

Developing Zynq Software with Xilinx SDK Lab 7 Boot from Flash



Feb 2018
Version 10

Lab 7 Overview

With the FSBL in place, we are now ready to create a boot image and boot one of our applications from non-volatile memory. A complete boot-up requires at least three things:

1. FSBL
2. Bitstream (Optional)
3. Application

This Lab will show you how to combine these pieces together to create the boot image for QSPI. You will then be able to experiment with a true embedded, non-tethered boot.

Lab 7 Objectives

When you have completed Lab 7, you will know:

- How to create a boot image
- Write and boot from QSPI

Experiment 1: Create the QSPI Boot Image

The first step is to create the non-volatile boot image. This example will target booting from QSPI.

Experiment 1 General Instruction:

Change the configuration for the Peripheral Test to Release. Create a boot image for QSPI.

Experiment 1 Step-by-Step Instructions:

We first need to decide which application that we will bootload. The important consideration here is that you do not want to choose an application that will compete for the same memory resource as the FSBL. Recall from Lab 6 that the FSBL is ~150kB, targeted at the on-chip RAM_0 and RAM_1.

Since the Test_Memory application runs from the same memory, this won't work. We may be able to re-design the linker script to split this application across the remaining RAM0 and also RAM1, but that is beyond the scope of this experiment.

The Hello_Zynq application is a good candidate, except recall that we targeted this application to on-chip RAM.

The Test Peripheral application is targeted at DDR. Since this application is compatible immediately with the FSBL, we will use it.

1. Similar to the FSBL, we will not be debugging the Test Peripheral application during this exercise. That means we should also change the active configuration to **Release** to take advantage of higher optimization. Right-click on the Test_Peripherals application, then select **Build Configurations → Set Active Release**.

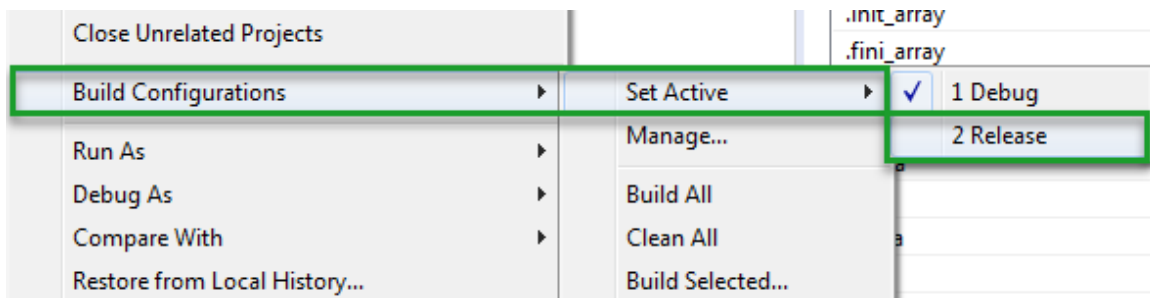


Figure 1 – Change to Release Configuration

2. In the main SDK menu, select **Project → Build All** to force the new configurations to build.
3. In SDK, select the Test_Peripherals application.

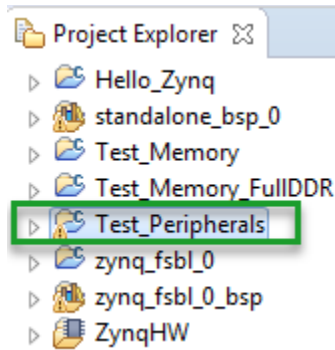


Figure 2 – Select Application to Boot

4. Select **Xilinx Tools** → **Create Boot Image**.

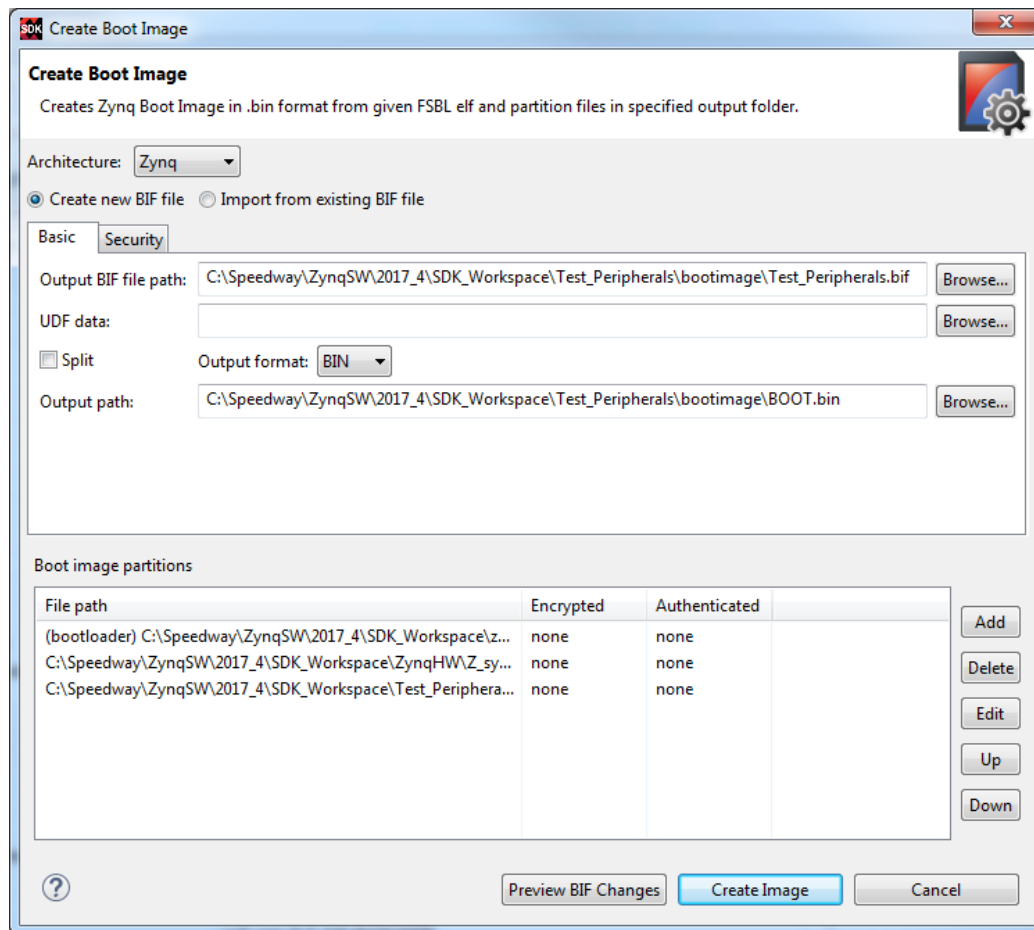


Figure 3 – Initial Create Zynq Boot Image Dialog

5. If your dialog is not pre-populated with files as shown above, then your Test_Peripherals application may have an issue. Click **Cancel**, then right-click on Test_Peripherals and select **Clean Project**.

6. In the dialog, leave the radio button selected for *Create a new BIF file*. BIF stands for Boot Image Format. The BIF is the input file into Bootgen that lists the partitions (bitstream, software) which Bootgen is to include in the image. The BIF also includes attributes for the partitions. Partition attributes allow the user to specify if the partition is to be encrypted and/or authenticated.
7. Click on the Security Tab. We will not be using Authentication or Encryption, so these checkboxes should be left unchecked. Return back to the Basic Tab

The *Boot Image Partitions* is prepopulated with the FSBL and Application ELF **Release** images as well as the bitstream. SDK 2017.4 has correctly pulled in the ELF's for our active configurations.

8. The *Output Path* area actually determines two things, although this is not obvious. Of course, it sets the output file name. The second thing that it determines is whether it creates an image for the SD Card (bin) or the QSPI (mcs). By default, it is set to .bin (SD Card), since the MiniZed by default can't boot from SD, this must be changed. Change the output format to MCS to generate a QSPI image. Change the output file name to **Test_Peripherals.mcs**. You should have the same setup and naming conventions as the image below

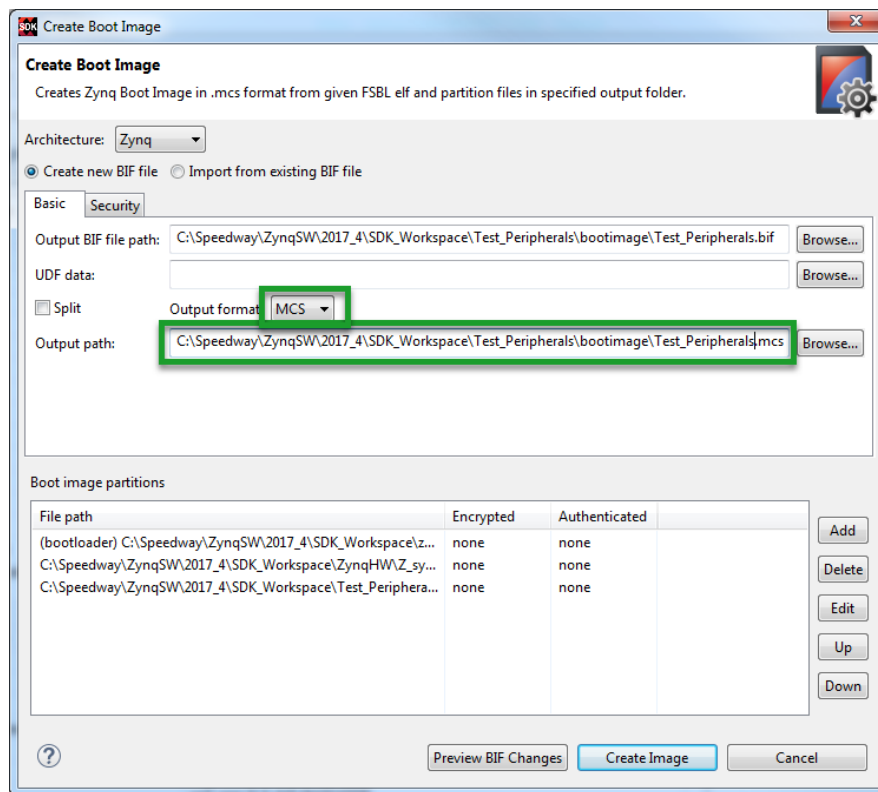


Figure 4 – Zynq Boot Image Dialog

When complete the dialog should look as above, with the FSBL first, the bitstream second, and the application third

9. Click **Create Image**.

10. Using Windows Explorer, navigate to the application directory, then into the newly created **bootimage** directory. Notice that two files have been created: .bif and .mcs.

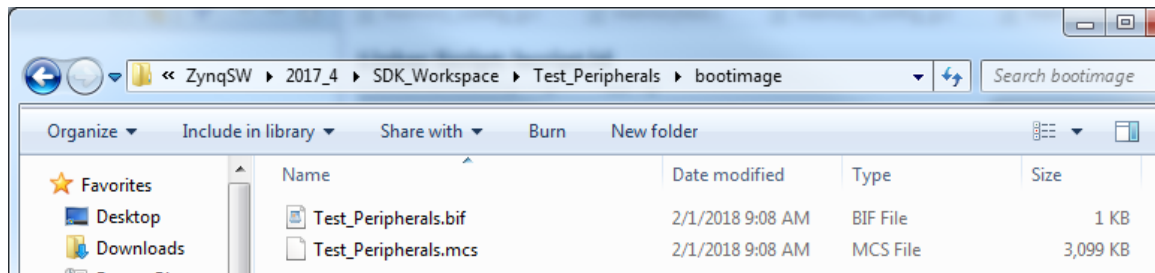


Figure 5 - bootimage Directory

11. Open the Test_Peripherals.bif file in a text editor. You'll notice that this is a very simple file. The absolute paths to the three files are listed. Close the file.

```
//arch = zynq; split = false; format = MCS
the_ROM_image:
{
  [bootloader]C:\Speedway\ZynqSW\2017_4\SDK_Workspace\zynq_fsbl_0\Release\zynq_fsbl_0.elf
  C:\Speedway\ZynqSW\2017_4\SDK_Workspace\ZynqHW\Z_system_wrapper.bit
  C:\Speedway\ZynqSW\2017_4\SDK_Workspace\Test_Peripherals\Release\Test_Peripherals.elf
}
```

Figure 6 – BIF Contents

Questions

- *Is the order of the images critical when generating the boot image? If so, list the order.*
.....
- *True or False? The Create Zynq Boot Image tool creates a boot image for the QSPI flash.*
.....

True. SDK uses a utility called bootgen to generate a boot container file that contains the FSBL, the Programmable Logic bitstream (optional), and the Standalone user application

Experiment 2: Write and boot from QSPI

First, we will program the QSPI with the MCS file from within SDK. Then we will boot the test application.

Experiment 2 General Instruction:

Program the QSPI with the MCS file. Disconnect power then change the MODE switch to QSPI. Power on the board and boot from QSPI.

Experiment 2 Step-by-Step Instructions:

1. Set up and turn on your board as before (USB-UART, and JTAG, turn power on).
2. In SDK, select **Xilinx Tools** → **Program Flash**.
3. Click the **Browse** button, and browse to the `SDK_Workspace\Test_Peripherals\bootimage` folder then select the `Test_Peripherals.mcs` image. Select it and click **Open**.
4. Now we must specify the FSBL file, click the **Browse** button and browse to the `SDK_Workspace\zynq_fsbl_0\Release` folder and then select the `zynq_fsbl_0.elf` file
5. Click **Program**.

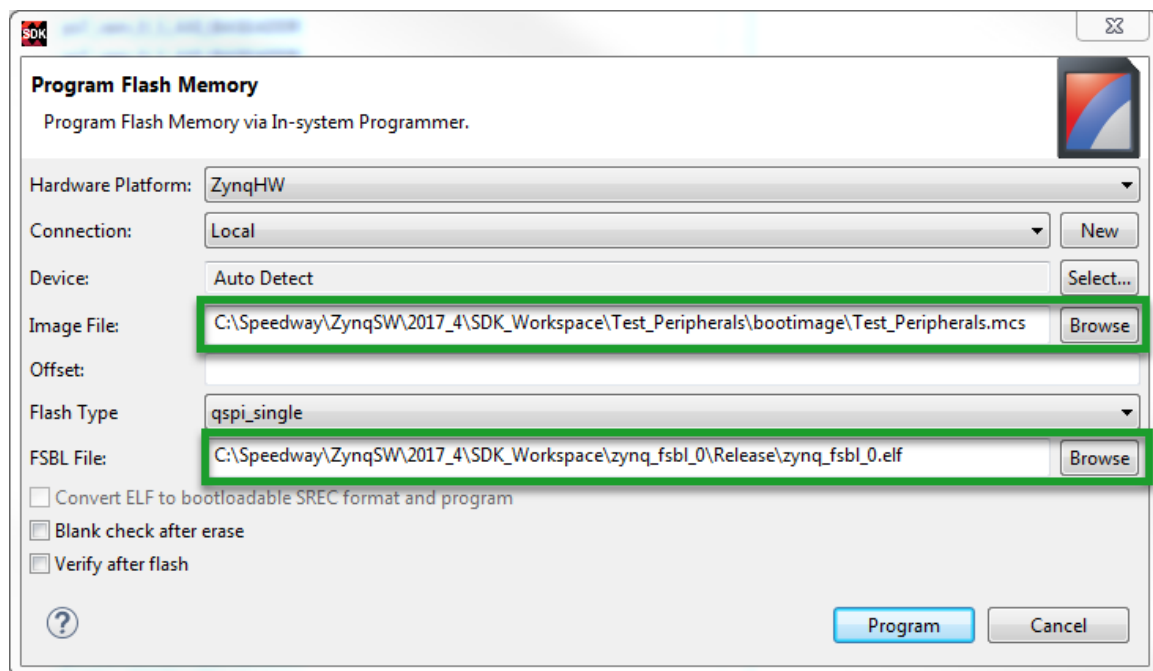


Figure 7 – Program QSPI Flash Memory

6. The operation should take approximately 1 minute. You should see something similar to the following in the console window.

```

***** Xilinx Program Flash
***** Program Flash v2017.4 (64-bit)
***** SW Build 2086221 on Fri Dec 15 20:55:39 MST 2017
***** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.

Connecting to hw_server @ TCP:127.0.0.1:3121

WARNING: Failed to connect to hw_server at TCP:127.0.0.1:3121
Attempting to launch hw_server at TCP:127.0.0.1:3121

Connected to hw_server @ TCP:127.0.0.1:3121
Available targets and devices:
Target 0 : jsn-openjtag2-1234-oj1A
        Device 0: jsn-openjtag2-1234-oj1A-4ba00477-0

Retrieving Flash info...

Initialization done, programming the memory
BOOT_MODE REG = 0x00000000
f probe 0 0 0

Performing Erase Operation...
Erase Operation successful.
INFO: [Xicom 50-44] Elapsed time = 2 sec.
Performing Program Operation...
0%...50%...100%
Program Operation successful.
INFO: [Xicom 50-44] Elapsed time = 7 sec.

Flash Operation Successful

```

Figure 8 – QSPI Programmed Successfully

7. Power off the board by unplugging J2 micro-USB connector.
8. Set the MiniZed boot mode switch SW1 to QSPI mode ('F' for Flash) as shown below.

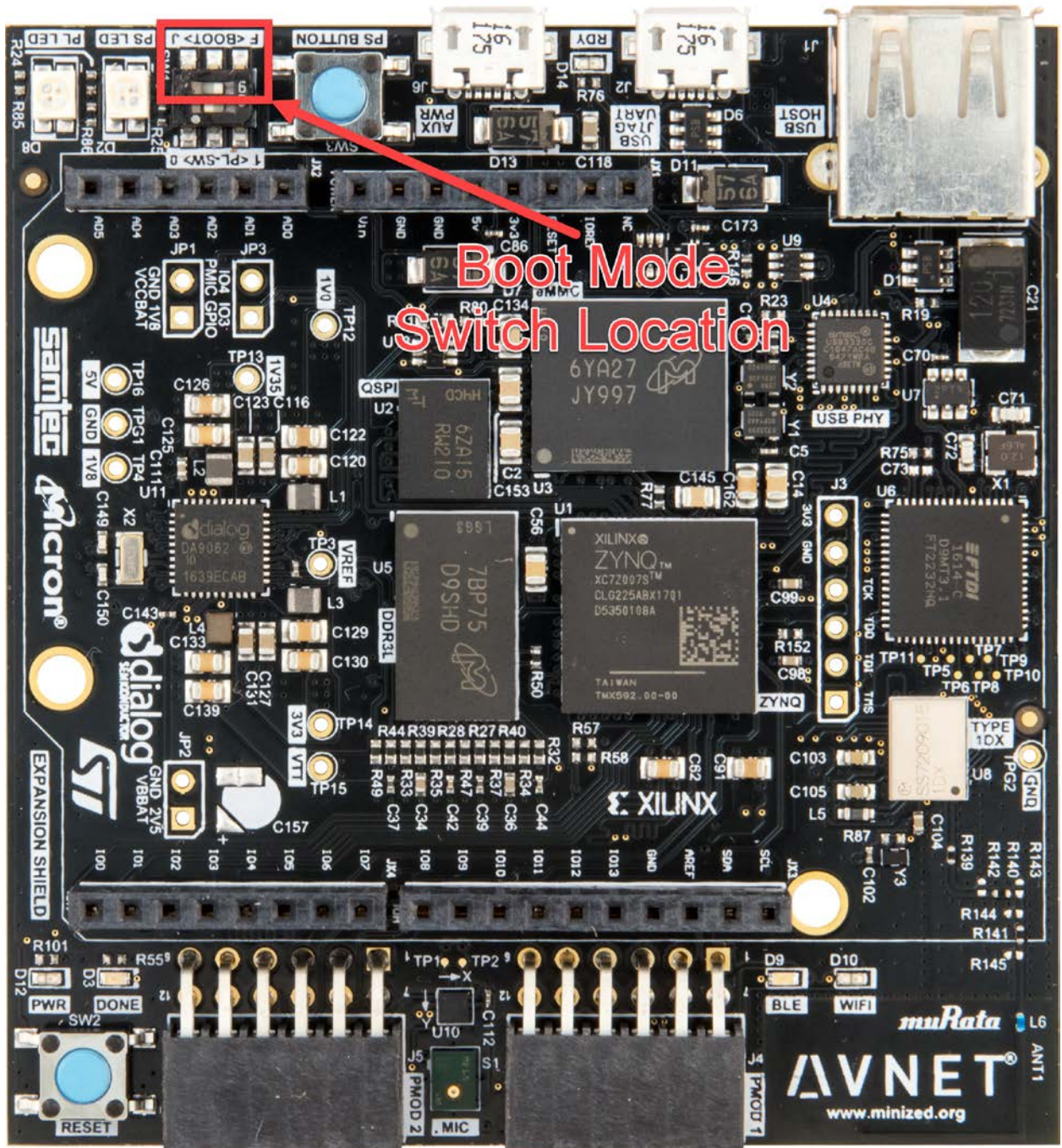


Figure 9 – MiniZed Switch Location

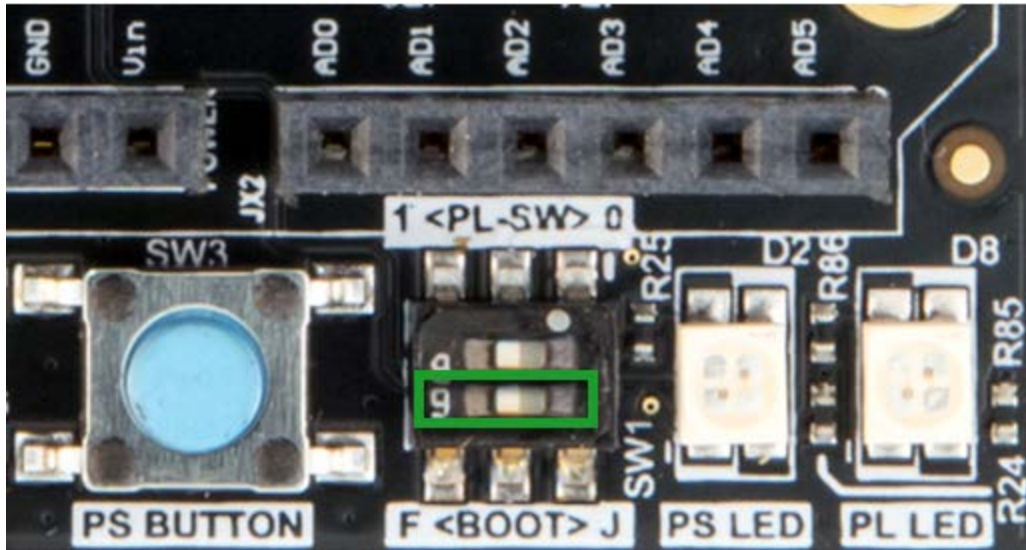
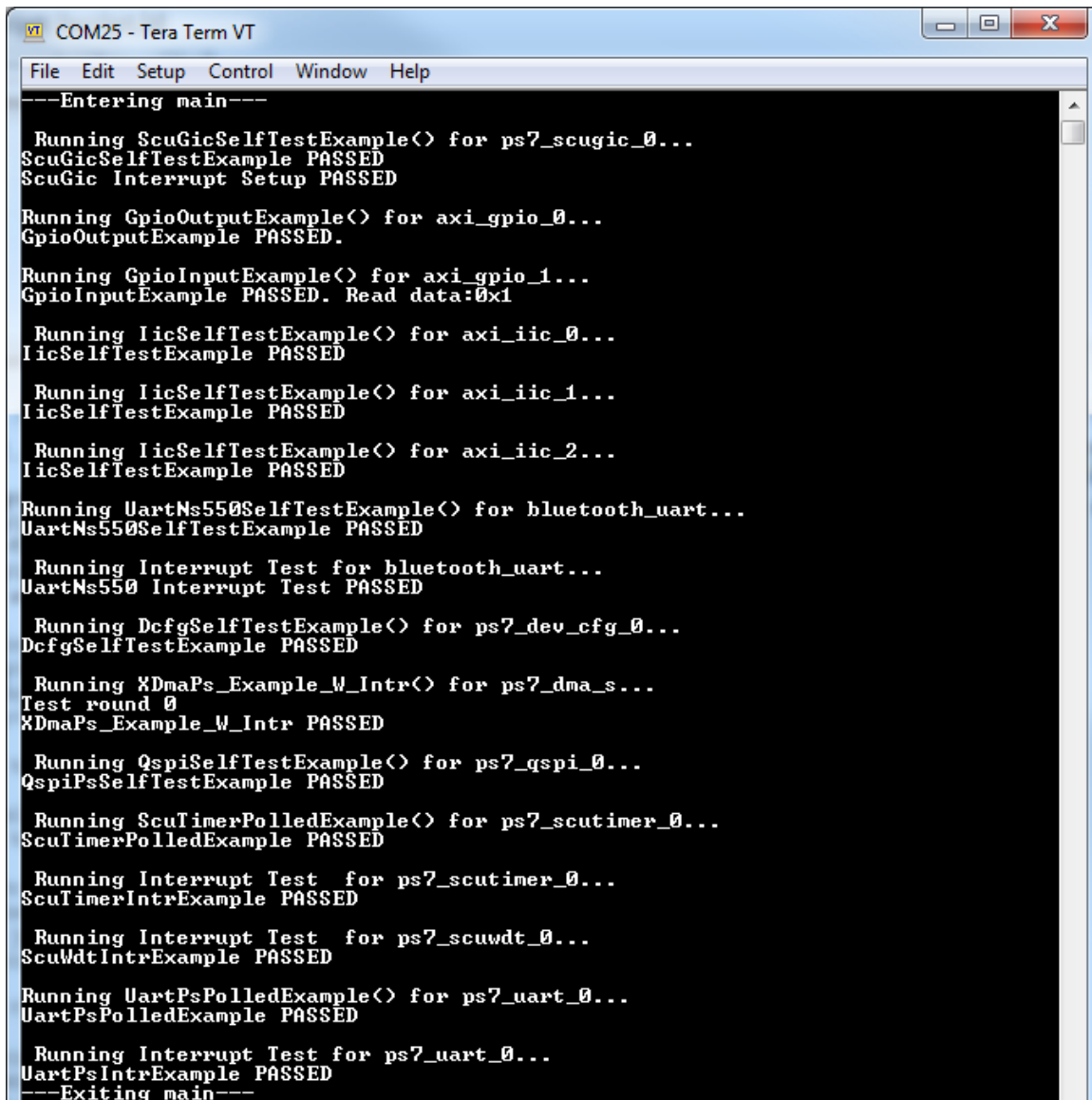


Figure 10 – QSPI/Flash Boot Mode

9. Close or disconnect the terminal that may have previously been open on your PC.
10. Power on the board by reconnecting the USB-JTAG-UART J2 connection. A blue LED should illuminate meaning the bitstream was loaded.
11. Launch a terminal program (TeraTerm) with the 115200/8/n/1/n settings.
12. Push the RESET button (SW2). You should see the results in the terminal.

A screenshot of a Tera Term VT window titled 'COM25 - Tera Term VT'. The window has a menu bar with 'File', 'Edit', 'Setup', 'Control', 'Window', and 'Help'. The main text area shows the output of a test program. It starts with '---Entering main---' and then lists several test functions and their results: 'Running ScuGicSelfTestExample() for ps7_scugic_0...' (PASSED), 'Running GpioOutputExample() for axi_gpio_0...' (PASSED), 'Running GpioInputExample() for axi_gpio_1...' (PASSED, Read data:0x1), 'Running IicSelfTestExample() for axi_iic_0...' (PASSED), 'Running IicSelfTestExample() for axi_iic_1...' (PASSED), 'Running IicSelfTestExample() for axi_iic_2...' (PASSED), 'Running UartNs550SelfTestExample() for bluetooth_uart...' (PASSED), 'Running Interrupt Test for bluetooth_uart...' (PASSED), 'Running DcfgSelfTestExample() for ps7_dev_cfg_0...' (PASSED), 'Running XDmaPs_Example_W_Intr() for ps7_dma_s...' (PASSED, Test round 0), 'Running QspiSelfTestExample() for ps7_qspi_0...' (PASSED), 'Running ScuTimerPolledExample() for ps7_scutimer_0...' (PASSED), 'Running Interrupt Test for ps7_scutimer_0...' (PASSED), 'Running Interrupt Test for ps7_scuwdt_0...' (PASSED), 'Running UartPsPolledExample() for ps7_uart_0...' (PASSED), and 'Running Interrupt Test for ps7_uart_0...' (PASSED). The window ends with '---Exiting main---'.

```
---Entering main---

Running ScuGicSelfTestExample() for ps7_scugic_0...
ScuGicSelfTestExample PASSED
ScuGic Interrupt Setup PASSED

Running GpioOutputExample() for axi_gpio_0...
GpioOutputExample PASSED.

Running GpioInputExample() for axi_gpio_1...
GpioInputExample PASSED. Read data:0x1

Running IicSelfTestExample() for axi_iic_0...
IicSelfTestExample PASSED

Running IicSelfTestExample() for axi_iic_1...
IicSelfTestExample PASSED

Running IicSelfTestExample() for axi_iic_2...
IicSelfTestExample PASSED

Running UartNs550SelfTestExample() for bluetooth_uart...
UartNs550SelfTestExample PASSED

Running Interrupt Test for bluetooth_uart...
UartNs550 Interrupt Test PASSED

Running DcfgSelfTestExample() for ps7_dev_cfg_0...
DcfgSelfTestExample PASSED

Running XDmaPs_Example_W_Intr() for ps7_dma_s...
Test round 0
XDmaPs_Example_W_Intr PASSED

Running QspiSelfTestExample() for ps7_qspi_0...
QspiPsSelfTestExample PASSED

Running ScuTimerPolledExample() for ps7_scutimer_0...
ScuTimerPolledExample PASSED

Running Interrupt Test for ps7_scutimer_0...
ScuTimerIntrExample PASSED

Running Interrupt Test for ps7_scuwdt_0...
ScuWdtIntrExample PASSED

Running UartPsPolledExample() for ps7_uart_0...
UartPsPolledExample PASSED

Running Interrupt Test for ps7_uart_0...
UartPsIntrExample PASSED

---Exiting main---
```

Figure 11 – Results from QSPI boot of Periph_Test

13. Close the terminal.

14. Unplug the USB-UART-JTAG cable.

This concludes Lab 7.

Revision History

Date	Version	Revision
12 Nov 13	01	Initial release
23 Nov 13	02	Revisions after pilot
01 May 14	03	MicroZed.org Training Course Release
10 Dec 14	04	Updated to Vivado 2014.3
07 Jan 15	05	Updated to Vivado 2014.4
12 Mar 15	06	Finalize SDK 2014.4
Oct 15	07	Updated to SDK 2015.2
Aug 16	08	Updated to SDK 2016.2
Jun 17	09	Updated to 2017.1 for MiniZed + Rebranding
Feb 18	10	Updated to Vivado/SDK 2017.4

Resources

www.microzed.org

www.minized.org

www.picozed.org

www.zedboard.org

www.xilinx.com/zyng

www.xilinx.com/sdk

www.xilinx.com/vivado

www.xilinx.com/support/documentation/sw_manuals/ug949-vivado-design-methodology.pdf

www.xilinx.com/support/documentation/sw_manuals/ug1046-ultrafast-design-methodology-guide.pdf

Answers

- *Is the order of the images critical when generating the boot image? If so, list the order.*

YES! FSBL ELF, then (optional) bitstream, then application ELF

- *True or False? The Create Zynq Boot Image tool creates a boot image for the QSPI flash..*

True. SDK uses a utility called bootgen to generate a boot container file that contains the FSBL, the Programmable Logic bitstream (optional), and the Standalone user application.