Mercury ZX1 SoC Module

Reference Design for Mercury+ PE1 Base Board User Manual

Purpose

The purpose of this document is to present to the user the overall view of the Mercury ZX1 SoC module reference design and to provide the user with a step-by-step guide to the complete Xilinx® SoC design flow used for the Mercury ZX1 SoC module.

Summary

This document first gives an overview of the Mercury ZX1 SoC module reference design and then guides through the complete Xilinx SoC design flow for the Mercury ZX1 SoC module in the getting started section. In addition, the internals and the boot options of the Mercury ZX1 SoC module reference design are described.

Product Information	Number	Name
Product	ME-ZX1	Mercury ZX1 SoC Module

Document Information	Reference	Version	Date
Reference / Version / Date	D-0000-403-001	05	12.03.2018

Approval Information	Name	Position	Date
Written by	DIUN	Design Engineer	04.01.2016
Verified by	GLAC	Design Engineer	15.01.2016
Approved by	RPAU	Quality Manager	16.01.2018

Copyright Reminder

Document History

Version	Date	Author	Comment
05	12.03.2018	RLOC	Fixed Led to Gpio assignment
04	16.01.2018	DDUE	Updated to Vivado and SDK 2017.4, other style updates and clarifications
03	10.06.2016	DIUN	Minor style updates and clarifications
02	02.02.2016	DIUN	Corrected switch configuration
01	27.01.2016	MOBE	Version 01

Table of Contents

1 Overview

1.1 Introduction

The Mercury ZX1 SoC module reference design demonstrates a system using the Mercury ZX1 SoC module in combination with the Mercury+ PE1 base board (any board variant) or with the regular Mercury PE1 base board (previous revisions). It presents the basic configuration of the device and features some example applications.

A troubleshooting section is included at the end of the document, to help the user solve potential issues related to board connectivity and/or system functionality.

Please note that the features presented in the reference design depend on the employed base board, therefore some features may not be available in certain hardware configurations.

An introduction to the Xilinx tools is provided by the documents below:

- Zynq®7000 All Programmable SoC Embedded Design Tutorial [1]
- Vivado Design Suite User Guide, Embedded Processor Hardware Design [2]
- Vivado Design Suite Tutorial, Embedded Processor Hardware Design [3]

More information on the Mercury ZX1 SoC module and the Mercury+ PE1 base board can be retrieved from their respective user manuals [5] [6].

1.2 Directory Structure

The Mercury ZX1 SoC module reference design is delivered as a ZIP archive file with the following directory structure and contents:

- `binaries` — Pre-compiled binaries directory
- `scripts` — Scripts directory required for Vivado project creation
- `SdkExport` — Pre-generated hardware description files required for SDK applications
- `software` — Software projects directory
- `src` — Xilinx pinout and timing constraints and VHDL source code directory
- `Mercury_ZX1_Reference_Design_for_Mercury_PE1_User_Manual.pdf` — User manual (this document)

1.3 Prerequisites

- IT
  - A computer with a microSD card slot (optional[1]) running Windows 7 64-bit (or later)
- Software
  - Xilinx Vivado 2017.4 WebPack, Evaluation, Design or System Edition (check the Mercury ZX1 SoC Module User Manual [5] for details on device support in Xilinx tools)
  - Xilinx Software Development Kit (SDK) 2017.4
  - Enclustra Module Configuration Tool (MCT) [7] (optional[2])
  - A terminal emulation program (e.g. Tera Term)
  - PuTTY (optional[3])
- Hardware
  - An Enclustra Mercury ZX1 SoC module
  - An Enclustra Mercury+ PE1 base board (any board variant) or a regular Mercury PE1 base board (previous revisions)
- Accessories
  - A standard micro USB cable or a standard type B USB cable[4]
  - A Xilinx JTAG programmer (e.g. Platform Cable USB) download cable (optional[5])
  - A microSD card (optional[1])

[1]Only required for SD card boot mode
[2]May be used for flash programming, for SoC device configuration or for FTDI configuration.
[3]Required for running the Ethernet lwIP example
[4]Both USB cables are required for running the USB storage emulation example.
[5]The FTDI device present on the base board can be configured to Xilinx JTAG mode using the Enclustra MCT software [7].

2 Reference Design Description

2.1 Block Diagram

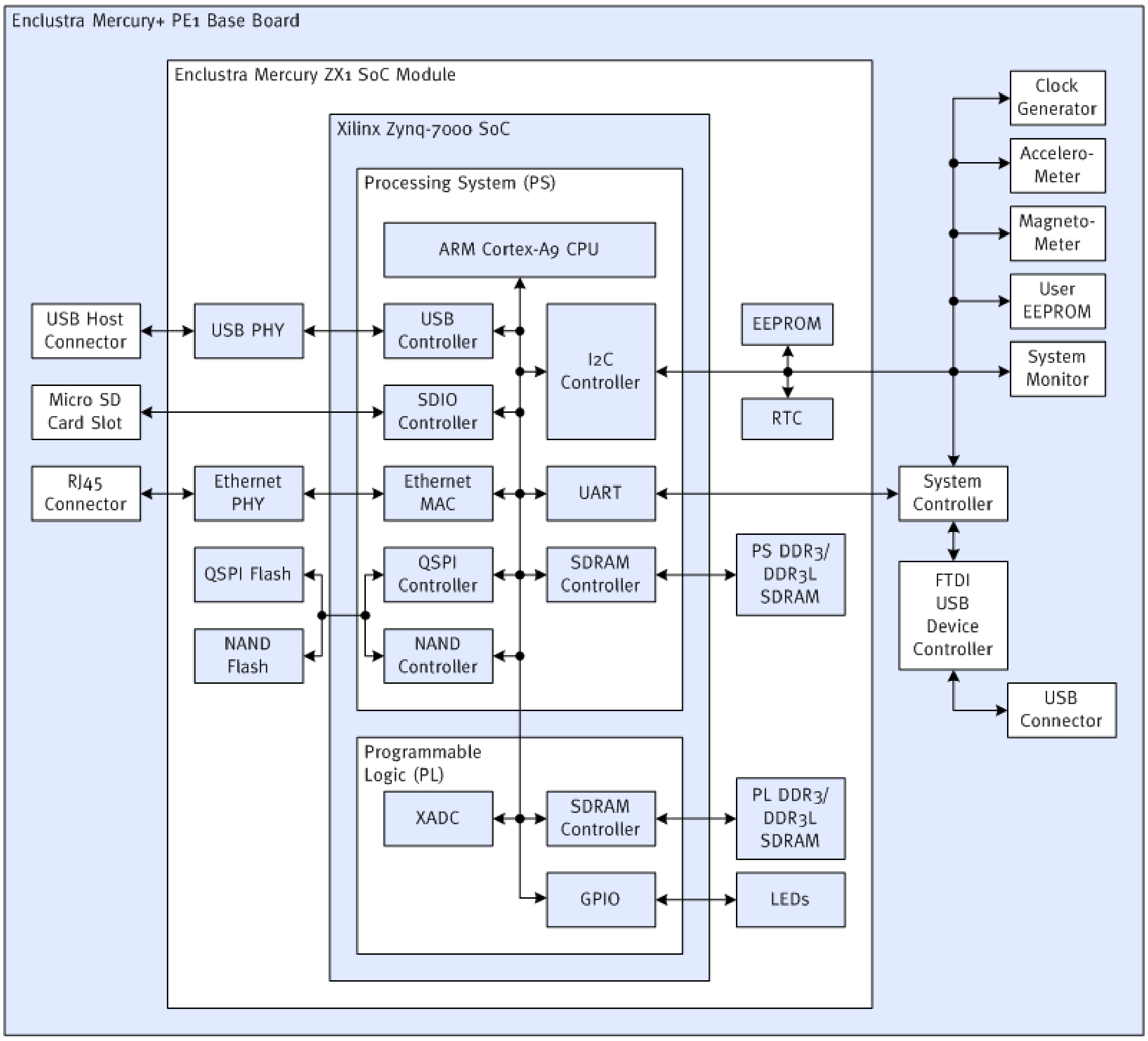


Figure 1: Hardware Block Diagram

2.2 Processing System (PS)

2.2.1 Clocks

The PS input clock frequency is configured to 33.33 MHz, while the CPU clock frequency is configured to 666.66 MHz. The CPU clock frequency multiplier can be modified in the Zynq system. A 50 MHz clock is exported from PS to the PL.

2.2.2 PS DDR3/DDR3L SDRAM

The Mercury ZX1 SoC module has two DDR3/DDR3L memory interfaces: one is connected to the PS, while the other is connected to the PL.

The PS DDR3/DDR3L SDRAM memory runs at 533 MHz (1066 Mbit/sec) at a voltage of 1.5 V by default. These parameters can be modified in the Zynq system. For a voltage of 1.35 V, beside the changes in the PS, it is necessary to change the top level assignment to DDR3_VSEL signal from high impedance to logic low.

The DDR settings in the Zynq system must be configured according to the Mercury ZX1 SoC Module User Manual [5].

2.2.3 SD Card

The SD card is configured in the PS to the MIO 40..45 pins. This enables SD card access, as well as booting from the SD card.
To allow the Mercury ZX1 SoC module to boot from the SD card, the hardware configuration on the Mercury+ PE1 base board must be done according to Section 4.2.2.

Please note that the SD pins are shared between the PS and the PL.

Warning!

Because the MIOs 40..45 are connected to FPGA pins AA13, AA12, AB17, AB16, AC17, AC16 in parallel, make sure the FPGA pins are in high impedance state before driving the PS SD pins and vice versa.

In order to be able to access and write to the SD card from Linux, it is recommended to enable the Write Protect (WP) and Card Detect (CD) pins in the PS, map them to EMIO pins and tie them off to high, respectively low state. An example is available in the reference design sources.

2.2.4 I2C

The I2C controller I2C0 is configured to the EMIO pins. Table 1 lists the connected devices on the Mercury ZX1 SoC module and Mercury+ PE1 base board.

Device	Address (7-bit)	Vendor	Part Type
Real-time clock	0x6F	Intersil	ISL12020MIRZ
RTC user SRAM	0x57	Intersil	ISL12020MIRZ
Secure EEPROM	0x64	Atmel	ATSHA204A-MAHDA-T
System controller	0x0D	Lattice Semiconductor	LCMXO2-4000HC-6MG132I
System monitor	0x2F	Texas Instruments	LM96080CIMT/NOPB
Clock generator	0x70	Silicon Labs	SI5338B-B-GMR
User EEPROM	0x57	Microchip	24AA128T-I/MNY
Accelerometer	0x1D	ST Microelectronics	LSM303CTR
Magnetometer	0x1E	ST Microelectronics	LSM303CTR

Table 1: I2C Devices

Please note that, depending on the employed base board, some of the I2C devices may not be available in certain hardware configurations.

The device vendors or addresses of the I2C devices may change in future revisions of Mercury ZX1 SoC module or Mercury+ PE1 base board.

For detailed information on the I2C devices, please refer to the corresponding user manuals [5] [6].

2.2.5 Quad SPI Flash Controller

The quad SPI flash controller is connected to MIO 1..6 and 8 pins in Single Slave Select mode. MIO 2..6 pins are shared between NAND flash and QSPI flash on the Mercury ZX1 SoC module. Please refer to the Mercury ZX1 SoC Module User Manual [5] for details about flash programming and usage.

To allow the Mercury ZX1 SoC module to boot from the QSPI flash, the hardware configuration on the Mercury+ PE1 base board must be done according to Section 4.1.2.

2.2.6 UART

The UART0 is mapped to MIO 46..47 pins and connected to the FTDI USB device controller on the Mercury+ PE1 base board. The UART is configured as shown in Table 2.

Parameter	Value
Baud rate	115’200
Data	8 bit
Parity	None
Stop	1 bit
Flow control	None

Table 2: UART Configuration

Warning!

Because the MIOs 46..47 are connected to FPGA pins AA15 and AA14 in parallel, make sure the FPGA pins are in high impedance state before driving the PS UART pins and vice versa.

2.2.7 Ethernet

The Ethernet MAC ENET0 is mapped to MIO 16..27 and 52..53 pins and is connected to a Micrel KSZ9031 Ethernet PHY on the Mercury ZX1 SoC module using RGMII interface. The PHY can be configured via the MDIO management interface on PHY address 3.

Please note that the RGMII delays in the Ethernet PHY need to be configured before the Ethernet interface can be used. In the reference design this is done in the First Stage Boot Loader (FSBL) and in the Ethernet lwIP example.

2.2.8 USB

The USB controller USB0 on MIO 28..39 pins is connected to a USB3320C USB 2.0 PHY. This interface can be configured for USB host, USB device and USB On-The-Go (OTG) operations.

Depending on the required USB mode, the settings in the system controller and the DIP switches on the Mercury+ PE1 base board must be configured correctly. Please refer to the Mercury+ PE1 Base Board User Manual [6] for details.

2.2.9 GPIOs

The unused MIO pins from the PS are available as GPIOs. They are mapped to MIO pins 15 and 48..51 in the PS. The function of the general purpose pins on the Mercury ZX1 SoC module is described in Table 3. The shared pins can be used either in the PL or in the PS.

GPIO	Signal	Function
MIO 15	MDIO_SEL_LED3#	MDIO select/LED3# shared signal, see Mercury ZX1 SoC Module User Manual [5] for details
MIO 48	GPIO 48	GPIO, shared with FPGA pin Y16
MIO 49	GPIO 49	GPIO, shared with FPGA pin Y15
MIO 50	GPIO 50	GPIO, shared with FPGA pin W16
MIO 51	GPIO 51	GPIO, shared with FPGA pin W15

Table 3: PS GPIO Configuration

Warning!

Because the MIOs 48..51 are connected to FPGA pins Y16, Y15, W16, W15 in parallel, make sure the FPGA pins are in high impedance state before driving the PS GPIO pins and vice versa.

2.3 Programmable Logic (PL)

2.3.1 PL DDR3/DDR3L SDRAM

The PL DDR3/DDR3L SDRAM memory runs at 400 MHz (800 Mbit/sec) at a voltage of 1.5 V by default. These parameters can be modified in the Memory Interface Generator (MIG) IP core. For a voltage of 1.35 V, beside the changes in the IP core, it is necessary to change the top level assignment to DDR3_VSEL signal from high impedance to logic low.

The DDR settings in the MIG IP core must be configured according to the Mercury ZX1 SoC module User Manual [5].

2.3.2 GPIOs

A Xilinx GPIO controller in the PL is connected to the PS via an AXI bus. The least significant bit of the PL GPIO bus is connected to LED0# in the top level, as described in Table 4.

The FPGA firmware contains a 24-bit counter freely running at 50 MHz. The MSB of this counter is used to blink LED2# on FPGA pin H9 with a frequency of approximately 3 Hz. LED1# is driven by the same counter and blinks twice as fast as LED2#.

FPGA Pin	Signal	Function
H7	FPGA_LED0#	GPIO 0, controlled by the PL GPIO controller
H6	FPGA_LED1#	Blinking LED counter MSB-1
H9	FPGA_LED2#	Blinking LED counter MSB

Table 4: FPGA Firmware I/O Configuration

2.3.3 XADC

A Xilinx XADC IP core instance is connected to the PS via an AXI bus, in order to monitor the temperature of the device. The temperature threshold for this module is configured to the industrial applications temperature, 85° Celsius.

The constraints provided in the reference design enable FPGA bitstream power-down, when the temperature increases above the threshold. In this case, the PL will be reset, while the ARM processor will still be running.

Depending on the user application, the Mercury ZX1 SoC module may consume more power than can be dissipated without additional cooling measures; always make sure the SoC is adequately cooled by installing a heat sink and/or providing air flow. Temperature control and monitoring is very important in a complex design.

Information that may assist in selecting a suitable heat sink for the Mercury ZX1 SoC module can be found in the Enclustra Modules Heat Sink Application Note [8].

3 Getting Started

This section describes the steps required to configure the Mercury ZX1 SoC module and Mercury+ PE1 base board in order to run the example applications. The section includes information on how to:

- Mount the module and configure the base board
- Generate the FPGA bitstream
- Prepare the software workspace
- Run the software applications

The example applications, including the expected results of running the software, are described in detail in Section 3.7.

3.1 Essential Information

Warning!

Never mount or remove the Mercury ZX1 SoC module to or from the Mercury+ PE1 base board while the Mercury+ PE1 base board is powered. Always remove or turn off the power supply before mounting or removing the Mercury ZX1 SoC module.

Warning!

It is possible to mount the Mercury ZX1 SoC module the wrong way round on the Mercury+ PE1 base board - always check that the mounting holes on the Mercury+ PE1 base board are aligned with the mounting holes of the Mercury ZX1 SoC module.

The base board and module may be damaged if the module is mounted the wrong way round and powered up.

Warning!

Depending on the user application, the Mercury ZX1 SoC module may consume more power than can be dissipated without additional cooling measures; always make sure the SoC is adequately cooled by installing a heat sink and/or providing air flow.

Warning!

Please read carefully the Mercury ZX1 SoC module and Mercury+ PE1 base board user manuals before proceeding.

Note that when Enclustra MCT [7] is used for SoC configuration or flash programming, all other tools that may be connected to the FTDI device (e.g. Vivado Hardware Manager, SDK, UART terminal) must be closed.

3.2 Hardware Setup

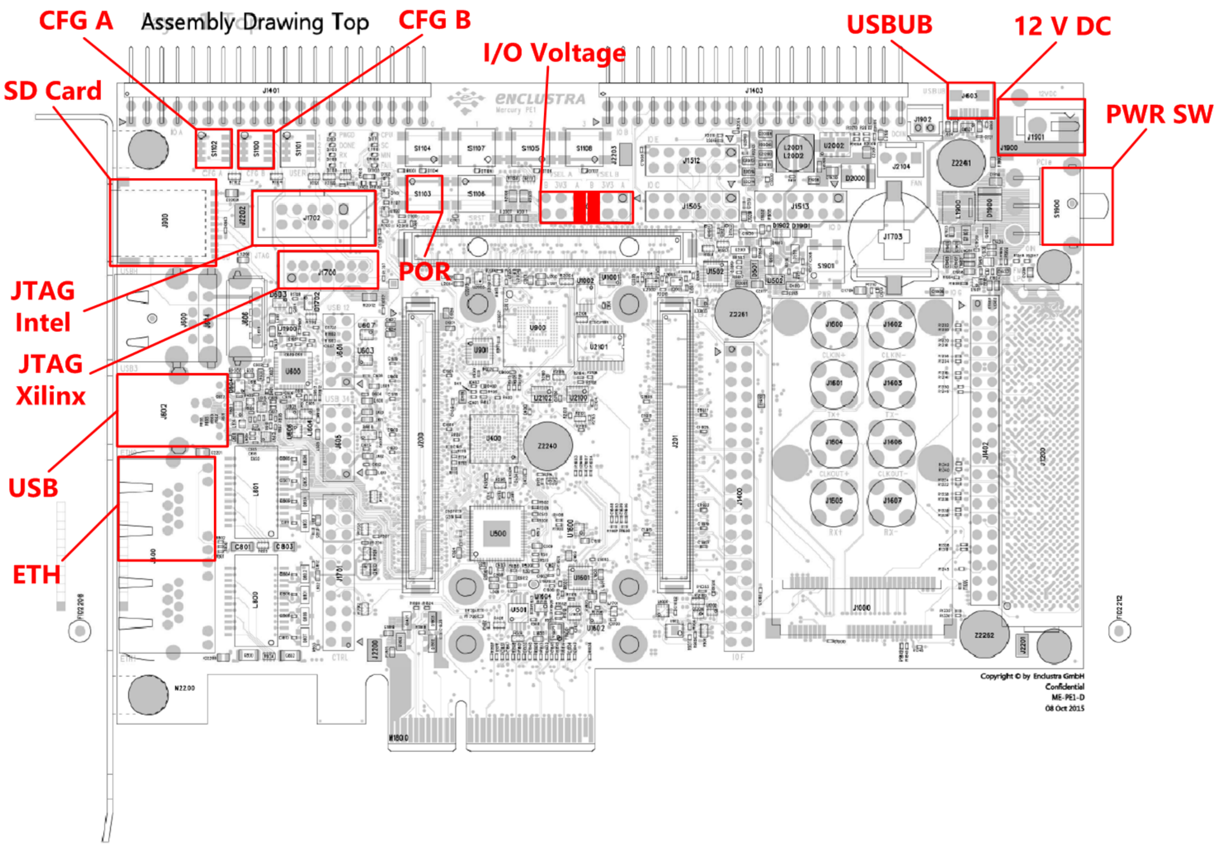


Figure 2: Mercury+ PE1 Base Board Assembly Drawing (Top View)

Step	Description
1	Set the I/O voltage jumpers on the Mercury+ PE1 base board according to label **I/O Voltage** in Figure 2 (the jumpers are marked with red rectangles): • VSEL A = 2.5 V (position A) • VSEL B = 1.8 V (position B)
2	Set the configuration DIP switches on the Mercury+ PE1 base board as follows (see labels **CFG A** and **CFG B** in Figure 2): • CFG A = [1: OFF, 2: OFF, 3: OFF, 4: ON] • Depending on the available USB cable, the CFG B DIP switches must be configured as follows: • For micro USB cable: CFG B = [1: OFF, 2: OFF, 3: ON, 4: ON] • For type B USB cable: CFG B = [1: ON, 2: ON, 3: ON, 4: ON]
3	Mount the Mercury ZX1 SoC module to the Mercury+ PE1 base board. Make sure that the mounting holes of the Mercury ZX1 SoC module are aligned with the mounting holes of the Mercury+ PE1 base board before proceeding.

Continued on next page...

Step	Description
4	Connect the USB cable between your computer and the Mercury+ PE1 base board. Use the micro USB port labeled **USBUB** in Figure 2 or the type B USB port labeled **USB**, according to your hardware configuration.
5	Connect the 12 V DC power supply plug to the power connector of the Mercury+ PE1 base board (see label **12 V DC** in Figure 2).
6	Set the power switch of the Mercury+ PE1 base board to ON (see label **PWR SW** in Figure 2).
7	Connect the Xilinx Platform Cable USB to the JTAG connector of the Mercury+ PE1 base board (see label **JTAG Xilinx** in Figure 2). Alternatively, JTAG over USB may be used. For this, the FTDI device on the Mercury+ PE1 base board must be configured to Xilinx JTAG mode using Enclustra MCT [7]. Details on the Xilinx JTAG mode configuration are presented in the Mercury+ PE1 Base Board User Manual [6]
8	Open a terminal program on your computer (e.g. Tera Term) and open a serial port connection using the COM port labeled with the higher number from the two newly detected ports. For issues related to COM ports detection, refer to Section 5.4. Configure the UART parameters according to Section 2.2.6.

Table 5: Hardware Setup Step-By-Step Guide

Warning!

Please make sure that a single JTAG adapter is connected to the base board and enabled at a given moment, otherwise the development tools may report errors during JTAG connecting attempts.

3.3 FPGA Bitstream Generation

For a fast test of the provided software applications, the pre-generated bitstream included in the `binaries` directory may alternatively be used, therefore the steps described in this section may be skipped.

The `<base_dir>\binaries` directory includes bitstream files for any SoC device that may be equipped on the module.

Step	Description
1	Edit the `fpga_part` variable in `scripts\settings.tcl` file, according to your SoC device. This file includes module name and board information required for the project creation script. All settings, except for `fpga_part` should be left on default. The list of options for `fpga_part` is given in the comments within the Tcl file. Save the file after editing.
2	Start Xilinx Vivado 2017.4 and create the Mercury ZX1 SoC module reference design project: 1. Click on the Tcl console at the bottom of the page and type: (a) `cd <base_dir>` (`<base_dir>` is the directory in which you extracted the archive contents). Note that you must use / for hierarchy separator, instead of \\. (b) `source scripts/create_project.tcl` 2. Wait for completion
3	Run Synthesis, Implementation & Bitstream Generation in Vivado 2017.4: 1. Click on Generate Bitstream from the Flow Navigator bar 2. In the Launch Runs window click OK - this will start automatically the entire implementation process 3. Wait for completion → select View Reports → OK
4	Export the hardware system information (required for the SDK projects): 1. File → Export → Export Hardware 2. Enable Include bitstream checkbox If you want to save the .hdf file to another path than the default location: 3. Click on Choose Location → Select 4. Hit OK

Table 6: FPGA Bitstream Generation Step-By-Step Guide

3.4 SDK Workspace Preparation

This section describes how to create and import the software applications. The steps are generic, and apply to all software examples provided in the release archive, along with this document.

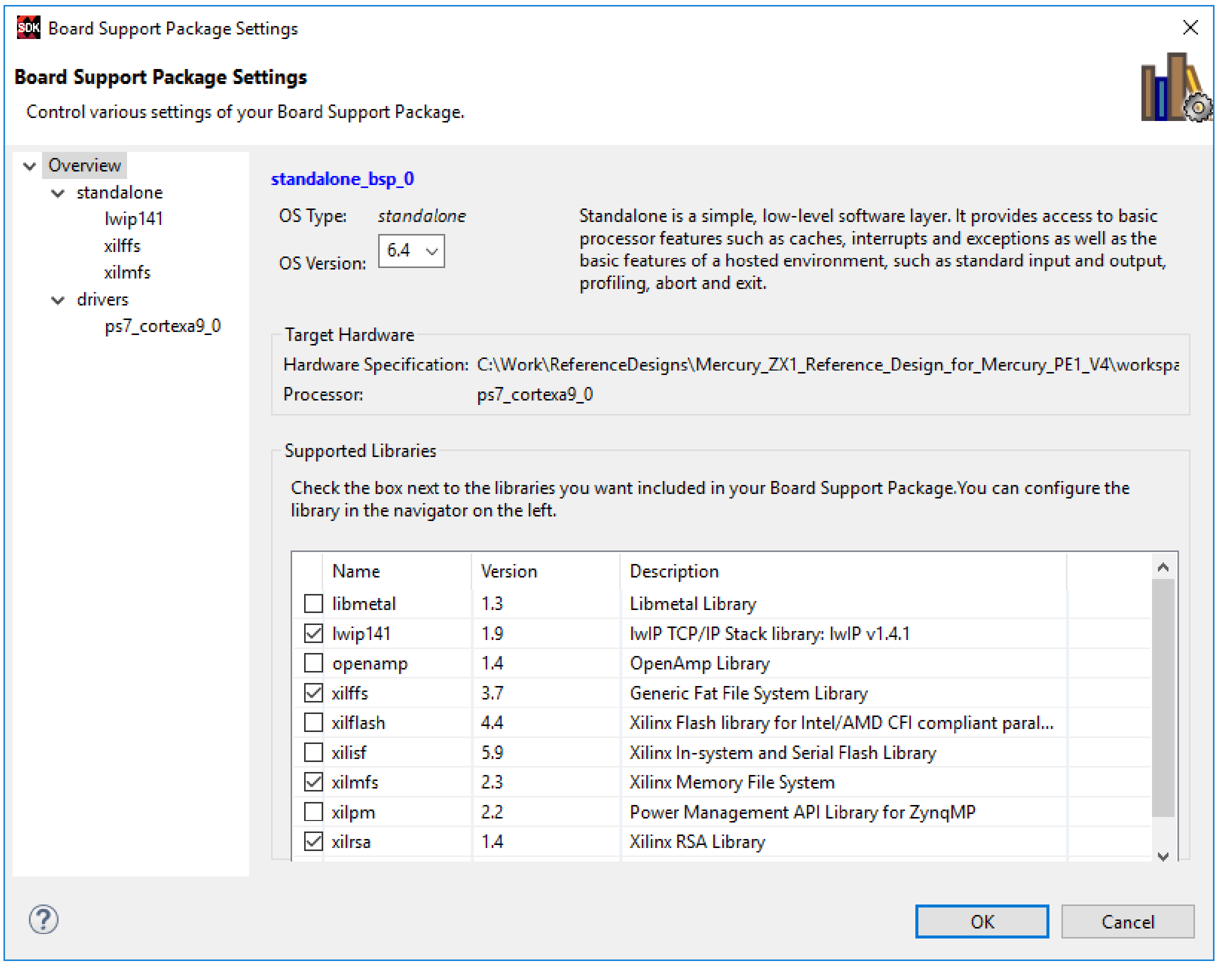
The `<base_dir>\SdkExport` directory includes pre-generated hardware description files for any SoC device that may be equipped on the module.

Step	Description
1	Start Xilinx SDK 2017.4 1. Select any workspace (e.g. `<base_dir>\workspace`)
2	Create a new board support package (BSP) 1. File → New → Board Support Package → Specify 2. In the New Hardware Project window: (a) For Project Name type hw_platform_0 (b) For Target Hardware Specification select the .hdf file you exported from Vivado, as described in Section 3.3. The default location used by Vivado is `<base_dir>\<vivado_proj_dir>\<project_name>.sdk\system_top.hdf` Alternatively, the pre-compiled hardware description file from `<base_dir>\SdkExport` may be used. (c) Hit Finish 3. In the New Board Support Package Project window: (a) For Project Name type standalone_bsp_0 (b) For Hardware Platform select hw_platform_0 (c) For CPU select ps7_cortexa9_0 (d) Hit Finish 4. In the Board Support Package Settings window (see Figure 3): (a) Enable the checkboxes corresponding to the drivers and libraries required in the BSP: lwip141, xilffs, xilmfs, xilrsa (b) Hit OK
3	Create new First Stage Boot Loader application 1. File → New → Application Project 2. In the Application Project window: (a) For Project Name type FSBL (b) For Board Support Package select Use existing, and select standalone_bsp_0 (c) Leave the other settings on default values (d) Hit Next 3. In the Templates window: (a) Select Zynq FSBL (b) Hit Finish 4. Copy the files provided in `<base_dir>\software\FSBL\src` directory from the reference design release archive to `<workspace>\FSBL\src` directory (and replace the fsbl_hooks.c file). This step is required as per explanation in Section 3.7.8.

Continued on next page...

Step	Description
4	Import the example projects into the workspace (see Figure 4): 1. File → Import... 2. Select General → Existing Projects into Workspace and hit Next 3. For Select root directory choose `<base_dir>\software` and hit OK 4. Enable checkbox Copy projects into workspace 5. Hit Finish
5	Build all projects 1. Hit Ctrl-B and wait for completion

Table 7: SDK Workspace Preparation Step-By-Step Guide


Board Support Package Settings

Board Support Package Settings

Control various settings of your Board Support Package.

standalone_bsp_0

OS Type: *standalone*

OS Version: 6.4

Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.

Target Hardware

Hardware Specification: C:\Work\ReferenceDesigns\Mercury_ZX1_Reference_Design_for_Mercury_PE1_V4\workspa

Processor: ps7_cortexa9_0

Supported Libraries

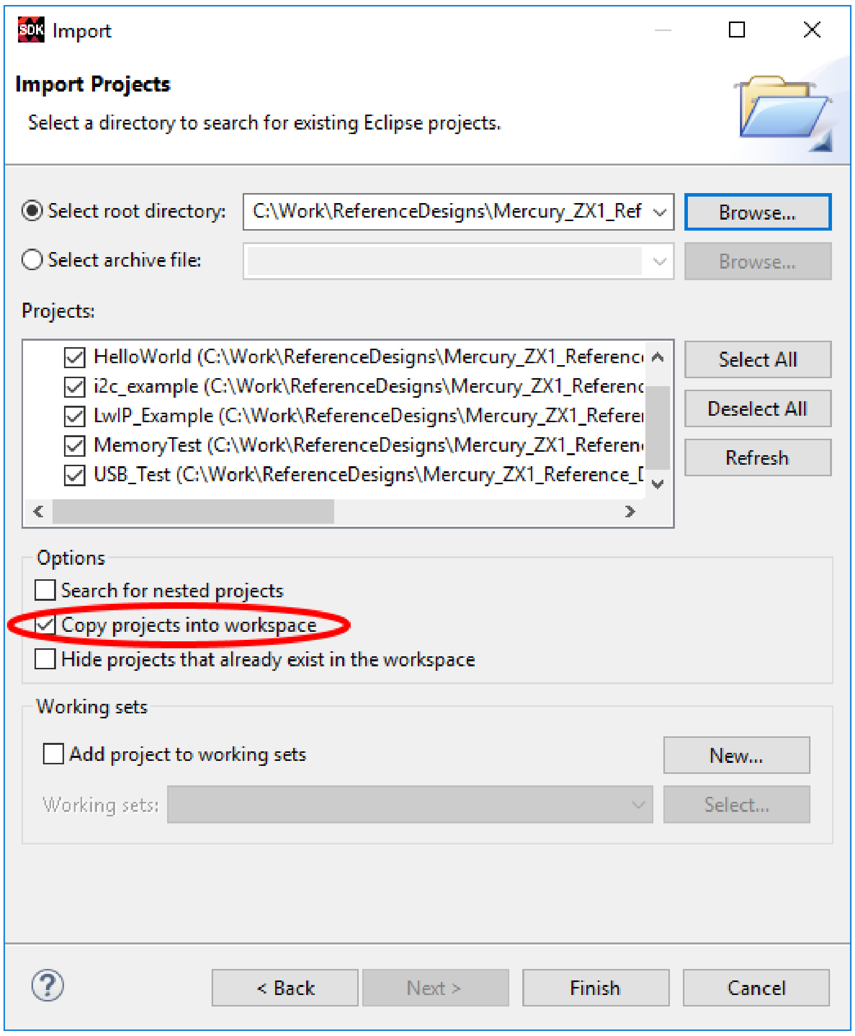
Check the box next to the libraries you want included in your Board Support Package.You can configure the library in the navigator on the left.

	Name	Version	Description
☐	libmetal	1.3	Libmetal Library
☑	lwip141	1.9	lwIP TCP/IP Stack library: lwIP v1.4.1
☐	openamp	1.4	OpenAmp Library
☑	xilffs	3.7	Generic Fat File System Library
☐	xilflash	4.4	Xilinx Flash library for Intel/AMD CFI compliant paral...
☐	xilisf	5.9	Xilinx In-system and Serial Flash Library
☑	xilmfs	2.3	Xilinx Memory File System
☐	xilpm	2.2	Power Management API Library for ZynqMP
☑	xilrsa	1.4	Xilinx RSA Library

Figure 3: Board Support Package Settings

Warning!

For a successful build of the software examples the hardware project must be named "hw_platform_0" and that the BSP must be named "standalone_bsp_0".


Import

Import Projects

Select a directory to search for existing Eclipse projects.

(●) Select root directory: C:\Work\ReferenceDesigns\Mercury_ZX1_Ref

( ) Select archive file:

Projects:

- [✓] HelloWorld (C:\Work\ReferenceDesigns\Mercury_ZX1_Referenc
- [✓] i2c_example (C:\Work\ReferenceDesigns\Mercury_ZX1_Referenc
- [✓] LwIP_Example (C:\Work\ReferenceDesigns\Mercury_ZX1_Refere
- [✓] MemoryTest (C:\Work\ReferenceDesigns\Mercury_ZX1_Referen
- [✓] USB_Test (C:\Work\ReferenceDesigns\Mercury_ZX1_Reference_

Options

- [ ] Search for nested projects
- [✓] Copy projects into workspace
- [ ] Hide projects that already exist in the workspace

Working sets

- [ ] Add project to working sets

Working sets:

Figure 4: Importing Software Projects

Warning!

Please note that the software applications may not work properly if they are imported in another software version than the one specified in this document. A solution for such cases is creating a fresh software project with the sources provided in the reference design.

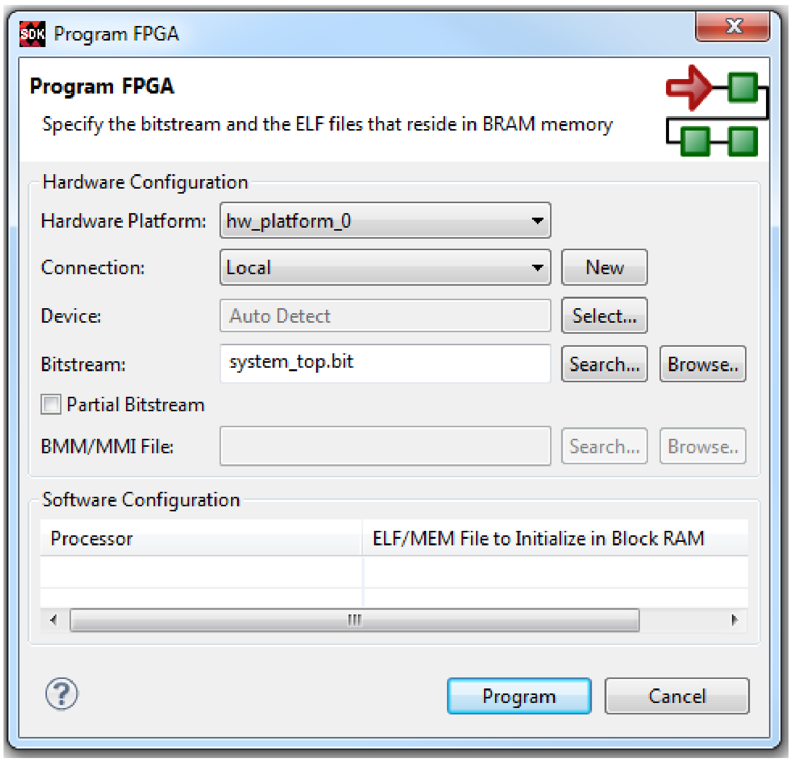
Warning!

Please make sure that during import process the checkbox for copying the projects into workspace has been enabled, otherwise the build step will fail due to incorrect file paths.

3.5 FPGA Programming

Step	Description
1	Open Xilinx SDK 2017.4: 1. Click on Xilinx Tools → Program FPGA 2. For Hardware Platform select hw_platform_0 3. For Bitstream field hit Search → select system_top.bit 4. Hit Program The configuration is shown in Figure 5. For issues related to JTAG connection, refer to Section 5.3.
2	After the FPGA is successfully configured, the **DONE** LED should be lit.

Table 8: FPGA Programming Step-By-Step Guide



Program FPGA

Program FPGA

Specify the bitstream and the ELF files that reside in BRAM memory

Hardware Configuration

Hardware Platform: hw_platform_0

Connection: Local

Device: Auto Detect

Bitstream: system_top.bit

Partial Bitstream

BMM/MMI File:

Software Configuration

Processor	ELF/MEM File to Initialize in Block RAM

Figure 5: FPGA Programming Settings

3.6 Running Software Applications

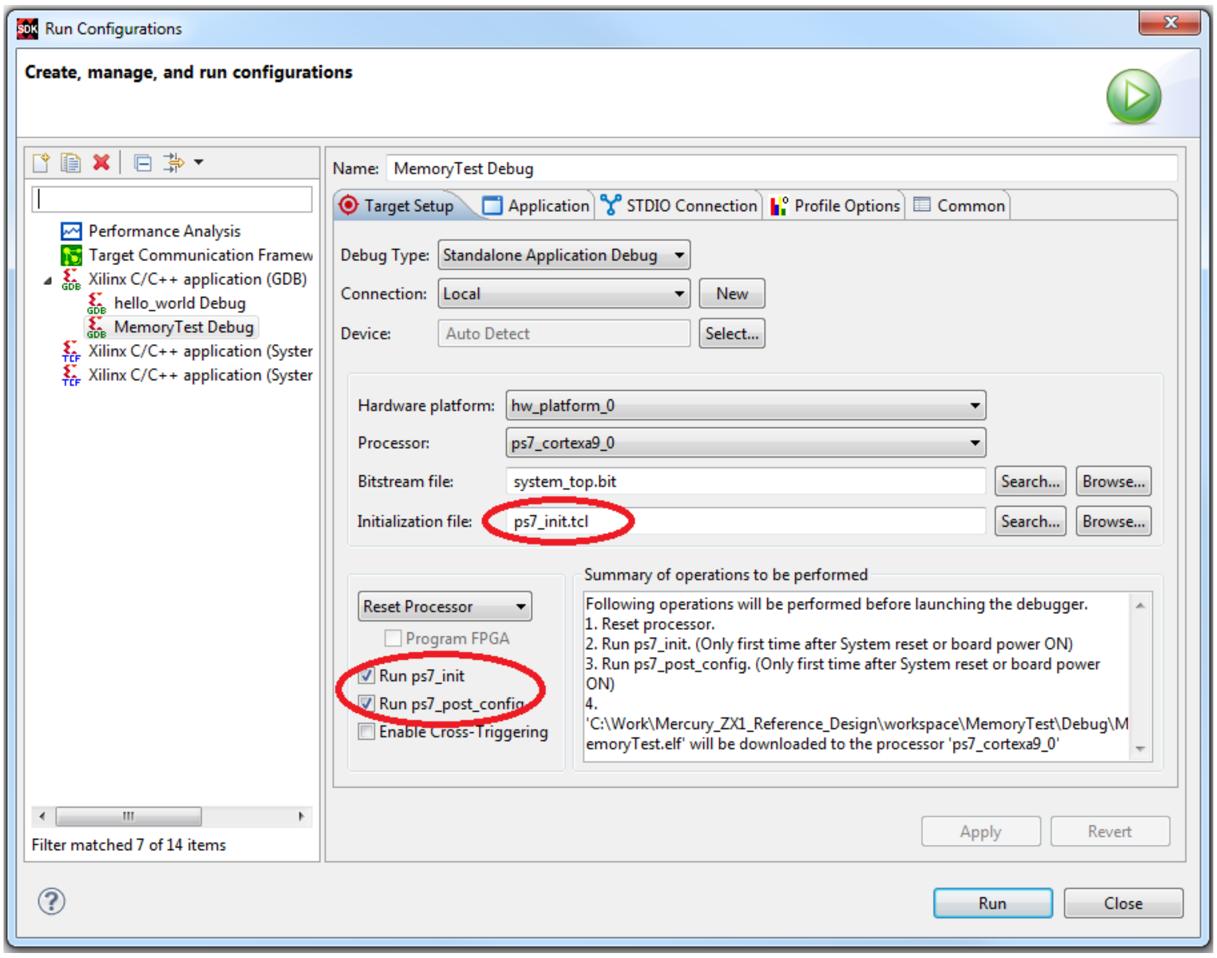
This section describes how to run software applications on the Mercury ZX1 SoC module. The steps are generic, and apply to all software examples provided in the release archive, along with this document.

In order to execute the applications, the hardware needs to be configured as described in Section 3.2.

Note that the FPGA must be programmed before running the software applications. Refer to Section 3.5 for details on FPGA programming.

Step	Description
1	Create a run configuration for the application in SDK 2017.4: 1. Run → Run Configurations... 2. Right-click Xilinx C/C++ application (GDB) and hit New or double-click on Xilinx C/C++ application (GDB) 3. Enter a run configuration name in the Name field (e.g. HelloWorld) 4. Target Setup tab (see Figure 6): (a) For Hardware Platform select hw_platform_0 (b) For CPU select ps7_cortexa9_0 (c) In the Bitstream file field, hit Search... (d) Select system_top.bit and hit OK (e) In the Initialization file field, hit Search... (f) Select ps7_init.tcl and hit OK (g) Enable checkboxes Run ps7_init and Run ps7_post_config 5. Application tab: (a) In the Project Name field click browse and select an application (e.g. MemoryTest) (b) In the Application field click search and select an .elf file (c) Hit Apply
2*	**Optional** - for the lwIP example application, the following extra steps are required: 1. Select the Application tab (see Figure 7) 2. In the Data Files to download before launch section: (a) Click Add (b) Select the memory file system image from the lwIP example: `<base_dir>\software\lwip_example\image.mfs` (c) In Address field type 0x8000000 and hit Open (d) Hit Apply
3	Start the application by clicking the Run button. In some test setup cases it was observed that the SDK tool was not able to start a second run session without a hardware reset. If required, power off and on the base board and restart the run configuration.

Table 9: Running an Application Step-By-Step Guide



Run Configurations

Create, manage, and run configurations

Name: MemoryTest Debug

Debug Type: Standalone Application Debug

Connection: Local

Device: Auto Detect

Hardware platform: hw_platform_0

Processor: ps7_cortexa9_0

Bitstream file: system_top.bit

Initialization file: ps7_init.tcl

Reset Processor

☐ Program FPGA

☑ Run ps7_init

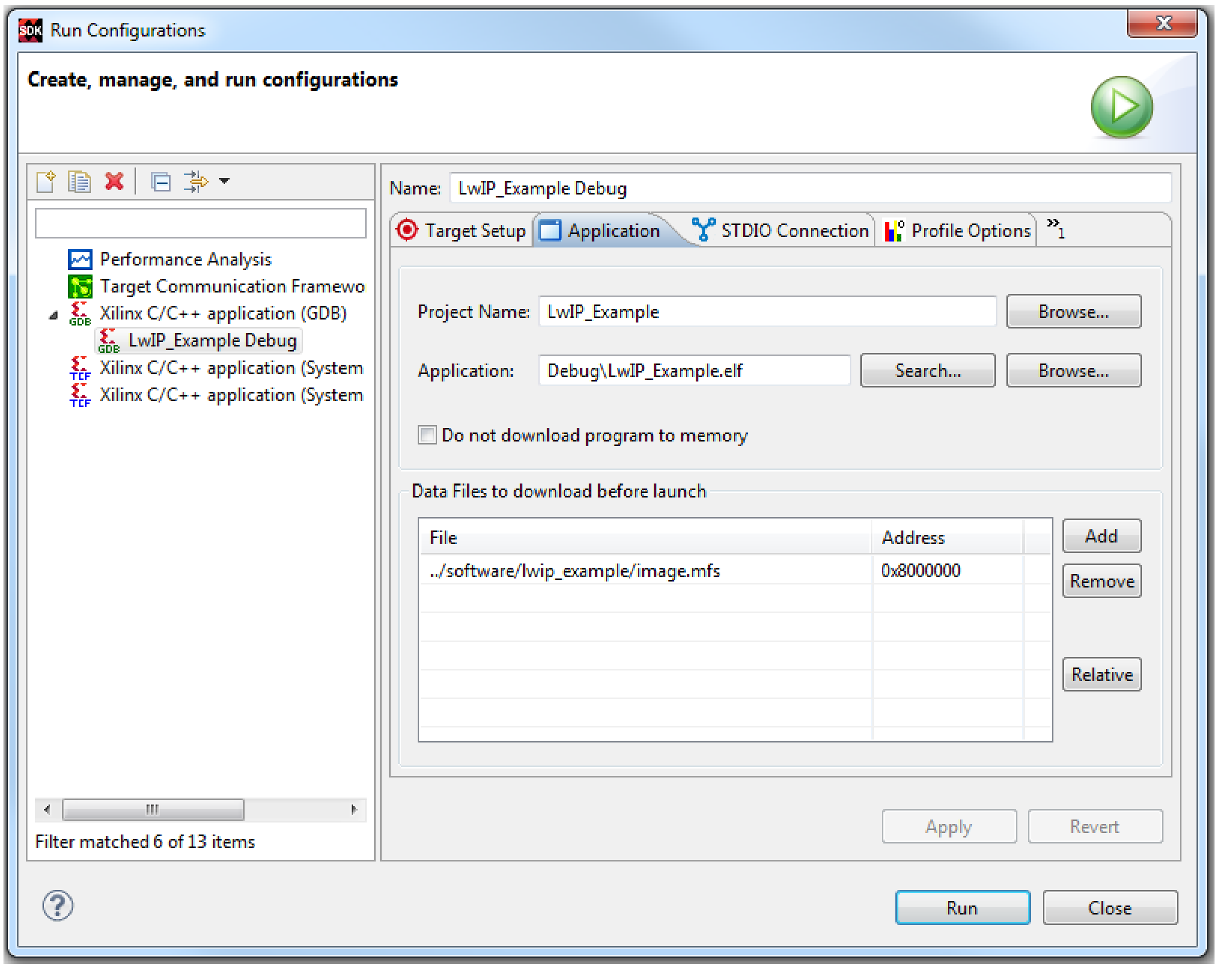
☑ Run ps7_post_config

☐ Enable Cross-Triggering

Summary of operations to be performed

```
Following operations will be performed before launching the debugger.
1. Reset processor.
2. Run ps7_init. (Only first time after System reset or board power ON)
3. Run ps7_post_config. (Only first time after System reset or board power
ON)
4.
'C:\Work\Mercury_ZX1_Reference_Design\workspace\MemoryTest\Debug\M
emoryTest.elf' will be downloaded to the processor 'ps7_cortexa9_0'
```

Figure 6: Run Configurations Settings


Run Configurations

Create, manage, and run configurations

- Performance Analysis
- Target Communication Framewo
- Xilinx C/C++ application (GDB)
  - LwIP_Example Debug
- Xilinx C/C++ application (System
- Xilinx C/C++ application (System

Filter matched 6 of 13 items

Name: LwIP_Example Debug

Tabs: Target Setup | Application | STDIO Connection | Profile Options

Project Name: LwIP_Example

Application: Debug\LwIP_Example.elf

☐ Do not download program to memory

Data Files to download before launch

File	Address
../software/lwip_example/image.mfs	0x8000000

Figure 7: Run Configurations Settings for LwIP Application

3.7 Embedded Software

This section describes the software examples and the expected UART output while running these applications.

3.7.1 General

The Mercury ZX1 SoC module reference design comes with a number of example applications, which show how to initialize the peripheral controllers and how to access the external devices. All of them are bare-metal applications that are executed in the DDR SDRAM memory.

The applications included are a Hello World example, an I2C test that reads and prints out the module and board configuration, a flash test which checks the QSPI flash memory available on the Mercury ZX1 SoC module, a web server implementation example using lwIP networking stack, a USB storage emulation example, and a memory test that checks the DDR SDRAM memory available on the module.

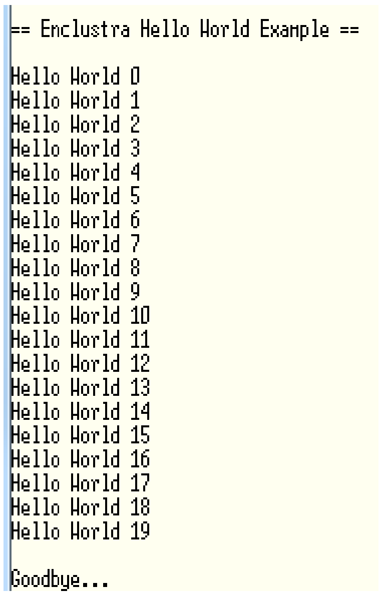
An additional section is included to describe how to create and modify the First Stage Boot Loader (FSBL) application. This application is required during the process of creating a boot image for the module.

The procedure of importing, creating and building the example applications is explained in Sections 3.4, 3.5 and 3.6.

3.7.2 Hello World Application

The Hello World application is a very simple application, which is used to demonstrate all the required steps for getting a bare-metal application running on the Mercury ZX1 SoC module.

The Hello World application prints „Hello World x“ for twenty times, while x is incremented by one at every iteration. Figure 8 shows the UART output of the Hello World application.



```
== Enclustra Hello World Example ==

Hello World 0
Hello World 1
Hello World 2
Hello World 3
Hello World 4
Hello World 5
Hello World 6
Hello World 7
Hello World 8
Hello World 9
Hello World 10
Hello World 11
Hello World 12
Hello World 13
Hello World 14
Hello World 15
Hello World 16
Hello World 17
Hello World 18
Hello World 19

Goodbye...
```

Figure 8: Hello World Application UART Output

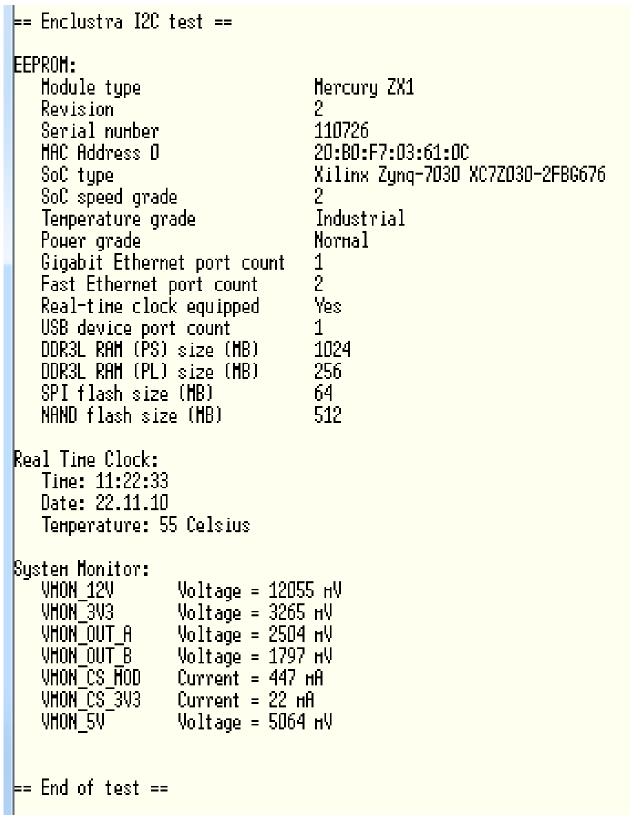
3.7.3 I2C Example Application

The I2C example application demonstrates the configuration and use of the I2C controller on the Mercury ZX1 SoC module.

The I2C example application reads the module configuration data from the secure EEPROM on the Mercury ZX1 SoC module, date and time values from the RTC on the Mercury ZX1 SoC module, as well as voltage and current information from the system monitor present on the Mercury+ PE1 base board.

The module configuration includes: module type, serial number, DDR SDRAM and flash memory sizes, and first MAC address. The MAC addresses can be used by the user to configure the Ethernet MAC.

Figure 9 shows the UART output of the I2C application.



```
== Enclustra I2C test ==

EEPROM:
   Module type                 Mercury ZX1
   Revision                    2
   Serial number               110726
   MAC Address 0               20:B0:F7:03:61:0C
   SoC type                    Xilinx Zynq-7030 XC7Z030-2FBG676
   SoC speed grade             2
   Temperature grade           Industrial
   Power grade                 Normal
   Gigabit Ethernet port count 1
   Fast Ethernet port count    2
   Real-time clock equipped    Yes
   USB device port count       1
   DDR3L RAM (PS) size (MB)    1024
   DDR3L RAM (PL) size (MB)    256
   SPI flash size (MB)         64
   NAND flash size (MB)        512

Real Time Clock:
   Time: 11:22:33
   Date: 22.11.10
   Temperature: 55 Celsius

System Monitor:
   VMON_12V       Voltage = 12055 mV
   VMON_3V3       Voltage = 3265 mV
   VMON_OUT_A     Voltage = 2504 mV
   VMON_OUT_B     Voltage = 1797 mV
   VMON_CS_MOD    Current = 447 mA
   VMON_CS_3V3    Current = 22 mA
   VMON_5V        Voltage = 5064 mV


== End of test ==
```

Figure 9: I2C Example Application UART Output

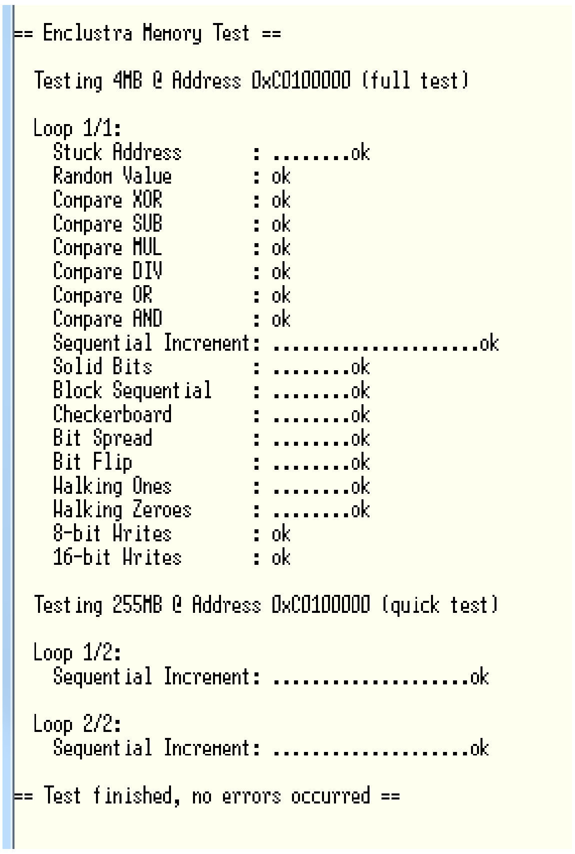
3.7.4 Memory Test Application

The memory test application performs several tests on the DDR memory present on the Mercury ZX1 SoC module. A quick simple test and a detailed full test are executed.

The simple test is run on a bigger part of the memory and checks that the sequential incrementing of the address and writing values to the memory works as expected. The full test uses several writing patterns on two different parts of the memory, then reads the values and compares the results. This test is performed on a smaller part of the memory.

Figure 10 shows the UART output of the memory test application. Note that the starting address and the memory size configured for the test depend on the module and on the application settings.

Please note that depending on the DDR memory controller speed, memory cache enable and on the test size, the memory test may take several minutes to complete.



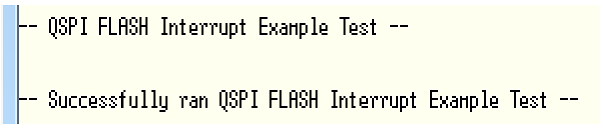
```
== Enclustra Memory Test ==

  Testing 4MB @ Address 0xC0100000 (full test)

  Loop 1/1:
    Stuck Address      : ........ok
    Random Value       : ok
    Compare XOR        : ok
    Compare SUB        : ok
    Compare MUL        : ok
    Compare DIV        : ok
    Compare OR         : ok
    Compare AND        : ok
    Sequential Increment: ....................ok
    Solid Bits         : ........ok
    Block Sequential   : ........ok
    Checkerboard       : ........ok
    Bit Spread         : ........ok
    Bit Flip           : ........ok
    Walking Ones       : ........ok
    Walking Zeroes     : ........ok
    8-bit Writes       : ok
    16-bit Writes      : ok

  Testing 255MB @ Address 0xC0100000 (quick test)

  Loop 1/2:
    Sequential Increment: ....................ok

  Loop 2/2:
    Sequential Increment: ....................ok

== Test finished, no errors occurred ==
```

Figure 10: Memory Test Application - UART Output Example (addresses and test sizes may vary)

3.7.5 Flash Test Application

The flash test application is based on the Xilinx SPI controller driver example and performs a write/read-back test of one sector starting at a specific address of the QSPI flash memory. The application initializes the SPI and the interrupt controller and includes read, write and erase functions that show how to access the flash memory on the Mercury ZX1 SoC module using the Xilinx SPI controller.

Figure 11 shows the UART output of the flash test application.



```
-- QSPI FLASH Interrupt Example Test --


-- Successfully ran QSPI FLASH Interrupt Example Test --
```

Figure 11: Flash Test Application UART Output

3.7.6 Ethernet LwIP Application

The Ethernet lwIP application represents a basic example of running a web server on the Mercury ZX1 SoC module using the open-source TCP/IP networking stack lightweight IP (lwIP).
The example is based on the Xilinx lwIP example (see Xilinx Application Note [4]), modified for compatibility with the Ethernet PHY used on the Mercury ZX1 SoC module.

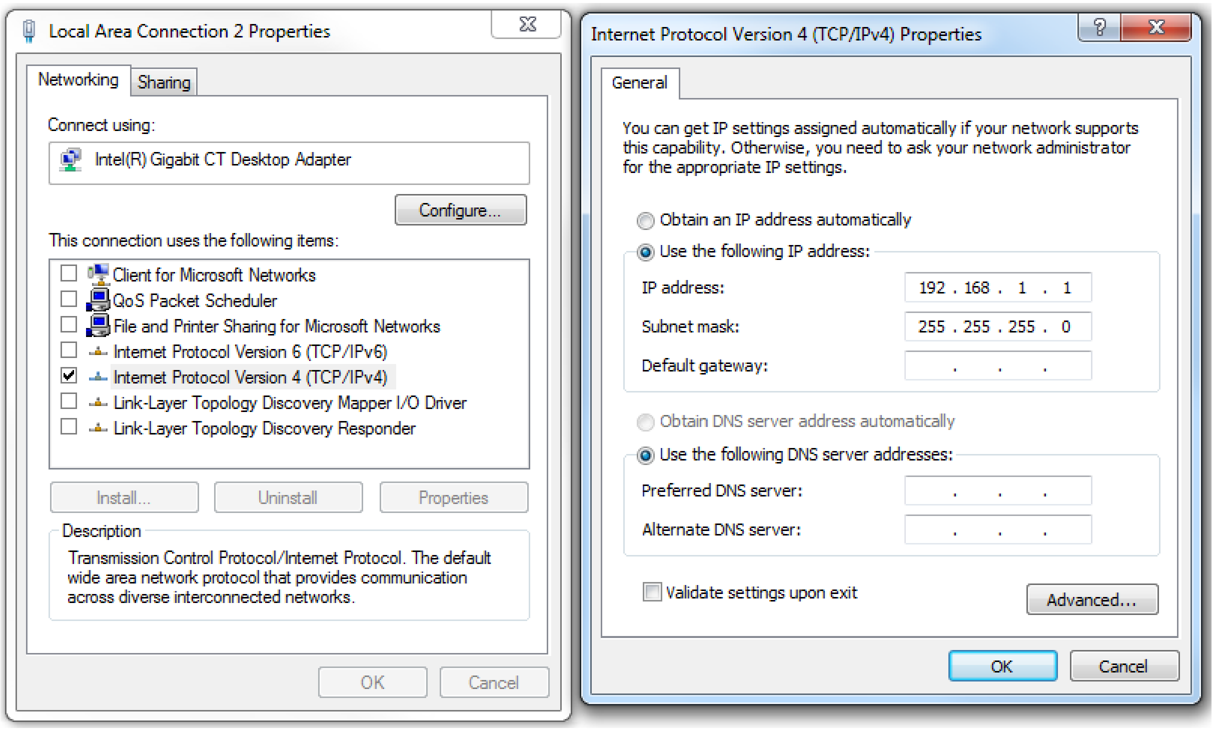
In order to do this change, one of the files generated automatically in the board support package (xemacpsif_physpeed.c) has been modified and copied to the project sources.

Before launching the Ethernet lwIP application, the network adapter of the computer needs to be configured as described below.
It is recommended to have a second network adapter for local connections and keep the current network setup on the default adapter, in order to avoid reconfiguring the connection parameters.

Computer Setup

Step	Description
1	Connect a network cable (RJ45) between your computer and the Mercury+ PE1 base board. Use the connector marked with label **ETH** in Figure 2.
2	Open Control Panel → Network and Internet → Network and Sharing Center
3	Click on the Local Area Connection → Properties (see Figure 12) 1. Disable all connections except Internet Protocol Version 4 (TCP/IPv4) 2. Click on Internet Protocol Version 4 (TCP/IPv4) → Properties (a) Set the IP address to 192.168.1.1 (b) Set the subnet mask to 255.255.255.0 (c) Hit OK

Table 10: Computer Network Setup Step-By-Step Guide



Local Area Connection 2 Properties

Networking | Sharing

Connect using:

Intel(R) Gigabit CT Desktop Adapter

This connection uses the following items:

- [ ] Client for Microsoft Networks
- [ ] QoS Packet Scheduler
- [ ] File and Printer Sharing for Microsoft Networks
- [ ] Internet Protocol Version 6 (TCP/IPv6)
- [✓] Internet Protocol Version 4 (TCP/IPv4)
- [ ] Link-Layer Topology Discovery Mapper I/O Driver
- [ ] Link-Layer Topology Discovery Responder

Description

Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

Internet Protocol Version 4 (TCP/IPv4) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address:	192 . 168 . 1 . 1
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	. . .

Obtain DNS server address automatically [✓ Use the following DNS server addresses:]

Preferred DNS server:	. . .
Alternate DNS server:	. . .

- [ ] Validate settings upon exit

Figure 12: Network Adapter Configuration for LwIP Application

Please note that a memory file system (MFS) image must be downloaded to the RAM for the application to run. Section 3.6 describes the required steps to run the lwIP Ethernet application.

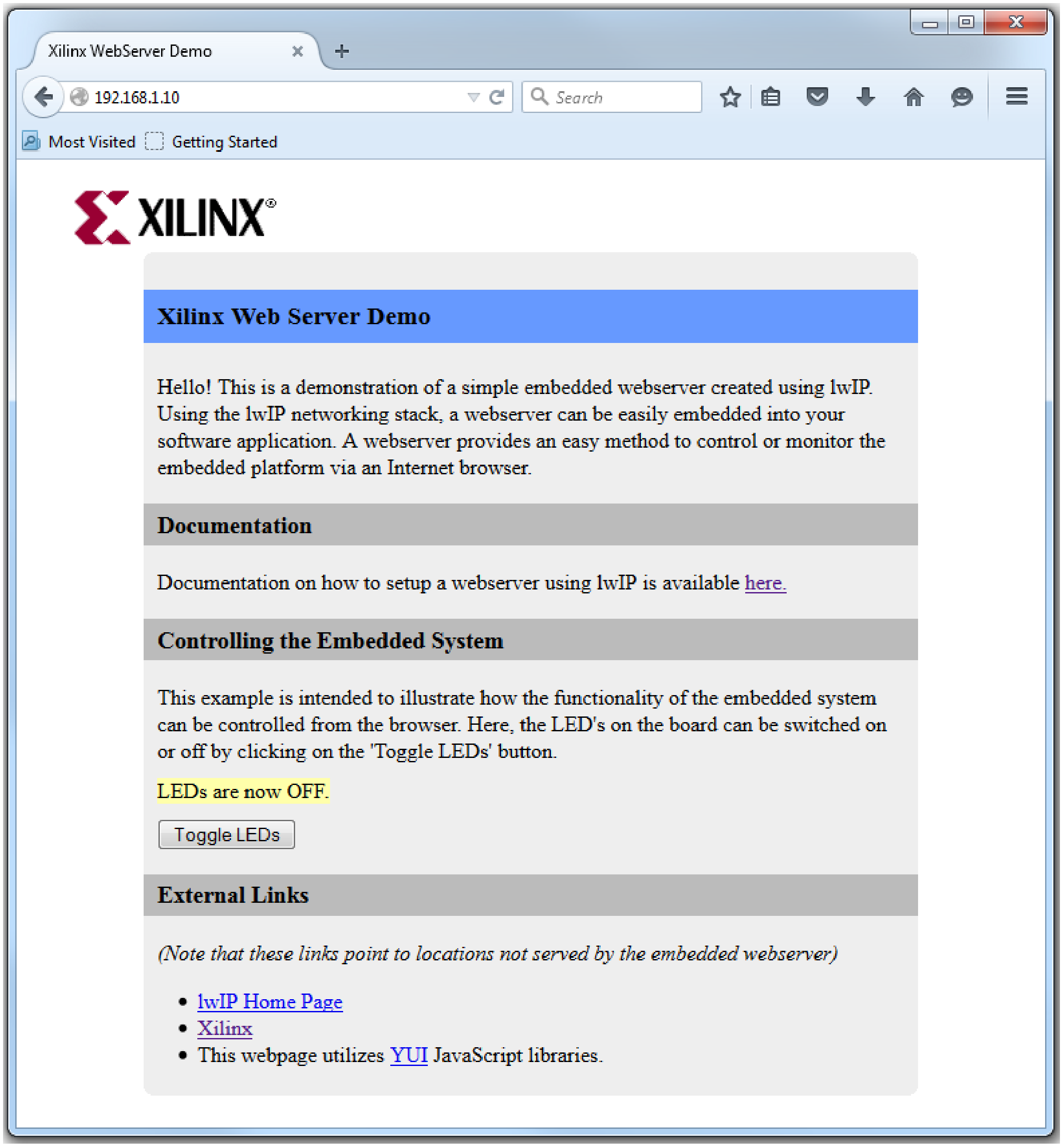
After having the computer setup, as well as the run configurations done, two tests can be performed using the lwIP application.

The testing procedure and the expected UART output are further presented.

Testing the LwIP Web Server

Step	Description
1	Open a web browser
2	Type http://192.168.1.10/ in the address bar A basic web page indicating the current state of the LEDs on the Mercury ZX1 SoC module will be loaded in the browser. The user can toggle the LEDs using a button in the web page. Figure 13 illustrates the lwIP application web page.
3	Click the Toggle LEDs button, in order to change the state of the LEDs.

Table 11: Testing the LwIP Web Server Step-By-Step Guide



Xilinx Web Server Demo

Hello! This is a demonstration of a simple embedded webserver created using lwIP. Using the lwIP networking stack, a webserver can be easily embedded into your software application. A webserver provides an easy method to control or monitor the embedded platform via an Internet browser.

Documentation

Documentation on how to setup a webserver using lwIP is available here.

Controlling the Embedded System

This example is intended to illustrate how the functionality of the embedded system can be controlled from the browser. Here, the LED's on the board can be switched on or off by clicking on the 'Toggle LEDs' button.

LEDs are now OFF.

External Links

(Note that these links point to locations not served by the embedded webserver)

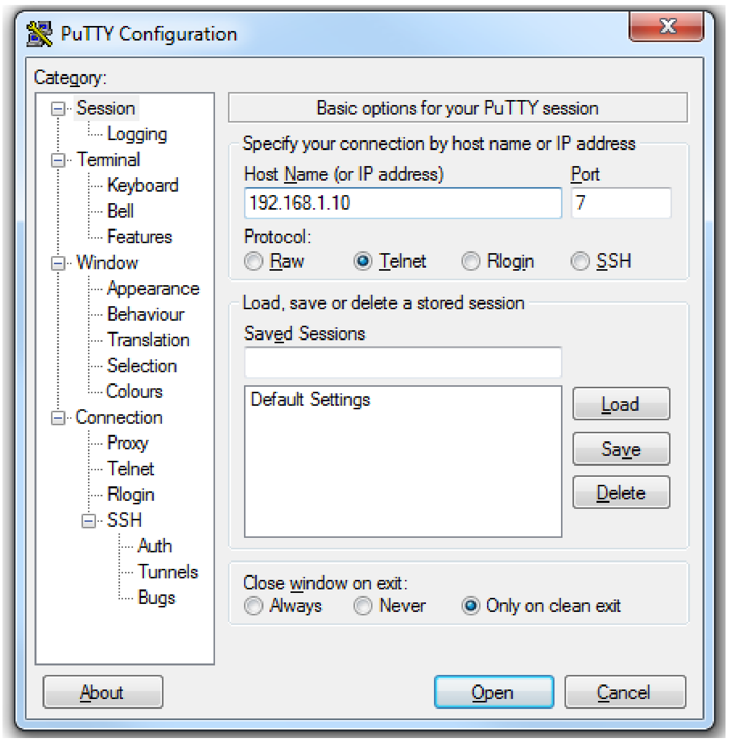
- lwIP Home Page
- Xilinx
- This webpage utilizes YUI JavaScript libraries.

Figure 13: LwIP Application Web Page

Testing the LwIP Telnet Echo Server

Step	Description
1	Open a Telnet program (e.g. PuTTY) and configure the following parameters (see Figure 14): 1. For Host Name type 192.168.1.10 2. For Protocol select Telnet 3. For Port type 7 4. Click Open
2	Each character sent to the Mercury ZX1 SoC module is immediately sent back (each character appears twice in the Telnet terminal).

Table 12: Testing the LwIP Telnet Echo Server Step-By-Step Guide


PuTTY Configuration

Category:

Basic options for your PuTTY session

Specify your connection by host name or IP address

Host Name (or IP address): 192.168.1.10

Port: 7

Protocol: Raw [✓ Telnet] Rlogin SSH

Load, save or delete a stored session

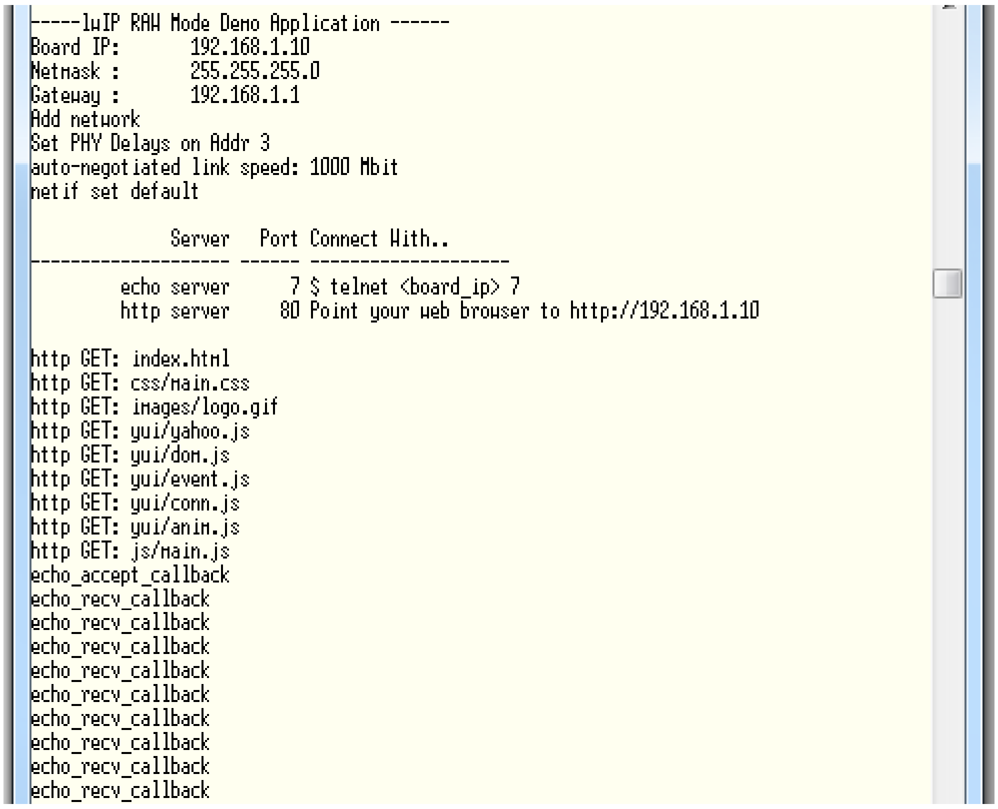
Saved Sessions

Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

Figure 14: PuTTY Configuration for Telnet Communication

Figure 15 displays the UART output of the Ethernet lwIP application, corresponding to web page loading action and to Telnet echo calls.



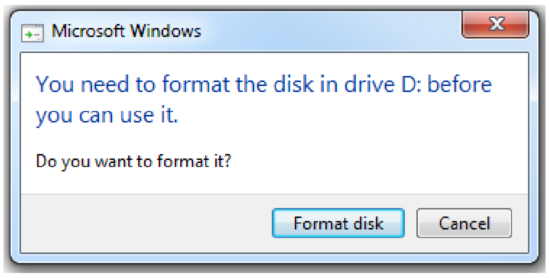
```
-----lwIP RAW Mode Demo Application ------
Board IP:       192.168.1.10
Netmask :       255.255.255.0
Gateway :       192.168.1.1
Add network
Set PHY Delays on Addr 3
auto-negotiated link speed: 1000 Mbit
netif set default

              Server   Port Connect With..
-------------------- ------ --------------------
         echo server      7 $ telnet <board_ip> 7
         http server     80 Point your web browser to http://192.168.1.10

http GET: index.html
http GET: css/main.css
http GET: images/logo.gif
http GET: yui/yahoo.js
http GET: yui/dom.js
http GET: yui/event.js
http GET: yui/conn.js
http GET: yui/anim.js
http GET: js/main.js
echo_accept_callback
echo_recv_callback
echo_recv_callback
echo_recv_callback
echo_recv_callback
echo_recv_callback
echo_recv_callback
echo_recv_callback
echo_recv_callback
echo_recv_callback
```

Figure 15: Ethernet LwIP Application UART Output

3.7.7 USB Storage Application

The USB storage application represents a basic example of a USB storage emulation.
The example is based on the Xilinx USB example (xusbps_intr_example.c), including additional printing messages.

In order to run the USB storage emulation example, two USB cables are required: one for the UART output and one for the USB storage emulation. While running the application, the USB cable for the USB storage needs to be connected to the base board. After starting the example the cable, Windows will issue a message asking the user to format the disk (see Figure 16).



Microsoft Windows

You need to format the disk in drive D: before you can use it.

Do you want to format it?

Figure 16: Windows Message after Connecting a USB Cable

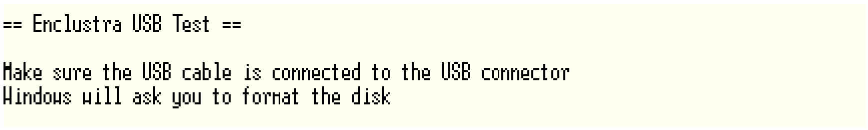
After formatting the disk (capacity 1 MB, FAT), a USB storage is visible and usable as a standard removable disk in Windows.

In order to run the USB example, the following steps are required:

Step	Description
1	Set the power switch of the Mercury+ PE1 base board to OFF/PCIe (see label **PWR SW** in Figure 2).
2	Disconnect all USB cables from the Mercury+ PE1 base board.
3	Set the configuration DIP switches on the Mercury+ PE1 base board as follows (see labels **CFG A** and **CFG B** in Figure 2): • CFG A = [1: OFF, 2: OFF, 3: OFF, 4: ON] • CFG B = [1: OFF, 2: OFF, 3: ON, 4: ON]
4	For UART output, connect a USB cable between your computer and the Mercury+ PE1 base board. Use the micro USB port labeled **USBUB** in Figure 2.
5	For the USB storage, connect a USB cable between your computer and the Mercury+ PE1 base board. Use the USB type B port labeled **USB** in Figure 2.
6	Set the power switch of the Mercury+ PE1 base board to ON (see label **PWR SW** in Figure 2).
7	Program the FPGA as described in Section 3.5 and start the application as described in Section 3.6
8	Format the disk and use it as standard removable disk storage.

Table 13: Running the USB Storage Example Step-By-Step Guide

Figure 17 shows the UART output of the USB storage application.



```
== Enclustra USB Test ==

Make sure the USB cable is connected to the USB connector
Windows will ask you to format the disk
```

Figure 17: USB Storage Application UART Output

3.7.8 First Stage Boot Loader (FSBL) Application

The First Stage Boot Loader application is used in the boot image creation process described in Section 4.1.1. It is not used as an independent application.

The FSBL application is based on the Xilinx FSBL, extended by functions that enable the RGMII delays in the Ethernet PHY and initialize the real-time clock (RTC).

In addition, the FSBL reads the Ethernet MAC address via I2C from the secure EEPROM and configures the Ethernet MAC accordingly. If the secure EEPROM cannot be accessed, a default MAC address is assigned to the MAC.

The changes mentioned above are done in fsbl_hooks.c file.

The source files provided in the ZIP archive along with this document must be copied into the software workspace, after generating the FSBL using Xilinx tools. Section 3.4 describes the required steps for FSBL project generation.

4 Boot Configurations

Once a software application has been developed and tested, this can be used to build a boot image for the module.

The boot image contains the FSBL, the bitstream for programming the PL, and the software bare-metal application.

In order to use a software application for the boot image, the code must be mapped for execution from the external DDR memory. If the program is mapped to the on-chip memory, it will overwrite the boot loader during execution. The example applications in the reference design are mapped for execution from the DDR memory by default.

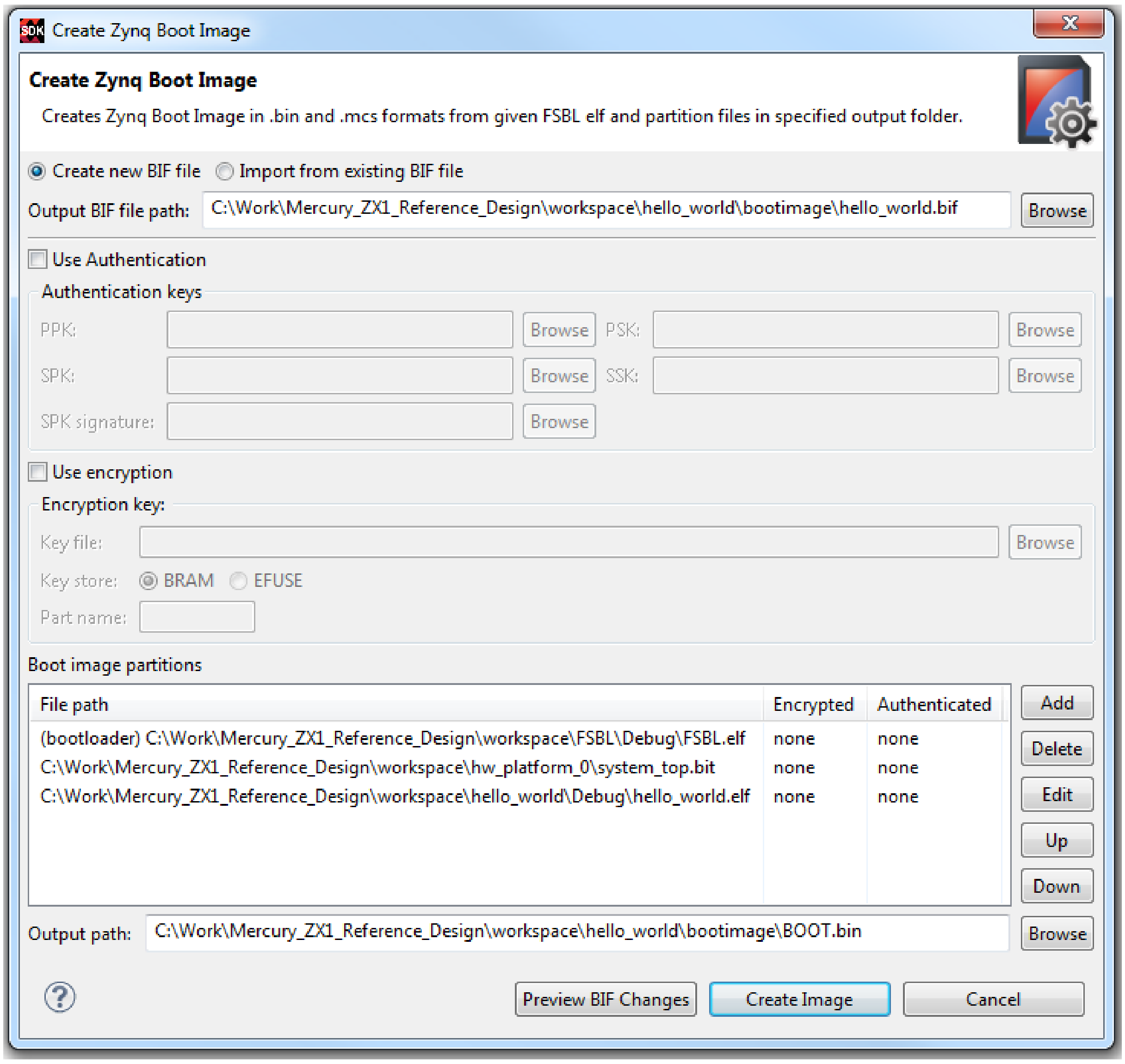
For a fast test of the boot configurations, the pre-generated .bin images included in the `<base_dir>\binaries` directory may be used for boot, instead of rebuilding the image. You need to select the file corresponding to the SoC device that is equipped on the module.

4.1 QSPI Flash Boot

4.1.1 Generating the Image Files

Step	Description
1	Create the boot image from Xilinx SDK 2017.4 (see Figure 18): 1. Right click on the application in the Project Explorer 2. Select Create Boot Image
2	For the lwIP Application, the following extra steps are required (see Figure 19): In Create Zynq Boot Image Window hit Add ... 1. In File path field type the path to the memory file system image from the lwIP example: `<base_dir>\software\lwip_example\image.mfs` 2. In Load field type 0x8000000 3. Hit OK
3	In Create Zynq Boot Image Window hit Create Image. An image will be created in `<workspace>\<app_name>\bootimage\BOOT.bin`.

Table 14: Generating the Image Files for QSPI Flash Boot Mode Step-by-Step Guide



Create Zynq Boot Image

Create Zynq Boot Image

Creates Zynq Boot Image in .bin and .mcs formats from given FSBL elf and partition files in specified output folder.

[✓ Create new BIF file] Import from existing BIF file

Output BIF file path: C:\Work\Mercury_ZX1_Reference_Design\workspace\hello_world\bootimage\hello_world.bif

☐ Use Authentication

Authentication keys

PPK: PSK: SPK: SSK: SPK signature:

☐ Use encryption

Encryption key:

Key file:

Key store: [✓ BRAM] EFUSE

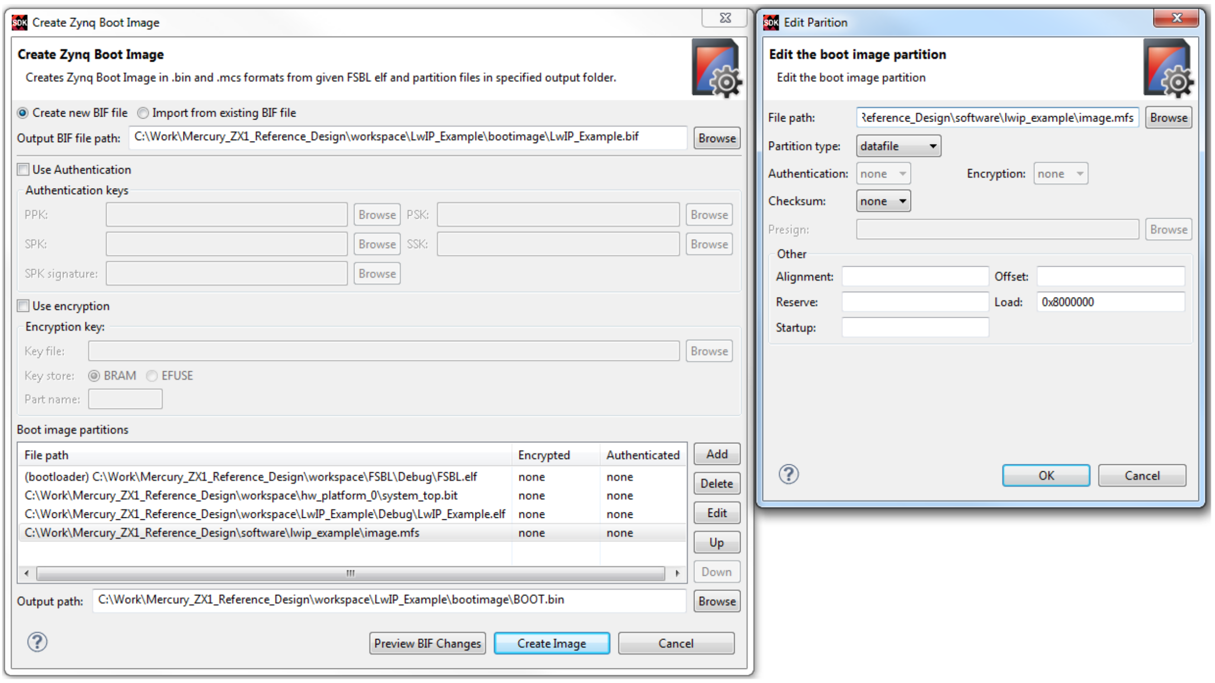
Part name:

Boot image partitions

File path	Encrypted	Authenticated
(bootloader) C:\Work\Mercury_ZX1_Reference_Design\workspace\FSBL\Debug\FSBL.elf	none	none
C:\Work\Mercury_ZX1_Reference_Design\workspace\hw_platform_0\system_top.bit	none	none
C:\Work\Mercury_ZX1_Reference_Design\workspace\hello_world\Debug\hello_world.elf	none	none

Output path: C:\Work\Mercury_ZX1_Reference_Design\workspace\hello_world\bootimage\BOOT.bin

Figure 18: Create Zynq Boot Image Settings


Create Zynq Boot Image

Create Zynq Boot Image

Creates Zynq Boot Image in .bin and .mcs formats from given FSBL elf and partition files in specified output folder.

[✓ Create new BIF file] Import from existing BIF file

Output BIF file path: C:\Work\Mercury_ZX1_Reference_Design\workspace\LwIP_Example\bootimage\LwIP_Example.bif

☐ Use Authentication

☐ Use encryption

Key store: [✓ BRAM] EFUSE

Boot image partitions

File path	Encrypted	Authenticated
(bootloader) C:\Work\Mercury_ZX1_Reference_Design\workspace\FSBL\Debug\FSBL.elf	none	none
C:\Work\Mercury_ZX1_Reference_Design\workspace\hw_platform_0\system_top.bit	none	none
C:\Work\Mercury_ZX1_Reference_Design\workspace\LwIP_Example\Debug\LwIP_Example.elf	none	none
C:\Work\Mercury_ZX1_Reference_Design\software\lwip_example\image.mfs	none	none

Output path: C:\Work\Mercury_ZX1_Reference_Design\workspace\LwIP_Example\bootimage\BOOT.bin

Edit Parition

Edit the boot image partition

Edit the boot image partition

File path:	Reference_Design\software\lwip_example\image.mfs
Partition type:	datafile
Authentication:	none
Encryption:	none
Checksum:	none
Presign:	
Alignment:	
Offset:	
Reserve:	
Load:	0x8000000
Startup:	

Figure 19: Create Zynq Boot Image Settings for LwIP Application

4.1.2 Preparing the Hardware

Step	Description
1	Set the power switch of the Mercury+ PE1 base board to OFF/PCIe (see label **PWR SW** in Figure 2).
2	Disconnect all USB cables from the Mercury+ PE1 base board.
3	Enable the QSPI flash boot mode by setting the configuration DIP switches on the Mercury+ PE1 base board as follows (see labels **CFG A** and **CFG B** in Figure 2): • CFG A = [1: ON, 2: OFF, 3: OFF, 4: ON] • Depending on the available USB cable, the CFG B DIP switches must be configured as follows: • For micro USB cable: 1. Set CFG B = [1: OFF, 2: OFF, 3: ON, 4: ON] 2. Connect a USB cable to the micro USB port on the Mercury+ PE1 base board (see label **USBUB** in Figure 2) • For type B USB cable: 1. Set CFG B = [1: OFF, 2: OFF, 3: ON, 4: ON] 2. Connect a USB cable to the USB type B port on the Mercury+ PE1 base board (see label **USB** in Figure 2) 3. Set CFG B = [1: ON, 2: ON, 3: ON, 4: ON]
4	Set the power switch of the Mercury+ PE1 base board to ON (see label **PWR SW** in Figure 2).

Table 15: Preparing the Hardware for QSPI Flash Boot Mode Step-by-Step Guide

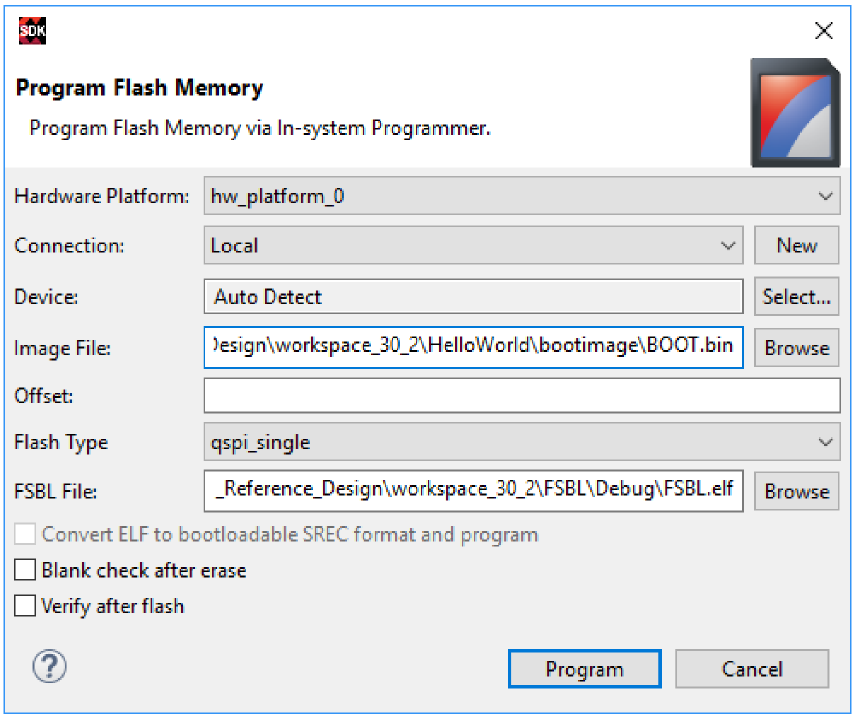
4.1.3 Programming the QSPI Flash

Step	Description
1	Open Xilinx SDK 2017.4: 1. Xilinx Tools → Program Flash 2. In Program Flash Memory window (see Figure 20): (a) For Image File select the boot image generated as described in Section 4.1.1 (b) For FSBL File select the FSBL binary generated as described in Section 3.4 (c) For Flash Type select qspi_single (d) Hit Program and wait for completion The settings in the pictures are for reference only. Note that the configuration file must be selected according to your application. Some Vivado tool versions support QSPI flash programming only in JTAG boot mode. Please check the Mercury ZX1 SoC module and the Mercury+ PE1 base board user manuals for details on how to configure this boot mode. If this is not available, please use one of the alternative methods.
2*	**Optional** - if SDK returns errors during flash programming or if the system does not boot properly, another option is to use Vivado 2017.4 to program the QSPI flash. 1. Flow → Open Hardware Manager 2. Click on Open target → Auto Connect 3. Right click on the corresponding FPGA device in the left bar → Add Configuration Memory Device (see Figure 21) (a) For Select Configuration Memory Part choose the memory part according to the Mercury ZX1 SoC Module User Manual [5], part type single. This is in most cases s25fl512s-qspi-x4-single. (b) Hit OK 4. In Program Configuration Memory Device window (see Figure 22): (a) For Configuration file select the boot image generated as described in Section 4.1.1 (b) For Zynq FSBL select the FSBL binary generated as described in Section 3.4 (c) In Program Operations section: • For Address Range select Entire Configuration Memory Device • Enable checkboxes Erase, Blank Check, Program and Verify • Hit OK and wait for completion The settings in the pictures are for reference only. Note that the memory part and the configuration file must be selected according to your application. Some Vivado tool versions support QSPI flash programming only in JTAG boot mode. Please check the Mercury ZX1 SoC module and the Mercury+ PE1 base board user manuals for details on how to configure this boot mode. If this is not available, please use one of the alternative methods.
3*	**Optional** - alternatively, Enclustra Module Configuration Tool (MCT) [7] can be used to program the QSPI flash. Close all other tools that may be connected to the FTDI device (Vivado Hardware Manager, SDK, UART terminal).

Continued on next page...

Step	Description
	Before programming the QSPI flash from MCT, make sure the hardware configuration on the Mercury+ PE1 base board is done according to Section 4.1.2. In order to see the output in the UART terminal, close the MCT tool after QSPI flash programming and disconnect and reconnect the USB cable from the board and the UART terminal.

Table 16: Programming the QSPI Flash for QSPI Flash Boot Mode Step-by-Step Guide


Program Flash Memory

Program Flash Memory via In-system Programmer.

Hardware Platform: hw_platform_0

Connection: Local

Device: Auto Detect

Image File:)esign\workspace_30_2\HelloWorld\bootimage\BOOT.bin

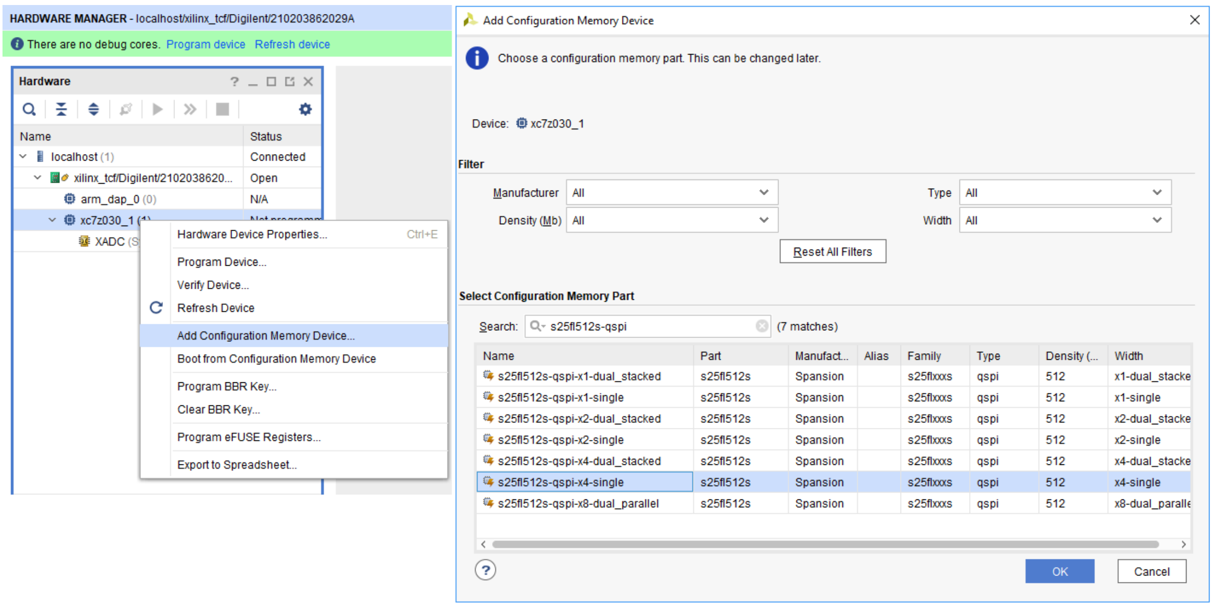
Offset:

Flash Type qspi_single

FSBL File: _Reference_Design\workspace_30_2\FSBL\Debug\FSBL.elf

- [ ] Convert ELF to bootloadable SREC format and program
- [ ] Blank check after erase
- [ ] Verify after flash

Figure 20: QSPI Flash Programming Settings in SDK


HARDWARE MANAGER - localhost/xilinx_tcf/Digilent/210203862029A

There are no debug cores. Program device Refresh device

Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/2102038620...	Open
arm_dap_0 (0)	N/A
xc7z030_1 (1)	
XADC (S	

Hardware
Hardware Device Properties... Ctrl+E
Program Device...
Verify Device...
Refresh Device
Add Configuration Memory Device...
Boot from Configuration Memory Device
Program BBR Key...
Clear BBR Key...
Program eFUSE Registers...
Export to Spreadsheet...

Add Configuration Memory Device

Choose a configuration memory part. This can be changed later.

Device: xc7z030_1

Filter

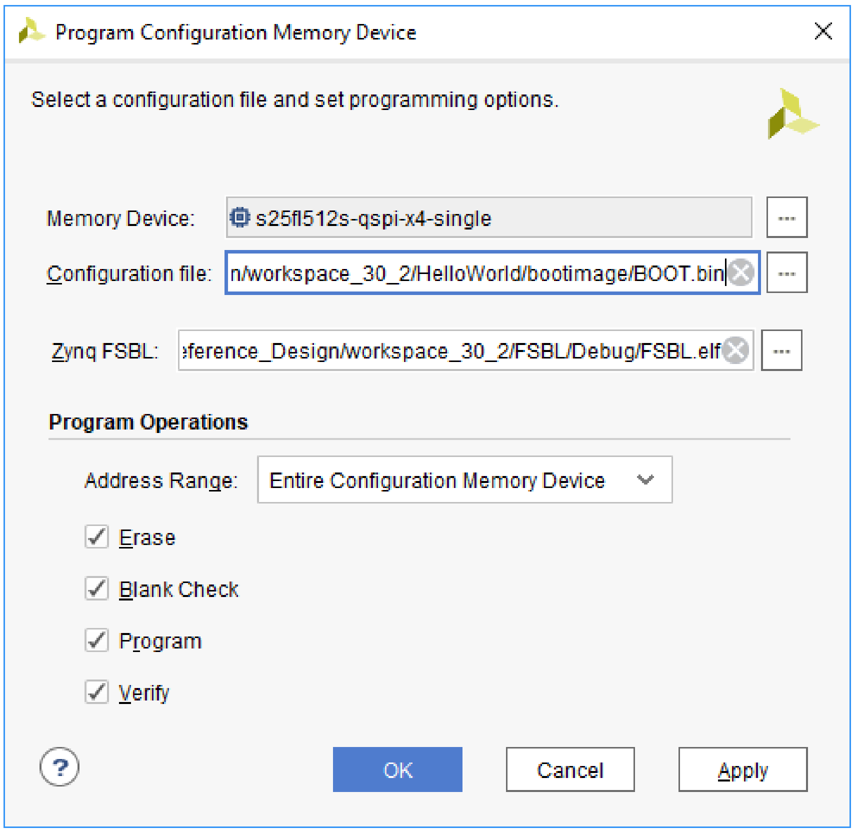
Manufacturer: All
Type: All
Density (Mb): All
Width: All

Select Configuration Memory Part

Search: s25fl512s-qspi (7 matches)

Name	Part	Manufact...	Alias	Family	Type	Density (...	Width
s25fl512s-qspi-x1-dual_stacked	s25fl512s	Spansion		s25flxxxs	qspi	512	x1-dual_stacke
s25fl512s-qspi-x1-single	s25fl512s	Spansion		s25flxxxs	qspi	512	x1-single
s25fl512s-qspi-x2-dual_stacked	s25fl512s	Spansion		s25flxxxs	qspi	512	x2-dual_stacke
s25fl512s-qspi-x2-single	s25fl512s	Spansion		s25flxxxs	qspi	512	x2-single
s25fl512s-qspi-x4-dual_stacked	s25fl512s	Spansion		s25flxxxs	qspi	512	x4-dual_stacke
s25fl512s-qspi-x4-single	s25fl512s	Spansion		s25flxxxs	qspi	512	x4-single
s25fl512s-qspi-x8-dual_parallel	s25fl512s	Spansion		s25flxxxs	qspi	512	x8-dual_paralle

Figure 21: QSPI Flash Programming Settings in Vivado - Adding the Memory Device


Program Configuration Memory Device

Select a configuration file and set programming options.

Memory Device: s25fl512s-qspi-x4-single

Configuration file: n/workspace_30_2/HelloWorld/bootimage/BOOT.bin

Zynq FSBL: :ference_Design/workspace_30_2/FSBL/Debug/FSBL.elf

Program Operations

Address Range: Entire Configuration Memory Device

✓ Erase

✓ Blank Check

✓ Program

✓ Verify

Figure 22: QSPI Flash Programming Settings in Vivado

4.1.4 Booting from the QSPI Flash

Step	Description
1	Check that the hardware configuration is done according to Section 4.1.2.
2	Press the power-on reset button (see label **POR** in Figure 2) and release it after a second.

Table 17: Booting from the QSPI Flash Step-by-Step Guide

4.2 SD Card Boot

4.2.1 Generating the Image Files

Please refer to Section 4.1.1 describing the steps required to generate a boot image.

4.2.2 Preparing the Hardware

Step	Description
1	Set the power switch of the Mercury+ PE1 base board to OFF/PCIe (see label **PWR SW** in Figure 2).
2	Disconnect all USB cables from the Mercury+ PE1 base board.
3	Enable the SD card boot mode (default) by setting the configuration DIP switches on the Mercury+ PE1 base board as follows (see labels **CFG A** and **CFG B** in Figure 2): • CFG A = [1: OFF, 2: OFF, 3: OFF, 4: ON] • Depending on the available USB cable, the CFG B DIP switches must be configured as follows: • For micro USB cable: 1. Set CFG B = [1: OFF, 2: OFF, 3: ON, 4: ON] 2. Connect a USB cable to the micro USB port on the Mercury+ PE1 base board (see label **USBUB** in Figure 2) • For type B USB cable: 1. Set CFG B = [1: OFF, 2: OFF, 3: ON, 4: ON] 2. Connect a USB cable to the USB type B port on the Mercury+ PE1 base board (see label **USB** in Figure 2) 3. Set CFG B = [1: ON, 2: ON, 3: ON, 4: ON]

Table 18: Preparing the Hardware for SD Card Boot Mode Step-by-Step Guide

4.2.3 Programming the SD Card

Step	Description
1	Write the Xilinx SD card boot image to the SD card 1. Insert the SD card into the SD card slot of your computer 2. Copy the boot image generated for your application from `<workspace>\<app_name>\bootimage\BOOT.bin` to your SD card (directly in the root directory). Note that the name of the image must be preserved.

Table 19: Programming the SD Card for SD Card Boot Mode Step-by-Step Guide

4.2.4 Booting from the SD Card

Step	Description
1	Insert the SD card into the SD card slot of the Mercury+ PE1 base board (see label **SD Card** in Figure 2).
2	Set the power switch of the Mercury+ PE1 base board to ON (see label **PWR SW** in Figure 2).

Table 20: Booting from the SD Card Step-by-Step Guide

5 Troubleshooting

5.1 Vivado Issues

- If the changes in the block design (including licenses for special IPs) are not propagated into implementation, open the Hierarchy tab in Vivado and regenerate the block design files:
  1. Right click on the block design file (.bd)
  2. Click on Reset Output Products → Reset
  3. Click on Generate Output Products → Generate → OK

5.2 SDK Runtime Exceptions

- In order to avoid runtime exceptions issued by SDK, always stop running processes with the red Terminate button. In debug mode, hit this button before resetting or disconnecting the Mercury ZX1 SoC module.
- If the SDK reports runtime exceptions while downloading a program to memory, the following steps should be followed:
  1. Close SDK
  2. Shutdown any javaw, eclipse or xmd processes in Windows Task Manager
  3. Power off the Mercury ZX1 SoC module
  4. Restart SDK and power on the Mercury ZX1 SoC module
- After using SD card or flash boot on Zynq, the debugger will not connect to the target device anymore. In order to re-enable the debugger connection, the following steps should be followed:
  1. Remove the SD card and select the SD card boot mode, or alternatively, delete the QSPI flash and select the flash boot mode
  2. Restart the Mercury ZX1 SoC module

5.3 JTAG Connection Issues

- If the JTAG cable is not detected, the following steps should be followed:
  1. Make sure that the hardware configuration is made according to Section 3.2
  2. Check that only one JTAG adapter is active and connected to the hardware at a given moment. Make sure that you are not using both built-in JTAG and Xilinx Platform Cable USB. More information on the Xilinx JTAG mode configuration on the Mercury+ PE1 base board can be retrieved from the base board user manual [6].
  3. If built-in JTAG is used, check that the FTDI device is configured to Xilinx JTAG mode. This can be done using the Enclustra MCT software [7].
  4. Remove the USB connection and power supply from the Mercury+ PE1 base board and close SDK
  5. Reconnect the USB and power supply and start SDK again
  6. If built-in JTAG is used, check for UART connection issues (refer to Section 5.4)
  7. Reboot the computer if the problem persists

5.4 UART Connection Issues

- If the computer is not able to recognize the USB UART on the Mercury+ PE1 base board:
  1. Check that the USB cable is connected properly
  2. Check that the FTDI VCP drivers are installed
     (a) Open Device Manager
     (b) Universal Serial Bus controllers → USB Serial Converter A/B → Properties → Advanced tab → enable Load VCP checkbox

    (c) Reboot the computer if the COM port is still not detected
  3. Reinstall the FTDI drivers if the problem persists

- If the computer does not output any character in the terminal program:
  1. Check that the FTDI device is set to UART mode:
     (a) Download and open FT_Prog utility (this is a third party tool offered by the FTDI company to configure FTDI devices)
     (b) DEVICES → Scan and Parse
     (c) Check that for Port A and B the RS232 UART property is true
  2. Check that the baud rate for the UART in the block design matches the baud rate set in the terminal program
  3. If the UART used is mapped to the EMIO pins in the PS, resetting the ARM core will not suffice. Reprogramming the PL is necessary, as the UART lines go through PL.
  4. Make sure that Enclustra MCT software is not open. After closing it, unplug and plug in again the USB cable corresponding to the UART communication.

5.5 QSPI Boot Issues

- If the Mercury ZX1 SoC module is not able to boot from the QSPI flash:
  1. Use Vivado to program the flash
     (a) Make sure that the Memory Device part type is correctly selected
     (b) Make sure Erase and Program options are enabled
     (c) Select Entire Configuration Memory Device for Address Range
  2. If the problem persists, a possible solution is to first erase the flash, and then program it either from Vivado or SDK

  Please refer to Section 4.1.3 for details on QSPI flash programming.

List of Figures

List of Tables

References

[1] Zynq-7000 All Programmable SoC Embedded Design Tutorial, UG1165, Xilinx, 2015
[2] Vivado Design Suite User Guide, Embedded Processor Hardware Design, UG898, Xilinx, 2016
[3] Vivado Design Suite Tutorial, Embedded Processor Hardware Design, UG940, Xilinx, 2016
[4] Xilinx XAPP1026, LightWeight IP Application Examples, Xilinx, 2014
[5] Mercury ZX1 SoC Module User Manual
→ Ask Enclustra for details

[6] Mercury+ PE1 Base Board User Manual
→ Ask Enclustra for details
[7] Enclustra Module Configuration Tool (MCT)
→ Ask Enclustra for details
[8] Enclustra Modules Heat Sink Application Note
→ Ask Enclustra for details