LEON-XCKU-EX

LEON5 Xilinx Ultrascale example design

2020 User's Manual

The most important thing we build is trust

LEON-XCKU-EX Quick Start Guide

Table of Contents

1. Introduction 3
  1.1. Overview 3
  1.2. Availability 3
  1.3. Prerequisites 3
  1.4. References 3
2. Overview 4
  2.1. Board 4
  2.2. The design 4
  2.3. Debug tools 4
3. Board Configuration 5
  3.1. Buttons and switches 5
  3.2. Connectors 5
  3.3. LEDs 5
  3.4. Memories 5
    3.4.1. 5
  3.5. Programming the bitstream 5
4. Software Development Environment 7
  4.1. Overview 7
5. GRMON hardware debugger 8
  5.1. Overview 8
  5.2. Debug-link alternatives 8
    5.2.1. Connecting via the USB JTAG connector 8
    5.2.2. Connecting via the Ethernet debug interfaces 8
    5.2.3. Connecting via the UART debug link 8
  5.3. First steps 8
  5.4. Connecting to the board 9
  5.5. Get system information 9
  5.6. Load a RAM application 9
6. RTEMS Real Time Operating System 11
  6.1. Overview 11
  6.2. Installing RCC 11
  6.3. Building an RTEMS sample application 11
  6.4. Running and debugging with GRMON 11
7. Support 13

1. Introduction

1.1. Overview

This document is a quick start guide for the LEON-XCKU-EX example designs.

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. Availability

The FPGA bitstreams are available on the LEON-XCKU-EX web page: https://www.gaisler.com/LEON-XCKU.

Sample linux images to load and run are available at https://www.gaisler.com/anonftp/linux/linux-2.6/images/.

1.3. Prerequisites

To use the provided bitstream, the user needs:

- Xilinx KCU105 board
- GRMON3, latest available version, available at https://www.gaisler.com/index.php/downloads/debug-tools.
- Xilinx Vivado Design Suite (to program the FPGA). Vivado is available at https://www.xilinx.com/products/design-tools/vivado.html.

1.4. References

Table 1.1. References

RD-1	The SPARC Architecture Manual, Version 8, Revision SAV080SI9308
RD-2	GRMON User's Manual [https://www.gaisler.com/doc/grmon3.pdf]
RD-3	RTEMS homepage [https://www.rtems.org]
RD-4	RTEMS User Manual [https://docs.rtems.org/branches/master/user/index.html]
RD-5	LEON/ERC32 RTEMS Cross Compilation System (RCC) [https://www.gaisler.com/index.php/products/operating-systems/rtems]
RD-6	RCC User's manual [https://gaisler.com/anonftp/rcc/doc]
RD-7	Cobham Gaisler RTEMS driver documentation [https://gaisler.com/anonftp/rcc/doc]
RD-8	Bare C Cross-Compilation System [https://www.gaisler.com/index.php/products/operating-systems/bcc]
RD-9	BCC User's Manual [https://www.gaisler.com/doc/bcc2.pdf]
RD-10	VxWorks 7 SPARC architectural port and BSP [https://www.gaisler.com/index.php/products/operating-systems/vxworks-7]
RD-11	LEON-XCKU-EX User Manual [https://www.gaisler.com/products/leon5/20200630/LEON-XCKU-EX-UM.pdf]
RD-12	KCU105 Board User Guide [https://www.xilinx.com/support/documentation/boards_and_kits/kcu105/ug917-kcu105-eval-bd.pdf]

2. Overview

2.1. Board

The LEON-XCKU-EX example designs can be used on the following board:

- Xilinx KCU105

2.2. The design

The SoC system is described in the LEON-XCKU-EX User's manual LEON-XCKU-EX-UM, available at https://www.gaisler.com/LEON-XCKU. For details about the the interfaces' connections in the board see (Chapter 3).

2.3. Debug tools

Non-intrusive debugging of the template design and application execution can be performed using the GRMON hardware debugger.

3. Board Configuration

This chapter describes boards items as used by the LEON-XCKU-EX design.

3.1. Buttons and switches

- `CPU_RST` button: Main reset to the FPGA design
- `SW7`, `SW8`, `SW9`, `SW10` buttons: GPIO0 inputs 4, 5, 6, 7
- `SW12[1:4]`: 4-Pole DIP Switch GPIO0 connected to inputs 0-1-2-3.

The Switch `SW12[1]` also acts as select signal for the UART interface.

- When '1' the UART interface is connected to the UART debug link (`AHBUART`).
- When '0' the UART interface is connected to the `APBUART`.

3.2. Connectors

- `J87`: USB JTAG interface via Digilent module with micro-B USB connector. Connector USB-JTAG. See (Chapter 5).
- `J4`:U SB UART interface. Connector USB-UART. AHBUART debug link or APBUART function selectable by `SW12[4]`.
- Ethernet PHY SGMII interface with RJ-45 connector. See (Chapter 5).
- `J52`: PMOD Connector J52: GPIO I/O 8-15.

3.3. LEDs

- `LED[0..3]`: Connected to GPIO outputs 16-17-18-19.
- `LED[4]`: Connected to DSU_SEL signal
- `LED[5]`: When ON indicates that the CPU is in error mode.
- `LED[6..7]`: When ON they indicate that the memory controller calibration is complete and the FPGA desing has access to the on-board SDRAM.

3.4. Memories

The board is equipped with 2 GB DDR4 component memory (four [256 Mb x 16] devices).

3.5. Programming the bitstream

A vivado script to program the FPGA is provided in the bitstream folder. It has been tested using Vivado 2018.1 and 2019.2. The bitstream folder contains several bistreams which represent different configurations of the processor (EX1,EX2,EX3 and EX4). Select one of the bitstreams (described in [RD-11]. and follow the instructions below to program the FPGA:

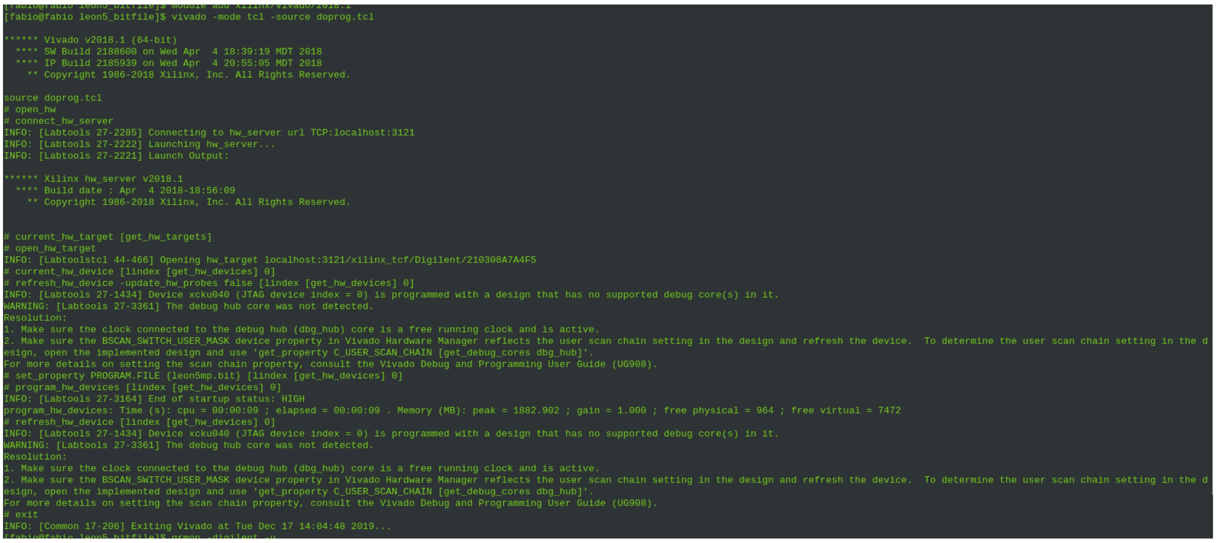
1. Connect the PC and the board using a standard micro-USB cable into the connector `USB-JTAG J87`.
2. Make sure that Vivado is added to your path variables
3. Open a terminal in the downloaded folder and issue the following command to launch Vivado:

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

4. To program the FPGA with the selected configuration, run in the Vivado console (in this case for EX4):

   ```
   doprog EX4
   ```

   The expected output should be as in the following image:



```
[fabio@fabio leon5_bitfile]$ vivado -mode tcl -source doprog.tcl

****** Vivado v2018.1 (64-bit)
  **** SW Build 2188600 on Wed Apr  4 18:39:19 MDT 2018
  **** IP Build 2185939 on Wed Apr  4 20:55:05 MDT 2018
    ** Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.

source doprog.tcl
# open_hw
# connect_hw_server
INFO: [Labtools 27-2285] Connecting to hw_server url TCP:localhost:3121
INFO: [Labtools 27-2222] Launching hw_server...
INFO: [Labtools 27-2221] Launch Output:

****** Xilinx hw_server v2018.1
  **** Build date : Apr  4 2018-18:56:09
    ** Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.


# current_hw_target [get_hw_targets]
# open_hw_target
INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/210308A7A4F5
# current_hw_device [lindex [get_hw_devices] 0]
# refresh_hw_device -update_hw_probes false [lindex [get_hw_devices] 0]
INFO: [Labtools 27-1434] Device xcku040 (JTAG device index = 0) is programmed with a design that has no supported debug core(s) in it.
WARNING: [Labtools 27-3361] The debug hub core was not detected.
Resolution:
1. Make sure the clock connected to the debug hub (dbg_hub) core is a free running clock and is active.
2. Make sure the BSCAN_SWITCH_USER_MASK device property in Vivado Hardware Manager reflects the user scan chain setting in the design and refresh the device.  To determine the user scan chain setting in the d
esign, open the implemented design and use 'get_property C_USER_SCAN_CHAIN [get_debug_cores dbg_hub]'.
For more details on setting the scan chain property, consult the Vivado Debug and Programming User Guide (UG908).
# set_property PROGRAM.FILE {leon5mp.bit} [lindex [get_hw_devices] 0]
# program_hw_devices [lindex [get_hw_devices] 0]
INFO: [Labtools 27-3164] End of startup status: HIGH
program_hw_devices: Time (s): cpu = 00:00:09 ; elapsed = 00:00:09 . Memory (MB): peak = 1882.902 ; gain = 1.000 ; free physical = 964 ; free virtual = 7472
# refresh_hw_device [lindex [get_hw_devices] 0]
INFO: [Labtools 27-1434] Device xcku040 (JTAG device index = 0) is programmed with a design that has no supported debug core(s) in it.
WARNING: [Labtools 27-3361] The debug hub core was not detected.
Resolution:
1. Make sure the clock connected to the debug hub (dbg_hub) core is a free running clock and is active.
2. Make sure the BSCAN_SWITCH_USER_MASK device property in Vivado Hardware Manager reflects the user scan chain setting in the design and refresh the device.  To determine the user scan chain setting in the d
esign, open the implemented design and use 'get_property C_USER_SCAN_CHAIN [get_debug_cores dbg_hub]'.
For more details on setting the scan chain property, consult the Vivado Debug and Programming User Guide (UG908).
# exit
INFO: [Common 17-206] Exiting Vivado at Tue Dec 17 14:04:48 2019...
```

Figure 3.1.

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

   ```
   grmon -digilent
   ```

   Please see (Chapter 5) for further information regarding GRMON and the available debug links.

Alternatively, the bitfile (`.bit`) can be programmed to the Xilinx KCU105 using the Vivado design suite. Start Vivado and select the menu item *Flow -> Open Hardware Manager*. Once the FPGA has been programmed, remember to close the hardware manage before connecting with GRMON.

4. Software Development Environment

4.1. Overview

Cobham Gaisler provides a comprehensive set of software tools to run several different operating systems. The LEON5 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 servises provided by an operating system
RTEMS	a hard Real Time Operating System. Cobham Gaisler provides, for LEON5, a preliminary toolchain and kernel to develop and compile RTEMS applications.
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 debug tools. The LEON5 platform is supported by the following:

GRMON	Used to run and debug applications on LEON-XCKU-EX hardware. See (Chapter 5).

The recommended method to load software onto LEON-XCKU-EX is by connecting to a debug interface of the board through the GRMON hardware debugger (Chapter 5).

5. GRMON hardware debugger

5.1. Overview

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

- Inspect LEON5 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 LEON-XCKU-EX example designs. After that, a basic first inspection of the board is exemplified.

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

5.2. Debug-link alternatives

5.2.1. Connecting via the USB JTAG connector

Connect the PC and the board using a standard micro-USB cable into the connector USB-JTAG J87 and issue the following command:

```
grmon -digilent
```

If the the debug link is not established, please see the section "Digilent HS1" of [RD-2].

5.2.2. Connecting via the Ethernet debug interfaces

If another address is wanted for the Ethernet debug link then one of the other debug links must be used to connect GRMON to the board. The EDCL IP address can then be changed using GRMON's **edcl** command. This new address will persist until next system reset.

With the Ethernet Debug Communication Link 0 address set to 192.168.0.51 the GRMON command to connect to the board is:

```
grmon -eth 192.168.0.51
```

5.2.3. Connecting via the UART debug link

Make sure that the switch `SW12[1]` select the UART debug link (ON position). Connect the PC and the board using a standard micro-USB cable into the connector USB-UART J4 and issue the following command:

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

5.3. 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 LEON-XCKU-EX board have been set up so that GRMON can connect to the board.

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

- AMBA system frequency is printed out at connect, if the frequency is wrong then it might be due to noise in auto detection (small error). See `-freq` flag in [RD-2].
- Memory location and size configuration is found from the **info sys** output.

5.4. Connecting to the board

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

```
grmon -digilent

GRMON LEON debug monitor v3.1.2.1-16-g061c5fb 64-bit 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

This internal version will expire on 06/12/2020

Parsing -digilent

Commands missing help:

JTAG chain (1): xcku040
Device ID:          0x288
GRLIB build version: 4245
Detected frequency:  100.0 MHz

Component                            Vendor
LEON5 SPARC V8 Processor             Cobham Gaisler
GR Ethernet MAC                      Cobham Gaisler
LEON5 Debug Support Unit             Cobham Gaisler
L2-Cache Controller                  Cobham Gaisler
AHB/APB Bridge                       Cobham Gaisler
Xilinx MIG DDR3 Controller           Cobham Gaisler
AHB Debug UART                       Cobham Gaisler
XILINX SGMII Interface               Cobham Gaisler
General Purpose I/O port             Cobham Gaisler
General Purpose I/O port             Cobham Gaisler
Generic UART                         Cobham Gaisler
Multi-processor Interrupt Ctrl.      Cobham Gaisler
Modular Timer Unit                   Cobham Gaisler

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

5.5. Get system information

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

```
grmon3> info reg uart0
  Generic UART
      0x80000104  UART Status register                    0x00000086
      0x80000108  UART Control register                   0x80000003
      0x8000010c  UART Scaler register                    0x00000145
grmon3> info sys gptimer0
  gptimer0  Cobham Gaisler  Modular Timer Unit
            APB: 80000300 - 80000400
            IRQ: 8
            16-bit scalar, 2 * 32-bit timers, divisor 100
```

5.6. Load a RAM application

An application linked to RAM can be loaded directly with the **load** and run with **run**.

```
grmon3> load hello.elf
  40000000 .text                 142.0kB / 142.0kB   [===============>] 100%
  400237D0 .rtemsroset              96B              [===============>] 100%
  40024840 .data                   4.4kB /   4.4kB   [===============>] 100%
  Total size: 146.44kB (777.96kbit/s)
  Entry point 0x40000000
  Image hello.elf loaded
```

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

grmon3> run
hello, world

  CPU 0:  Program exited normally.
```

The two lines starting with `Hello World` is the program output which is forwarded to the GRMON terminal.

6. RTEMS Real Time Operating System

6.1. Overview

RTEMS is a real time operating system maintained at [RD-3] 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 LEON-XCKU-EX example designs. 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, LEON3/4/5 BSPs single-core and for multi-core development, see [RD-6] for more information. The LEON BSP specific drivers are documented in [RD-7].

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

6.2. 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-6]. 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:

```
$ 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
```

6.3. 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 LEON3/4/5 by default. The default load address is at the start of the RAM, i.e. 0x40000000. All compilation options are described in [RD-6], but some useful options are reported below:

Table 6.1. RCC's GCC compiler relevant options

`-g`	generate debugging information - must be used for debugging with gdb
`-msoft-float`	emulate floating-point - must be used if no FPU exists in the system
`-mcpu=v8`	generate SPARC V8 mul/div instructions - needs hardware multiply and divide
`-O2` or `-O3`	optimize code maximum performance and minimal code size

6.4. Running and debugging with GRMON

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

[...]

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
```

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 (781.11kbit/s)
  Entry point 0x40000000
  Image /home/andrea/Desktop/samples/rtems-hello loaded

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
  0x400011f8: 1110007f  sethi  %hi(0x4001FC00), %o0  <Init+4>

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-6].

7. 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.

Cobham Gaisler AB
Kungsgatan 12
411 19 Gothenburg
Sweden
www.cobham.com/gaisler
sales@gaisler.com
T: +46 31 7758650
F: +46 31 421407