MCXL Reference Designs User Guide

Created on: Mar 22, 2022

Created by: Johannes Schwenk

CHAPTER

ONE

ABOUT THIS MANUAL

1.1 Imprint

Address:

ARIES Embedded GmbH

Schöngeisinger Str. 84

D-82256 Fürstenfedbruck

Germany

Phone:

+49 (0) 8141/36 367-0

Fax:

+49 (0) 8141/36 367-67

1.2 Disclaimer

ARIES Embedded does not guarantee that the information in this document is up-to-date, correct, complete or of good quality. Liability claims against ARIES Embedded, referring to material or non-material related damages caused, due to usage or non-usage of the information given in this document, or due to usage of erroneous or incomplete information, are exempted, as long as there is no proven intentional or negligent fault of ARIES Embedded. ARIES Embedded explicitly reserves the rights to change or add to the contents of this Preliminary User Guide or parts of it without notification.

1.3 Copyright

1.4 Registered Trademarks

1.5 Care and Maintenance

- Keep the device dry. Precipitation, humidity, and all types of liquids or moisture can contain minerals that will corrode electronic circuits. If your device does get wet, allow it to dry completely.
- Do not use or store the device in dusty, dirty areas. Its moving parts and electronic components can be damaged.
- Do not store the device in hot areas. High temperatures can shorten the life of electronic devices, damage batteries, and warp or melt certain plastics.
- Do not store the device in cold areas. When the device returns to its normal temperature, moisture can form inside the device and damage electronic circuit boards.
- Do not attempt to open the device.
- Do not drop, knock, or shake the device. Rough handling can break internal circuit boards and fine mechanics.
- Do not use harsh chemicals, cleaning solvents, or strong detergents to clean the device.
- Do not paint the device. Paint can clog the moving parts and prevent proper operation.
- Unauthorized modifications or attachments could damage the device and may violate regulations governing radio devices.

1.6 Change Log

Revision	Date	Revised	Comment
1.0	22.03.2022	js	Initial creation

CHAPTER

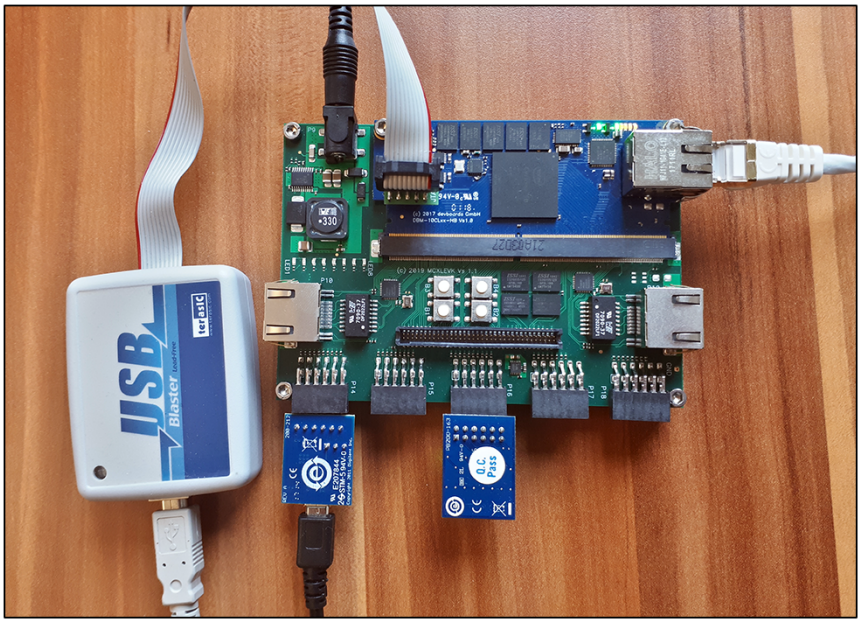
TWO

INTRODUCTION

The MCXL is an FPGA System-on-Module featuring the Intel Cyclone 10 LP FPGA. The two variants MCXL-S and MCXL-H implement SDRAM or HyperBus technology (HyperRAM and HyperFlash) as memory for the FPGA respectively.

The reference designs demonstrate the implementation of an open-source RISC-V core running FreeRTOS with TCP. Access to the HyperBus interface is provided via the SLL MBMC (HyperBus) IP Core. The examples are also suitable as starting point for development.

This guide shows how to install the necessary requirements and how to run the RISC-V examples on the MCXL.



CHAPTER

THREE

REQUIREMENTS

Note: As reference for this guide Linux Ubuntu 20.04 is used, other Linux distributions may work similarly. While the toolchain can be used on Windows only limited support is available.

To evaluate the demos the following hardware items are required:

- One of the following SoMs:
  - MCXL-H055BBB-I (10CL055YU484I7G)
  - MCXL-S055BC-I (10CL055YU484I7G)
- USB-Blaster with 0.05" adapter

The following items are intended to be used by the design but are optional:

- PMod USBUART (if serial console is desired)
- PMod 8LD (as GPIO output)

The following software is required, the installation is outlined in this chapter:

- RISC-V GCC - to compile the firmware
- Intel Quartus Prime (Lite) - to build the hardware design
- Quartus Programmer - to program the FPGA (usually included in Quartus Prime)

3.1 RISC-V GCC Toolchain

This guide installs the toolchain under `/opt/riscv`, this path is configurable. For other Linux distributions the toolchain can be installed similarly. For more information, please visit the official RISC-V GNU Compiler Toolchain repository.

Note: To use the RISC-V GCC toolchain on Windows the Windows Subsystem for Linux is recommended. The guide for Ubuntu then applies.

The first step is to download the prerequisites. Open a terminal window and enter the following command:

```
sudo apt-get install autoconf automake autotools-dev curl python3 libmpc-dev \
libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool \
patchutils bc zlib1g-dev libexpat-dev
```

Navigate to a temporary directory and clone the toolchain from github:

```
git clone https://github.com/riscv/riscv-gnu-toolchain
cd riscv-gnu-toolchain
```

Configure the build for the available architectures and run make to start the build:

Note: This step may take a while.

```
./configure --prefix=/opt/riscv --with-multilib-generator="rv32i-ilp32--;"\
"rv32im-ilp32-rv32ima-;rv32imc-ilp32-rv32imac-;rv32imafc-ilp32f--"
sudo make
```

Add the build tools to the path by opening `~/.bashrc` (or equivalent) and add the line:

```
export PATH="$PATH:/opt/riscv/bin"
```

Finally reload the terminal with the following command:

```
source ~/.bashrc
```

Now the RISC-V tools are available under `riscv64-unknown-elf-(*)`

3.2 Intel Quartus Prime

To synthesize the hardware design Intel Quartus Prime is required. The lite edition is available from Intel free of charge, please make sure that the Cyclone 10 LP Device support is included.

The MCXL SoM is programmed via JTAG through a Quartus Prime compatible USB-Blaster connected to the JTAG Header on the module.

3.3 Serial Console

The reference design implements UART routed to PMod P14 compatible with the Pmod USBUART. To use the serial communication to the FPGA a console emulator is required.

3.3.1 Linux

One can install **picocom** on Linux and add the user to the dialout-group using the following commands on the terminal:

```
sudo apt install picocom
sudo usermod -a -G dialout $USER
```

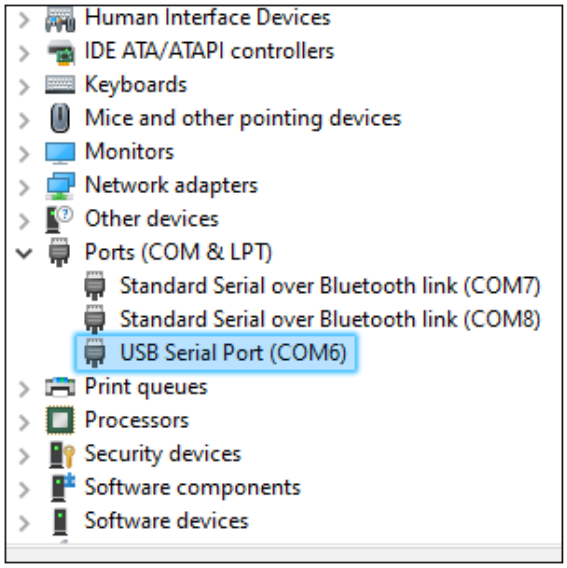
Note: Changes of the user's groups may require a logout or reboot.

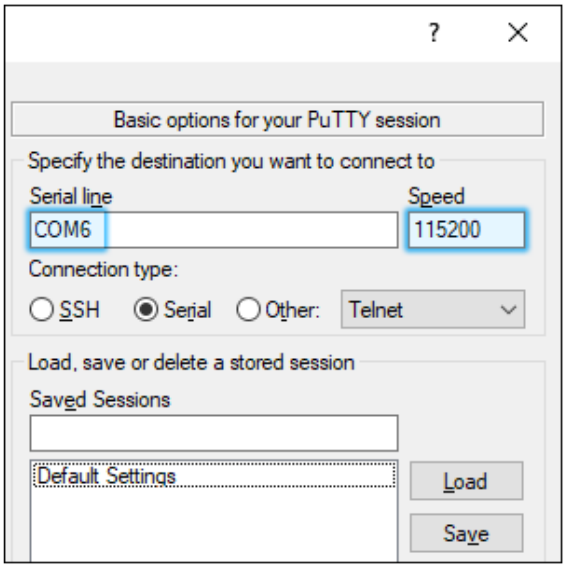
The UART on the FPGA uses a fixed baudrate of 115200, connect to the serial port with the following command. **ttyUSB0** refers to the default device name, it may be different per user.

```
picocom -b 115200 /dev/ttyUSB0
```

3.3.2 Windows

On Windows the serial port is available as **COMx** device and can be used with tools such as PuTTY or TeraTerm.





CHAPTER

FOUR

PROGRAMMING THE DEMOS

Note: The directory **Prebuild** contains a precompiled firmware image as well as a prebuild FPGA image. These may be used to skip the corresponding steps in this chapter.

4.1 Downloading the Sources

Open a terminal and use the following command to download the git repository:

```
git clone https://github.com/ARIES-Embedded/mcxl-reference-designs
```

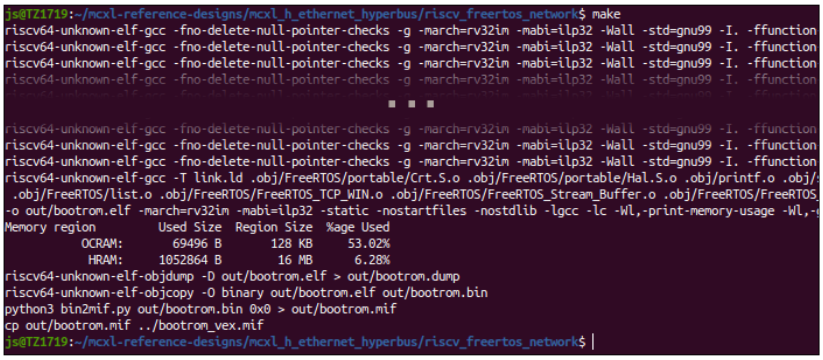
4.2 Downloading the HyperBus IP

The IP Core to interface with the HyperBus is provided by SLL with a time-limited license for ARIES customers. Download the sources from ARIES downloads and merge the content of the archive with the respective directories of the repository.

```
cd mcxl-reference-designs
wget downloads.aries-embedded.de/products/Spiderboard/software/mcxl-sll-hyperbus.zip
unzip mcxl-sll-hyperbus.zip
```

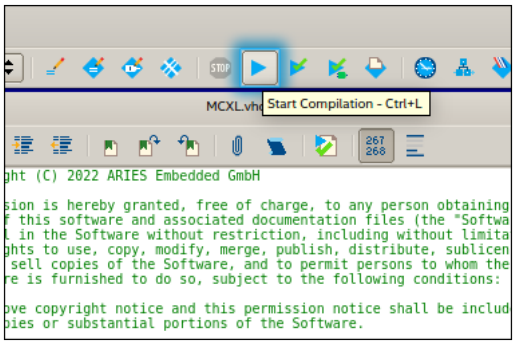
4.3 Compiling the Firmware

Navigate to the respective project directory corresponding to the module (*mcxl_h_ethernet*, *mcxl_h_ethernet_hyperbus*, *mcxl_s_ethernet*). Then navigate to the subdirectory *riscv_freertos_network*, open a terminal there and call **make**. The RISC-V-GCC will create the firmware in the subdirectory **out** and copy the **bootrom_vex.mif** file to the FPGA project directory.



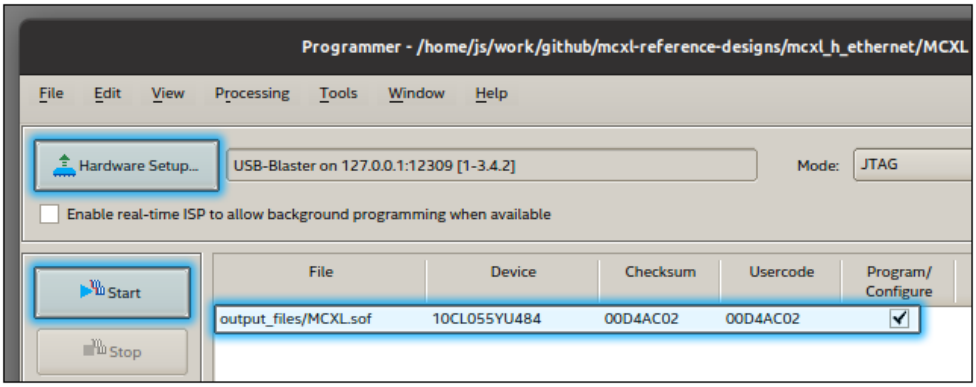
4.4 Building the Hardware Design

Start Intel Quartus Prime and open the respective project. Press **Start Compilation** and Quartus will build the FPGA image and generate the programming files in the subdirectory **output_files/**



4.5 Programming the FPGA

Open the Quartus Programmer and under **Hardware Setup...** select the USB-Blaster. Then select the **.sof** file either in the project subdirectory **output_files/** or in the directory with the prebuild files and press **Start**.



CHAPTER

FIVE

REFERENCE DESIGNS

The reference design implements the following IP cores:

- RISC-V Core **VexRiscv** (RV32IM)
- 128 KiB On-Chip RAM
- UART
- GPIO
- Intel TSE MAC (Gigabit Ethernet)
  - Custom Avalon-Streaming to Avalon-MemoryMapped Bridge

The `hyperbus`-design also implements:

- SLL MBMC (HyperBus) interfacing with:
  - 2x 64 MiB HyperRAM
  - 2x 256 MiB HyperFlash

5.1 FPGA Design

The top-level file for the FPGA design is **MCXL.vhd** and provides the port declaration to interface with the physical pins of the device. It also declares and instantiates the Qsys component. The MCXL provides a 25 MHz clock for the FPGA.

5.1.1 Intel Platform Designer Qsys

The Intel Platform Designer implements the RISC-V system. The CPU core and peripheral devices are instantiated, configured and communicate via the Avalon Interconnect. Each device occupies a memory range in the address space, the interconnect will automatically resolve the access signals.

5.1.1.1 RISC-V Core VexRiscv

The VexRiscv core is configured as RV32IM core (without caches) but can be also configured as RV32I without cache or as RV32IM, RV32IMAC, RV32IMAFC with 4KiB instruction and data caches. For different configurations the `Makefile` of the firmware needs to be updated. The line `ARCH := rv32im` should reflect the capabilities of the core, if the RV32IMAFC configuration is chosen, the `ABI` should also be set to `ilp32f`. When using a configuration with caches, the IO region parameter should be configured to include all IO devices. As alternative to VexRiscv the SERV core or PicoRV32 core can be implemented, see the riscv-on-max10 repository for more information.

Parameter	Description
Reset Vector	Address loaded into the program counter when the core starts.
Exception Vector	Address loaded into the program counter when an exception (interrupt or trap) occurs.
IO Region Begin	First (inclusive) address of the uncached region; volatile memory such as registers of external modules are required to be in this region. Does not have an effect if no caches are used.
IO Region End	Last (inclusive) address of the uncached region. Does not have an effect if no caches are used.
Core Configuration	Specifies the implemented instruction set and caches.

5.1.1.2 Peripherals

The Qsys System implements a UART, GPIO and a On-Chip Memory core. The Memory core provides the VexRiscv with 128 KiB RAM and is initialized with the firmware. The UART core is routed to the PMod P14 connector and provides a serial interface to the RISC-V core. The GPIO core PMod connectors TODO

5.1.1.3 Ethernet

The FPGA implementation for ethernet consists of the Intel Triple Speed Ethernet (TSE) MAC configured for gigabit operation and a custom Streaming-To-MemoryMapped `AvalonST2MM` core, which connects the TSE to the VexRiscv core and can buffer 2048 bytes, i.e. one ethernet packet, for both transmit and receive direction. The TSE can buffer further 128 KiB while the `AvalonST2MM` is occupied.

5.2 Firmware

The firmware for the RISC-V core is build for bare-metal execution. The `mcxl_[hs]_ethernet` design stores the firmware binary with all sections in the on-chip memory. While the `Crt` zero-initializes the `bss` section, changes to variables in the data section are not restored on reset. As such a proper reset requires the reprogramming or restart of the FPGA via programmer or configuration device.

For the `mcxl_h_ethernet_hyperbus` project with HyperBus enabled the linker-script includes `hyperram0` as available memory. The on-chip memory then is declared as read-only (though it can still be written) and contains the text section (program code) and the data section as load address space. The hyperram0 provides space for the `bss` section and hosts the data section as virtual address space. On startup the `Crt` reads the data section from the on-chip memory and copies it to the HyperRAM, the firmware uses the data section stored in the HyperRAM when accessing initialized variables. Additionally the `Crt` also initializes the `bss` section with zero. Since the firmware does not modify the content of the on-chip memory, the core can be reset via soft reset.

After core initialization the `Crt` calls the `main` function and starts the user firmware. The provided examples implement FreeRTOS+TCP to simplify access to the ethernet functionality of the board. For demonstration purposes the firmware prints firmware information and loopback on UART, bounces LEDs on PMod P15 to P18 and connects to network using DHCP to provide basic functionality such as `ping` response.

FreeRTOS was modified to provide task switching in a compliant manner, functions to access the `task control block` from the `system tick handler` were added. The reference design with FreeRTOS+TCP depend on a number of configuration parameters and function stubs which may be modified for their intended functionality. The files `FreeRTOSConfig.h` and `FreeRTOSIPConfig.h` contain the standard configuration for FreeRTOS and the TCP stack respectively. The TCP stack uses DHCP to acquire an IP address but will fallback to a predefined configuration upon failure. This configuration is found in `Config.h`.