timer 0 Reload value register</td>
</tr>
<tr>
<td>0x80000318</td>
<td>Timer 0 Control register</td>
</tr>
<tr>
<td>0x80000320</td>
<td>Timer 1 Value register</td>
</tr>
<tr>
<td>0x80000324</td>
<td>Timer 1 Reload value register</td>
</tr>
<tr>
<td>0x80000328</td>
<td>Timer 1 Control register</td>
</tr>
<tr>
<td>0x80000330</td>
<td>Timer 2 Value register</td>
</tr>
<tr>
<td>0x80000334</td>
<td>Timer 2 Reload value register</td>
</tr>
<tr>
<td>0x80000338</td>
<td>Timer 2 Control register</td>
</tr>
<tr>
<td>0x80000340</td>
<td>Timer 3 Value register</td>
</tr>
<tr>
<td>0x80000344</td>
<td>Timer 3 Reload value register</td>
</tr>
<tr>
<td>0x80000348</td>
<td>Timer 3 Control register</td>
</tr>
</tbody>
</table>

SPI Controller

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000400</td>
<td>Capability register</td>
</tr>
<tr>
<td>0x80000420</td>
<td>Mode register</td>
</tr>
<tr>
<td>0x80000424</td>
<td>Event register</td>
</tr>
<tr>
<td>0x80000428</td>
<td>Mask register</td>
</tr>
<tr>
<td>0x80000430</td>
<td>Command register</td>
</tr>
<tr>
<td>0x80000434</td>
<td>Transmit register</td>
</tr>
<tr>
<td>0x80000438</td>
<td>Receive register</td>
</tr>
</tbody>
</table>

General Purpose Register

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000600</td>
<td>GR712RC general purpose register</td>
</tr>
</tbody>
</table>

General Purpose I/O

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000900</td>
<td>I/O port data register</td>
</tr>
<tr>
<td>0x80000904</td>
<td>I/O port output register</td>
</tr>
<tr>
<td>0x80000908</td>
<td>I/O port direction register</td>
</tr>
<tr>
<td>0x8000090c</td>
<td>I/O interrupt mask register</td>
</tr>
<tr>
<td>0x80000910</td>
<td>I/O interrupt polarity register</td>
</tr>
<tr>
<td>0x80000914</td>
<td>I/O interrupt edge register</td>
</tr>
<tr>
<td>0x80000918</td>
<td>I/O bypass register</td>
</tr>
</tbody>
</table>

General Purpose I/O

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000a00</td>
<td>I/O port data register</td>
</tr>
<tr>
<td>0x80000a04</td>
<td>I/O port output register</td>
</tr>
<tr>
<td>0x80000a08</td>
<td>I/O port direction register</td>
</tr>
<tr>
<td>0x80000a0c</td>
<td>I/O interrupt mask register</td>
</tr>
<tr>
<td>0x80000a10</td>
<td>I/O interrupt polarity register</td>
</tr>
<tr>
<td>0x80000a14</td>
<td>I/O interrupt edge register</td>
</tr>
<tr>
<td>0x80000a18</td>
<td>I/O bypass register</td>
</tr>
</tbody>
</table>

AMBA Wrapper for OC I2C-master
One can limit the output to certain cores by specifying the core(s) name(s) to the info sys and info reg commands. As seen below the memory parameters, first UART and first Timer core information is listed.

The GR712RC has a clock-gating unit which can disable and enable clock gating and generate reset signals of certain cores in the SOC. With the GRMON greg 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 GRETH Ethernet core's clocks are turned on (not gated).
<table>
<thead>
<tr>
<th>Gate</th>
<th>Core(s)</th>
<th>Description</th>
<th>Unlocked</th>
<th>Enabled</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GRETH</td>
<td>10/100 Ethernet MAC</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>GRSPW</td>
<td>Spacewire link 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>GRSPW</td>
<td>Spacewire link 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>GRSPW</td>
<td>Spacewire link 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>GRSPW</td>
<td>Spacewire link 3</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>GRSPW</td>
<td>Spacewire link 4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>GRSPW</td>
<td>Spacewire link 5</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>CAN</td>
<td>CAN core 1 &amp; 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>SatCAN</td>
<td>SatCAN controller</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>GRTM</td>
<td>Telemetry Encoder</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>GRTC</td>
<td>Telecommand Decoder</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>B1553BRM</td>
<td>MIL-STD-1553 BRM</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

grmon2> grcg enable 0

grmon2> grcg

GRCLKGATE GR712RC info:
Unlock register: 0x00000000
Clock enable register: 0x00000007
Reset register: 0x00000ff8

GR712RC decode of values:

<table>
<thead>
<tr>
<th>Gate</th>
<th>Core(s)</th>
<th>Description</th>
<th>Unlocked</th>
<th>Enabled</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GRETH</td>
<td>10/100 Ethernet MAC</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>GRSPW</td>
<td>Spacewire link 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>GRSPW</td>
<td>Spacewire link 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>GRSPW</td>
<td>Spacewire link 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>GRSPW</td>
<td>Spacewire link 3</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>GRSPW</td>
<td>Spacewire link 4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>GRSPW</td>
<td>Spacewire link 5</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>CAN</td>
<td>CAN core 1 &amp; 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>SatCAN</td>
<td>SatCAN controller</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>GRTM</td>
<td>Telemetry Encoder</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>GRTC</td>
<td>Telecommand Decoder</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>B1553BRM</td>
<td>MIL-STD-1553 BRM</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
5. TSIM LEON simulator

5.1. Overview

TSIM is a simulator that can emulate a single-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 GR712RC Development Board.

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

5.2. Startup

To start TSIM, use the command:

tsim-leon3 -gr712rc -ahbm gr712.so

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

tsim-leon3 -gr712rc -ahbm gr712.so -designinput module.so -designinputend

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

After TSIM has been started all simulated peripherals can be listed by using the leon command. To inspect the status of a core use one of the listed status commands. All TSIM commands can be listed with the help command.

```
tsim> leon

Address         Description               Status command
-------------    ------------------------   -----
0x80000000      Memory configurations     mctrl_status
0x80000200      Iromp                    iromp_status
0x80000300      GPTIMER0                  gptimer0_status
0x80010600      GRTIMER0                  grtimer0_status
0x80000100      APBUART0                  uart0_status
0x80100100      APBUART1                  uart1_status
0x80100200      APBUART2                  uart2_status
0x80100300      APBUART3                  uart3_status
0x80100400      APBUART4                  uart4_status
0x80100500      APBUART5                  uart5_status
0x80000f00      AHBSTATUS0                ahbstatus0_status
0x80100800      GRSPW controller 0        grspw0_status
0x80100900      GRSPW controller 1        grspw1_status
0x80100a00      GRSPW controller 2        grspw2_status
0x80100b00      GRSPW controller 3        grspw3_status
0x80100c00      GRSPW controller 4        grspw4_status
0x80100d00      GRSPW controller 5        grspw5_status
0x80000400      SPI controller 0          spio0_status
0x80100e00      GPIO controller 0         gpio0_status
0x80100f00      GPIO controller 1         gpio1_status

Register         Register description               Value
--------------    ------------------------           ----
CCTRL           Cache control register            0x00020000
ICCFG           Icache config register            0x13230008
DCCFG           Dcache config register            0x1b220008
ASR16           LEONFT register file prot. reg    0x00000000
ASR17           Processor config register         0x00000507
```

tsim> uart0_status

Address         Register description               Value
-------------    ------------------------           -----
0x80000104      UART 0 status register            0x00000086
0x80000108      UART 0 control register           0x08000000
Command summary:

- `ahb len <length>` Set amba bus trace buffer length
- `ahb [length]` Show amba bus trace history
- `batch <file>` Execute a batch file of TSIM commands
- `load <file> [addr]` Load a binary file
- `bp [addr] [cpuX]...` Load a binary file
- `bp [<num>]` Enable idle-loop optimisation.
- `del <num>` Delete breakpoint <num>
- `bt [cpuX]...` Print backtrace
- `cont [cnt|time]` Continue execution for [cnt] instructions or [time] time
- `coverage <args>...` Coverage control, see manual for details
- `cp` List CPU info.
- `cpu <X>` Switch active CPU to be X
- `debug <level>` Set debug level
- `dbgon <flag>` Toggle debug <flag> for all cores, see manual for details
- `disassemble [addr][count]` Disassemble [count] instructions at address [addr]
- `dump [file][addr][count]` Dumps memory at [addr] to [file].
- `ep [address|clear]` Show, set or clear entry point
- `event` Show event queue
- `exit [val]` Exit the simulator with exit value [val] or 0.
- `flush [args]` Flush cache or caches, see manual for details
- `float [-v]` Print the FPU registers
- `gdb` Start gdb server listening for gdb connection
- `go [addr] [cnt/time]` Restart, reset and start execution
- `hist [trace_length]` Show combined inst/ahb trace history
- `icache` Show contents of instruction cache
- `inst len <length>` Set instruction trace buffer length
- `inst [length]` Show instruction trace history
- `leon` Display LEON peripherals registers
- `load <file>` Load a file into simulator memory
- `mcfgX [val]` Set or show user defined memory controller settings, X=1,2,3
- `mem [addr] [count]` Display memory at <addr> for [count] bytes
- `nolog <cmd>` Suppress log output of a command.
- `perf [reset]` Show/reset performance statistics
- `prof [0|1] [period]` Show/enable/disable profiling
- `quit` Exit the simulator
- `reg [reg] [val]` Show/set CPU registers (or windows, eg 'reg w2')
- `reset` Reset simulator
- `restore <file>` Temporarily disabled: Restore simulator state from file
- `save <file>` Temporarily disabled: Save simulator state to file
- `shell <cmd>` Execute shell command
- `step` Single step
- `symbols [file]` Load symbols from [file]
- `symbols list` Show symbols.
- `symbols lookup [symbol]` Lookup [symbol]
- `trace [cnt/time]` Trace instructions for [cnt] instructions or [time] time
- `thread [info|bt]` Print thread info or backtrace
- `version` Print the TSIM version and build date
- `vxmem <addr> <val>...` Write word(s) to virtual address <addr> (and onwards)
- `walk <addr>` Print a MMU table walk
- `watch <addr>` Add a watchpoint at <addr>
- `wmem <addr> [val]...` Write word(s) to address <addr> (and onwards)
- `wmeth <addr> [val]...` Write byte(s) to address <addr> (and onwards)
- `xmem <asi> <addr> <val>` Write word <val> to <addr> in address space <asi>

Type Ctrl-C to interrupt execution

See manual for details and additional command arguments.

For native TCL commands use "tcl_help"

IP cores and User modules:

- `mctrl_status` Print Memory configurations
- `irmgp_status` Print irmgp status
- `gptimer0_status` Print GPTIMER0 status
- `grtimer0_status` Print GRTIMER0 status
- `uart0_status` Print APBUART0 status
- `uart1_status` Print APBUART1 status
- `uart2_status` Print APBUART2 status
- `uart3_status` Print APBUART3 status
- `uart4_status` Print APBUART4 status
- `uart5_status` Print APBUART5 status
- `ahbstatus0_status` Print AHBSTATUS0 status
- `grspw0_dbg` Activate dbg output, *grspw0_dbg [<flags>][<clean>][<list>][<help>]* for info
grspw0_connect <ip>  connect GRSPW2 core to packet server at <ip>
grspw0_server <port> start packet server for GRSPW2 core on port <port>
grspw0_status [1|2|4] print GRSPW2 register status. flags:1=output verbose,2=dont compr...
grspw1_dbg Activate dbg output, *grspw1_dbg [flags]>[clean][list][help]* for info connect GRSPW2 core to packet server at <ip>
grspw1_server <port> start packet server for GRSPW2 core on port <port>
grspw1_status [1|2|4] print GRSPW2 register status. flags:1=output verbose,2=dont compr...
grspw2_dbg Activate dbg output, *grspw2_dbg [flags]>[clean][list][help]* for info connect GRSPW2 core to packet server at <ip>
grspw2_server <port> start packet server for GRSPW2 core on port <port>
grspw2_status [1|2|4] print GRSPW2 register status. flags:1=output verbose,2=dont compr...
grspw3_dbg Activate dbg output, *grspw3_dbg [flags]>[clean][list][help]* for info connect GRSPW2 core to packet server at <ip>
grspw3_server <port> start packet server for GRSPW2 core on port <port>
grspw3_status [1|2|4] print GRSPW2 register status. flags:1=output verbose,2=dont compr...
grspw4_dbg Activate dbg output, *grspw4_dbg [flags]>[clean][list][help]* for info connect GRSPW2 core to packet server at <ip>
grspw4_server <port> start packet server for GRSPW2 core on port <port>
grspw4_status [1|2|4] print GRSPW2 register status. flags:1=output verbose,2=dont compr...
grspw5_dbg Activate dbg output, *grspw5_dbg [flags]>[clean][list][help]* for info connect GRSPW2 core to packet server at <ip>
grspw5_server <port> start packet server for GRSPW2 core on port <port>
grspw5_status [1|2|4] print GRSPW2 register status. flags:1=output verbose,2=dont compr...
spi0_status print spi core 0 status information
spi0_dbg activate dbg output, *spi0_dbg [flags]>[list][help][clean]* for info
gpio0_status print gpio ctrl 0 status information
gpio0_dbg activate dbg output, *gpio0_dbg [flags]>[list][help][clean]* for info
gpio1_status print gpio ctrl 1 status information
gpio1_dbg activate dbg output, *gpio1_dbg [flags]>[list][help][clean]* for info
can_oc0_connect <ip> connect CAN_OC core 0 to packet server at <ip>
can_oc0_server <port> start packet server for CAN_OC core 0 on port <port>
can_oc0_status print CAN_OC core 0 status information
can_oc0_dbg activate dbg output, *can_oc0_dbg [flags]>[list][help][clean]* for info
can_oc1_connect <ip> connect CAN_OC core 1 to packet server at <ip>
can_oc1_server <port> start packet server for CAN_OC core 1 on port <port>
can_oc1_status print CAN_OC core 1 status information
can_oc1_dbg activate dbg output, *can_oc1_dbg [flags]>[list][help][clean]* for info
greth_connect <ip> connect to packet server at <ip>. If <ip> is not specified the default is localhost
greth_dump <file> Dump packets to Ethereal readable <file>. When <file> is not specified the current dumpfile will be closed
greth_ping <ip> Simulate a ping. Packets will be generated by Tsim. If <ip> is not specified the default <ip> is 192.168.0.80
g712_dbgon Activate dbg output, "g712_dbgon help" for info
6. Toolchains

6.1. Bare C Cross-Compiler System

6.1.1. 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 GR712RC Development Board. It will be demonstrated how to build an example program and run it on the GR712RC-BOARD using GRMON.

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.

6.1.2. Compiling with BCC

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

```bash
$ cat hello.c
#include <stdio.h>
int main(void)
{
    printf("hello, world\n");
    return 0;
}

$ sparc-gaisler-elf-gcc -qbsp=gr712rc -mcpu=leon3 -mfix-gr712rc -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>-qsrt</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

```bash
-qbsp=gr712rc -mcpu=leon3 -mfix-gr712rc
```

with GR712RC. For more details, see [RD-10].

6.1.3. Running and debugging with GRMON

Once your application is compiled, connect to your GR712RC-BOARD with GRMON. The following log shows how to load and run an application. Note that the console output is redirected to GRMON by the use of the `-u` command line switch, so that the application standard output is forwarded to the GRMON console.

```bash
$ grmon -f tdi -u
GRMON2 LEON debug monitor v2.0.42 professional version
```
To debug the compiled program you can insert breakpoints, step and continue execution directly from the GRMON console. Compilation symbols are loaded automatically by GRMON once you load the application. An example is provided below.

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

grmon3> cont
hello, world
CPU 0:  Program exited normally.
```

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

### 6.1.4. Running and debugging with TSIM

Once your application is compiled, start TSIM with the `-gr712rc -ahbm gr712.so` option. The following log shows how to load and run an application.

```plaintext
$ tsim-leon3 -gr712rc -ahbm gr712.so
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: 0x40000000, size 25824 bytes
section: .rodata, addr: 0x400064e0, size 128 bytes
section: .data, addr: 0x40006560, size 1184 bytes
read 350 symbols

tsim> run
starting at 0x40000000
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.

```plaintext
tsim> load hello.elf
section: .text, addr: 0x40000000, size 25824 bytes
section: .rodata, addr: 0x400064e0, size 128 bytes
section: .data, addr: 0x40006560, size 1184 bytes
read 350 symbols

// Set a breakpoint at main

tsim> bp main
breakpoint 1 at 0x4000124c: main + 0x4

// Start the program execution

tsim> run
starting at 0x40000000

// Step through the first instruction

tsim> step
3846 4000124c b0102000 mov 0, %i0                  [00000000]

tsim> step
3847 40001250 11100019 sethi %hi(0x40006400), %o0    [40006400]

// Continue execution

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].

## 6.2. RTEMS Real Time Operating System

### 6.2.1. Overview

RTEMS is a real time operating system maintained at [RD-6] that supports the LEON CPU family. Cobham Gaisler distributes a precompiled RTEMS toolchain for LEON called RCC [RD-7]. This section gives the reader a brief introduction on how to use RTEMS together with the GR712RC Development Board. It will be demonstrated how to install RCC and build an existing sample RTEMS project from RCC and run it on the board using GRMON.

The RCC toolchain includes a prebuilt toolchain with GNU BINUTILS, GCC, NewlibC and GDB for Linux and Windows (mingw). It also contains prebuilt RTEMS kernels for the LEON2, LEON3/4 BSPs single-core and for multi-core development, see [RD-8] for more information. The LEON BSP specific drivers are documented in [RD-9].

Sample RTEMS projects are available within the toolchain package, installed into `rtems-x.y/src/samples`.

### 6.2.2. Installing RCC

The RCC toolchain is downloadable from the RCC homepage at [RD-7]. The full installation procedure is found in the RCC manual [RD-8]. Windows users are recommended to install the UNIX-like environment MSYS before proceeding.

The installation process of RCC is straight forward by first extracting the toolchain into `C:\opt` or `/opt` on Linux, then extracting the source distribution into the `/opt/rtems-x.y/src/` directory. In order for the compiler to be found one has to add the binary directory `/opt/rtems-x.y/bin` into the PATH variable as below:

```bash
$ cd /opt
$ tar -xf sparc-rtems-4.10-...-linux.tar.bz2
$ cd rtems-4.10/src
$ tar -xf rtems-4.10-...-src.tar.bz2
$ export PATH=$PATH:/opt/rtems-4.10/bin
```

### 6.2.3. Building an RTEMS sample application

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

```
sparc-rtems-gcc -g -O2 rtems-hello.c -o rtems-hello
```
RCC's gcc creates executables for LEON3/4 by default. The default load address is at the start of the RAM, i.e. 0x40000000. All compilation options are described in [RD-8], but some useful options are reported below:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-g</code></td>
<td>generate debugging information - must be used for debugging with gdb</td>
</tr>
<tr>
<td><code>-msoft-float</code></td>
<td>emulate floating-point - must be used if no FPU exists in the system</td>
</tr>
<tr>
<td><code>-mcpu=v8</code></td>
<td>generate SPARC V8 mul/div instructions - needs hardware multiply and divide</td>
</tr>
<tr>
<td><code>-O2 or -O3</code></td>
<td>optimize code maximum performance and minimal code size</td>
</tr>
<tr>
<td><code>-qleon3std</code></td>
<td>generate LEON3/4 executable without driver manager startup initialization</td>
</tr>
<tr>
<td><code>-qleon3mp</code></td>
<td>generate LEON3/4 Multiprocessor executable (AMP)</td>
</tr>
</tbody>
</table>

### 6.2.4. Running and debugging with GRMON

Once your executable is compiled, connect to your GR712RC-BOARD with GRMON. The following log shows how to load and run an executable. Note that the console output is redirected to GRMON by the use of the `-u` command line switch, so that printf output is shown directly in the GRMON console.

```
[andrea@localhost samples]$ grmon -ftdi -u
GRMON2 LEON debug monitor v2.0.42 internal version
Copyright (C) 2013 Aeroflex Gaisler - All rights reserved.
For latest updates, go to http://www.gaisler.com/
Comments or bug-reports to support@gaisler.com
Parsing -ftdi
Parsing -u
[...]
```

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

```
grmon2> load rtems-hello
40000000 .text                    136.4kB / 136.4kB   [===============>] 100%
400221A0 .data                      4.4kB /   4.4kB   [===============>] 100%
40023350 .jcr                         4B              [===============>] 100%
Total size: 140.83kB (780.05kbit/s)
Entry point 0x40000000
Image /home/andrea/Desktop/samples/rtems-hello loaded
grmon2> run
Hello World
CPU 0:  Program exited normally.
CPU 1:  Power down mode
```

grmon2> bp Init
Software breakpoint 1 at <Init>
```
grmon2> run
CPU 0:  breakpoint 1 hit
0x400011f8: 1110007f  sethi  %hi(0x4001FC00), %o0  <Init+4>
CPU 1:  Power down mode
```
```
grmon2> step
0x400011fc: 4000003b  call  0x400012E8  <Init+8>
```
```
grmon2> cont
```
Hello World

CPU 0: Program exited normally.
CPU 1: Power down mode

grmon2> Exiting GRMON

Alternatively you can run GRMON with the -gdb command line option and then attach a gdb session to it. For further information see Chapter 3 of [RD-8].

6.3. VxWorks

6.3.1. Overview

VxWorks is an embedded real-time operating system developed by WindRiver. Cobham Gaisler provides a LEON architectural port (HAL) and a Board Support Package (BSP) in full source code for VxWorks 7, described at [RD-12].

The VxWorks package includes a quick start guide and technical support. Contact support@gaisler.com for more information.

6.4. MKPROM2

6.4.1. Overview

To run application from the on-board PROM, it’s necessary to create a bootable PROM image file. MKPROM2 is a utility program to create boot-images for programs compiled with the BCC or RTEMS cross-compiler. It encapsulates the application in a loader suitable to be placed in a boot PROM. The application is compressed with a modified LZSS algorithm, typically achieving a compression factor of 2.

The boot loader operates in the following steps:

- The register files of IU and FPU (if present) are initialized.
- The memory controller, UARTs and timer unit are initialized according to the specified options.
- The application is decompressed and copied into RAM.
- Finally, the application is started, setting the stack pointer to the top of RAM.

6.4.2. Usage of MKPROM2

The MKPROM2 tool can be downloaded from the Cobham Gaisler website [http://gaisler.com/index.php/downloads/compilers]. No installation is required, but the directory containing the executable must be included in the system's PATH, together with a valid SPARC toolchain (sparc-gaisler-elf, sparc-elf, sparc-rtems or sparc-linux).

To generate a boot PROM for a GR712RC Development Board and running your program from SRAM:

```
mkprom2 -leon3 -freq 80 -rmw -ramsize 8192 -romsize 8192 -baud 38400 -ramws 2 -o hello.prom hello.exe
```

This example command will work for a board in the default configuration at delivery, with a system clock frequency of 80 MHz. The -ramsize and -romsize options are expressed in KiB. The former value is 8MiB, the size of the SRAM. The latter value is 8 MiB as well, and represents the size of the on-board flash. Finally the -ramws option sets the number of wait states during SRAM access to 2, needed when running the system at 80 MHz, but might be a lower value at lower frequencies.

To generate a boot PROM for a GR712RC Development Board and running your program from SDRAM:

```
mkprom2 -leon3 -freq 80 -rmw -nosram -sdram 128 -romsize 8192 -baud 38400 -o hello.prom hello.exe
```

This example command will, again, work for a board in the default configuration at delivery. The -nosram option will disable the SRAM and the -sdram option sets the size of the available SDRAM. This value is 128 MiB, which is the value for the SODIMM provided by default with the board.

When SRAM is disabled, the SDRAM address range is moved from 0x60000000 to 0x40000000, therefore not requiring recompilation of executables. To run your program in SDRAM without disabling SRAM, you need to
link your program to the 0x60000000 address at compilation time. See the manual of your toolchain for more information.

It's required that the MKPROM2 command line parameters match your system configuration. For more information about command line options, please refer to [RD-13].

If EDAC is enabled on the board's PROM, then the `-bch8` flag must be included in the command line. The generated PROM image that needs to be flashed on the device in this case would be `hello.prom.bch8`.

Once the PROM file is generated, it can be loaded onto the board with GRMON. Once GRMON is attached to the board, run the following commands to program the PROM.

```
flash
flash erase all
flash load hello.prom
verify hello.prom
```

For further information about connecting to the board with GRMON, see Chapter 4.
7. Frequently Asked Questions / Common Mistakes / Know Issues

This section contains application information applicable to the GR712RC-BOARD and custom systems with the GR712RC component.

7.1. GR712RC

7.1.1. Clock gating

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

Alternatively, if a boot loader is used instead of GRMON to load an executable, then clock gating must be setup via the General Purpose register. Clock source/divider selection must also be setup for the MIL-1553, SpaceWire and TM cores. See Chapter 13 of [RD-2].

7.1.2. GRMON issues

When connected to the board, the message "stack pointer not set" will be shown by the command `info sys` in case GRMON doesn't find any memory.

7.1.3. GPIO controller does not remember interrupt requests

The GR712RC GPIO controller allows controlling interrupt mask/edge/polarity and generate interrupt requests to the interrupt controller. It does not however store the history of interrupts it has generated, so if the interrupt request number is shared with other interrupts, the interrupt handler must have some external way to determine if the interrupt was actually generated by the GPIO or some of the other (shared) interrupts.

7.1.4. Multiprocessor & legacy support

Code compiled for the single core LEON3 will generally be able to run unmodified on the GR712RC. The second core is inactivated after reset and unless it's activated (by writing a specific bit in the IRQ controller) it will remain inactivated and the chip will behave as a single-CPU system.

7.1.5. Inter-processor interrupts

When using a multiprocessor OS like RTEMS-AMP, Linux or VxWorks the default IRQ for interprocessor cross-calls, IRQ 14, clashes with the MIL-1553, Ethernet and Telecommand IP cores. The OS may need to be reconfigured by changing the IRQ value, which is usually a define, in the source code of your operating system and rebuilding it. This should not be an issue with single-core RTEMS.

7.1.6. Interrupt considerations

7.1.6.1. IRQMP `ilevel` functionality

IRQMP "ilevel (0 or 1)" functionality is not used by the operating systems supported by Cobham Gaisler. It is just set to all 0.

The reason is that `ilevel` is of limited use. In particular, it does not really add anything to applications using nested interrupts. Whatever the value of `ilevel`, it can not change the interrupt request level provided to the CPU (CPUx.IRL[3:0]).

So for example when interrupt 13 is being processed with nesting, then the interrupt trap handler sets PSR.PIL=13 to allow interrupts 14 and higher. However, the interrupt 2 can never interrupt this until PSR.PIL is lowered again by the trap handler.

One scenario where the `ilevel` functionality is useful is when nesting is not used. When nesting is not used, then all interrupt trap handlers set PSR.PIL to 15. In this case, `ilevel` can be used to control which interrupt to take when a previous interrupt trap handler exits (PSR.PIL restored to 0).
7.1.6.2. GR712RC interrupt assignments are static

There is no way to remap the interrupt request level for peripherals in the GR712RC. Later LEON components have support for true interrupt remapping.

7.1.6.3. Extended interrupts always clears interrupt pending bit 12

When the CPU acknowledges interrupt 12 (including "extended") on GR712RC, the interrupt controller will always clear bit 12 in the IRQMP pending register, independent of what the cause was. This means that GPIO 12 interrupt can get lost if extended interrupts (16..31) is used in the application.

The recommendation is thus to not use GPIO 12 interrupt if the application also uses any of interrupt 16..31. If GPIO 12 interrupt must be used, then it needs to be checked each time any of extended interrupt occurs. That is, at the end of ISR for 16..31, call also the GPIO 12 ISR.

7.1.6.4. Downgrading a high prio interrupt

A high prio interrupt can be "downgraded" to a low prio interrupt with some software support: Install an ISR (or direct trap handler) for the high prio interrupt which just forces interrupt 1 and then returns. When the high prio interrupt returns, the interrupt 1 will be taken. Interrupt 1..15 can be forced atomically with the IRQMP force register.

7.1.7. GRMON Debug Link Limitations

The GR712RC does not support debugging over Ethernet. EDCL is not included in the Ethernet core design. Refer to Chapter 4 for an introduction to the supported debug links.

7.1.8. MIL-1553

The 1553 IP core in the GR712RC is an Actel Core1553BRM with an AMBA adapter developed by Cobham Gaisler. Actel's core is documented on Actel's website [http://www.actel.com/ipdocs/Core1553BRM_HB.pdf], while the wrapper is documented in [RD-2].

The correct RTEMS driver to use for the MIL-1553 core is B1553BRM. This should not be confused with GR1553B which is the driver for Cobham Gaisler's in-house developed core. To use the core, users need to set up clock gating and clock selection with the general purpose register.

There are some restrictions on what clock frequencies can be used, see Section 3.3 of [RD-2].

Users also need to set a register inside the Core1553BRM to match the BRM frequency used. This is usually done by the driver in the RTEMS/VxWorks case (default is 24 MHz). Below is provided an example routine for setting up GR712RC clocking to external 24 MHz clock. This routine can be used, for instance, as mkprom2 bdinit. In this case it needs to be compiled with -O2 to avoid using stack.

```c
static void gr712_init(void) {
    volatile unsigned long *p;
    /* Select external 1553 clk through GPRG */
    p = (volatile unsigned long *)0x80000600;
    *p |= 0x20;
    /* Ungate 1553 clock and reset */
    p = (volatile unsigned long *)0x80000D00;
    p[0] = (1<<11);
    p[1] = (1<<11);
    p[2] = (1<<11);
    p[2] = 0;
    p[0] = 0;
    /* Set Core1553BRM to 24 MHz operation */
    p = (volatile unsigned long *)0xFFF00000;
    p[32] |= 3;
}
```

7.1.9. CAN multiplexing

The CAN bus outputs are disabled at reset and should be enabled before use by programming the CAN multiplexer. To enable OC-CAN1 on CAN bus A and OC-CAN2 on CAN bus B, the following GRMON command can be used:
There is also an RTEMS driver named `canmux` for the CAN bus multiplexor distributed with the RCC distribution. The multiplexer is programmed by opening the file "/dev/canmux" and requesting an IOCTL. An example of this is provided in the RCC example src/samples/gr712/rtems-satcan.c.

### 7.1.10. Concurrent CAN and Ethernet

Ethernet and CAN pins are conflicting in the GR712RC switch matrix so the functions can not be used concurrently. It is possible to switch between the interfaces at run-time.

The conflict comes from a set of pins which are activated when the CAN interface is enabled. These pins are described as "proprietary" in the rightmost column of [RD-2], table named I/O switch matrix pin description. The "proprietary" interface is obsolete and not part of the public interface of the GR712RC but are still part of the I/O switch matrix.

When the CAN interface is enabled, the pins 191, 190, 185, 184 and 172 are also driven with "random values" unless these pins are assigned to an interface with higher I/O switch matrix priority, such as when the MIL-STD-1553B interface is enabled. Of these pins, 191, 190 and 185 are shared with Ethernet (RMII) and there is the conflict.

Ethernet has lower switch matrix priority than CAN. So when CAN is enabled, the pins 191, 190 and 185 are not driven by the Ethernet controller.

GPIO bits 13...16 are unavailable when CAN or SPW-2 is enabled. GPIO pins always has lowest priority in the switch matrix.

How a particular device is "enabled" with regard to the switch matrix is described in [RD-2], Table 9. Most devices are I/O-enabled by the clock gating unit and some are enabled via device control registers.

### 7.1.11. Hardware behavior at CPU reset and power management

GR712RC has the following behavior with regard to how the power-down mode and program counter (PC) is managed:

- At GR712RC power-on, and at system reset issued by asserting RESETN (including watchdog), the following is done:
  - CPU0 sets PC=0 and starts execution.
  - CPU1 sets PC=0 and is powered down.
  - A CPU enters power-down mode by writing to its own CPU local register %ASR19. The local PC is set to the instruction following the %ASR19 write.
  - A CPU can not power down another CPU.
  - Any CPU can read the power-down status of any other CPU by reading the Multiprocessor status register in the interrupt controller.
  - Any CPU can resume execution on any other CPU by writing to the Multiprocessor status register in the interrupt controller.
  - When a CPU is being resumed (powered up) by a write to the Multiprocessor status register, it continues executing at its current PC.
  - If an (unmasked) interrupt occurs on a powered down CPU, then the target CPU resumes execution on the current PC.

The above behavior has some implications with regard to how processors have to cooperate on software initiated restart.

At power-on, the RESETN signal is engaged and thus the PC for CPU1 has a good known value. No specific preparations on CPU1 has to be performed for power-on.

For a software-initiated restart of the system, all processors need to synchronize before issuing the restart. All processors except for CPU0 should typically be powered down and the other processors should resume execution in memory which is guaranteed to exist as soon as they are resumed. This is because the currently executed appli-
cation and its data (in RAM) will typically disappear when CPU0 performs low-level initialization of memory controller and clearing RAM.

To solve this for the secondary processors, a dedicated “parking routine” in ROM could be entered by secondary processors before CPU0 performs the software-initiated restart.

7.2. GR712RC-BOARD

7.2.1. Clock problems

Ensure that the jumper JP84, selecting the clock source, is always present. A combination of its absence and the presence of jumper JP88, can lead to unexpected processor behavior.

When jumper JP88 is present, the oscillator in socket X5, which is provided by default, must be disconnected or it will short with the main clock source, leading to possible damage to the oscillators and unexpected behavior.

7.2.2. Switch Matrix Configuration Problems

Ensure that the jumper array is properly configured and that any I/O peripheral required is clock ungated or enabled. The internal switch matrix routing is explained more in depth in Chapter 2 of GR712RC User Manual.

If an IP core behaves correctly, as seen from software, but does not receive/transmit any data from the outside, first check that the jumper array is properly configured. The problem might also arise when conflicting cores are enabled. Check Table 8 from [RD-2] for further information on conflicting cores.

7.2.3. GPIO used as configuration at reset

Some of the GPIOs have special meaning on power-up, GPIO[1] and GPIO[3] configure the PROM area of the memory controller and GPIO[42], GPIO[40], GPIO[37] and GPIO[34] are used for the SPW clock divider reset value.

These pins are provided with pull-down resistors by default. If measuring the state of these GPIO pins, please take into account the effect of these pull-down resistors. Conversely, if an external signal is connected to the GPIO[3] and GPIO[1] pins, this may override the expected state of the pin at power up.

See Section 2.3.2 and Section 2.6.2 of [RD-1] for more information.

7.2.4. SDRAM configuration

SDRAM is, by default, not configured on the board. Ensure that the switch matrix jumper configuration is correctly set as to enable SDRAM. If in doubt, you can use a default configuration that supports SDRAM. See Section 2.3 for more details.

Only half of the installed SDRAM will be available in the system, as reported by GRMON's info sys command. This limitation is due to the fact that the SODIMM provides 64 bit data paths, but in the standard LEON model only 32 bits of the SDRAM are used, plus 16 additional data bits for the RS/EDAC memory bits.
8. Support

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

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

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