

Developing Zynq Software with Xilinx SDK

Lab 7

Boot from Flash



November 2013
Version 03

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
3. Application

This Lab will show you how to combine these pieces together to create the boot images for both QSPI and SD Card. 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
- Write and boot from SD Card

Experiment 1: Create the Boot Image

The first step is to create the non-volatile boot images for ZedBoard. ZedBoard has two non-volatile bootable sources, QSPI and SD Card.

Experiment 1 General Instruction:

Change the configuration for the Peripheral Test to Release. Create a boot image for both QSPI and SD Card.

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 ~140K, targeted at the on-chip RAM_0.

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. We would need to modify the linker script again, or modify the linker to put this application in the remaining RAM0 and also RAM1.

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**.

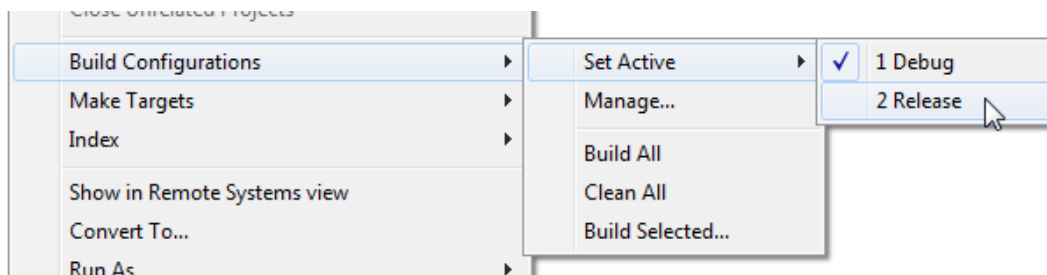


Figure 1 – Change to Release Configuration

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.

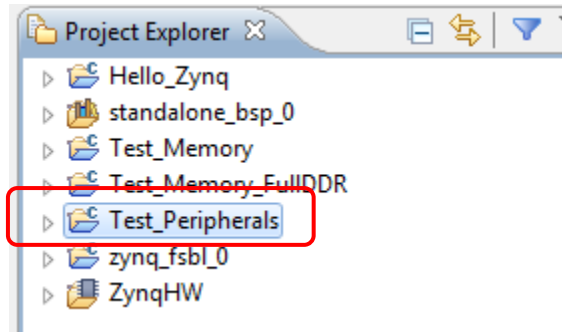


Figure 2 – Select Application to Boot

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

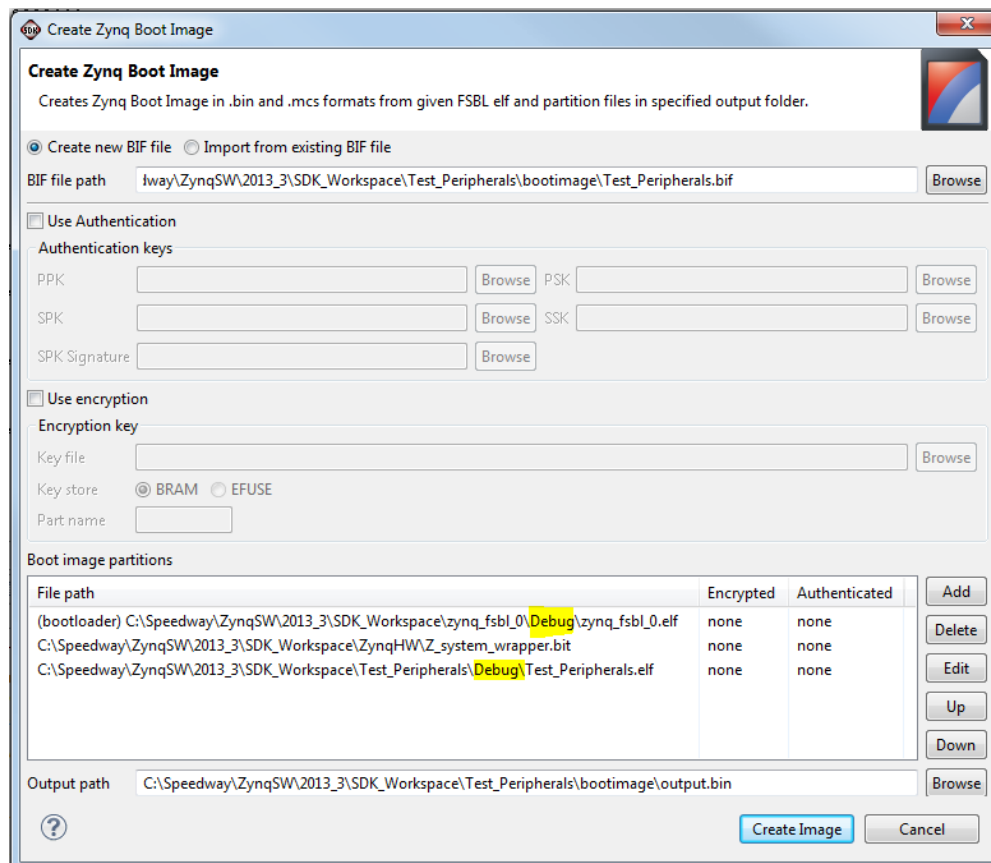


Figure 3 – Initial Create Zynq Boot Image Dialog

5. If your dialog looks like the figure below with no pre-populated files, then your Test_Peripherals application may have an issue. Click **Cancel**, then right-click on Test_Peripherals and select **Clean Project**.

Create Zynq Boot Image

Create Zynq Boot Image

✖ BIF File path is empty

☒ Create new BIF file ☐ Import from existing BIF file

BIF file path Browse

☐ Use Authentication

Authentication keys

PPK Browse PSK Browse

SPK Browse SSK Browse

SPK Signature Browse

☐ Use encryption

Encryption key

Key file Browse

Key store ☒ BRAM ☐ EFUSE

Part name

Boot image partitions

File path	Encrypted	Authenticated

Add
Delete
Edit
Up
Down

Output path Browse

? Create Image Cancel

Figure 4 – Create Zynq Boot Image Appears Blank

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. We will not be using Authentication or Encryption, so these checkboxes should be left unchecked.

The *Boot Image Partitions* is prepopulated with the FSBL and Application ELF ***Debug*** images as well as the bitstream. This is technically not correct since ***Release*** is the current active configuration. We've been told that this will be fixed in a future SDK release. For 2013.3, we must manually switch the Boot Image files to meet our goals.

The order of the files is important. The FSBL is loaded before the Application. When a bitstream is present in the design, it should be added between the FSBL and the application. Why? Because one function of the FSBL is to program the PL. After the PL is configured, the application is loaded.

8. Select the `zynq_fsbl_0.elf` entry. Click **Edit**. Browse to the `SDK_Workspace\zynq_fsbl_0\Release` folder and select `zynq_fsbl_0.elf`, then click **Open**. Then click **OK**.

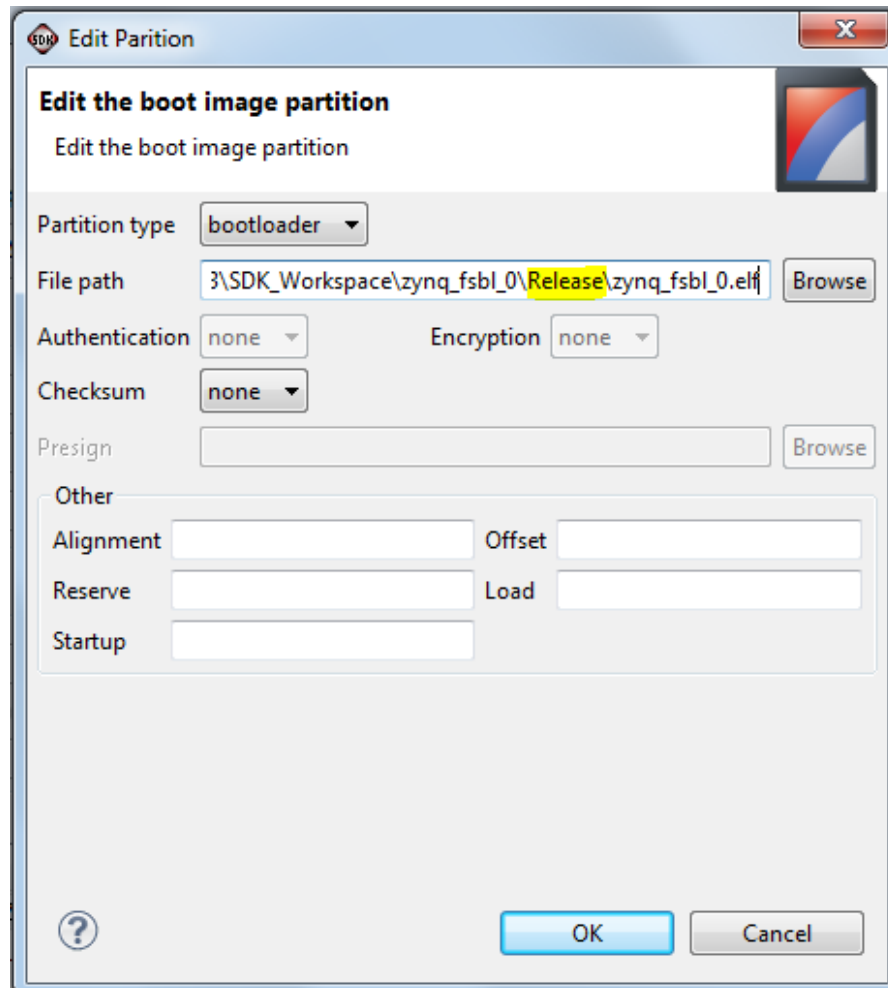


Figure 5 – Select Release Configuration for FSBL ELF

9. Repeat the process for the Test_Peripherals application.

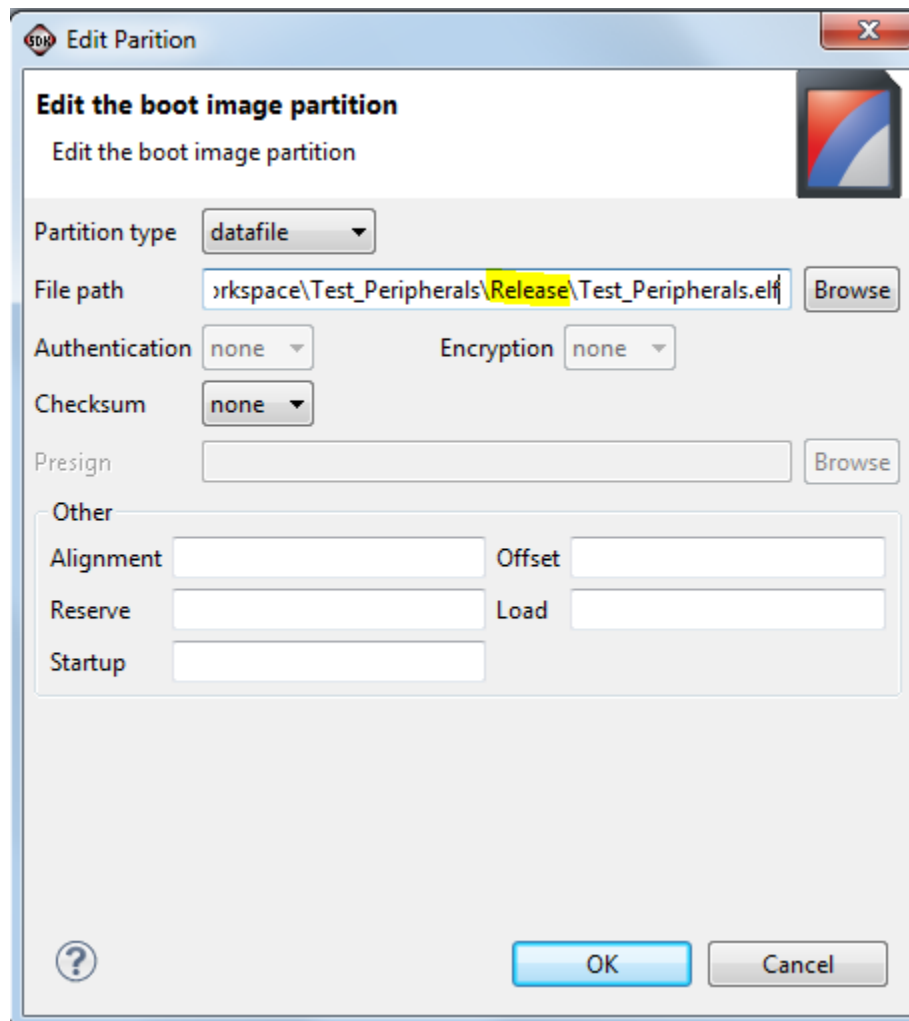


Figure 6 – Selecting Release Configuration of Test_Peripherals.elf

10. 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, so we will create that one first. The directory location of `Test_Peripherals\bootimage` is good. Change the output file name to **Test_Peripherals.bin**.

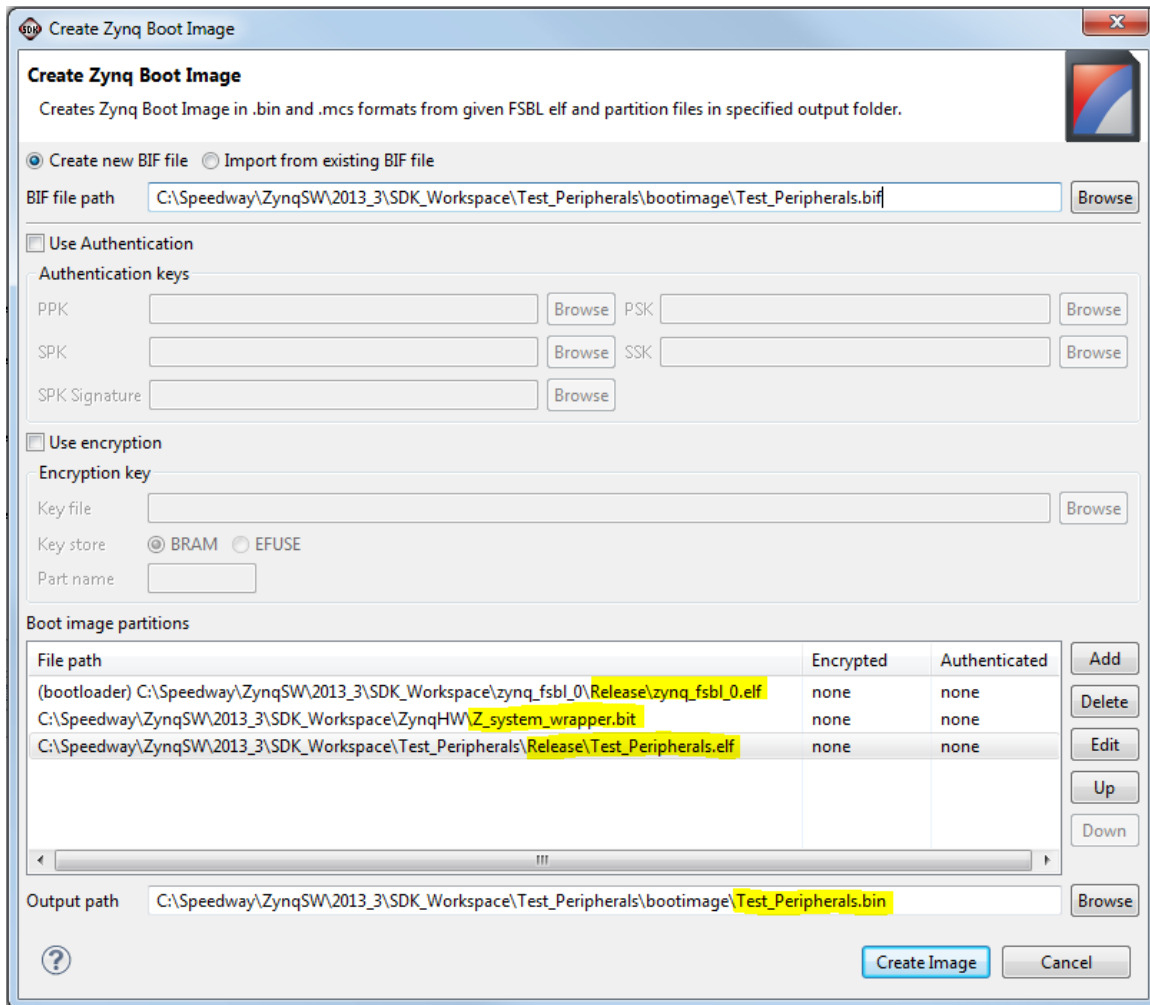
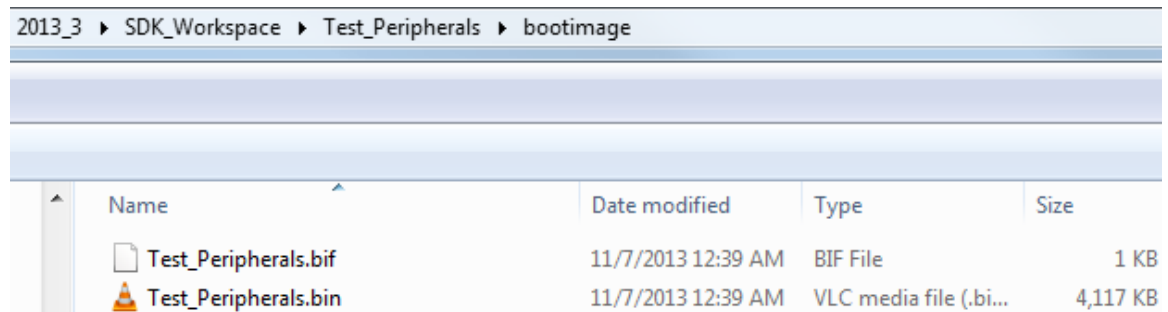


Figure 7 – Zynq Boot Image Dialog

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

11. Click **Create Image**.

12. Using Windows Explorer, navigate to the application directory, then into the newly created **bootimage** directory. Notice that two files have been created: .bif and .bin. If we were only going to boot from SD Card, this would be enough.





2013_3 > SDK_Workspace > Test_Peripherals > bootimage				
Name	Date modified	Type	Size	
 Test_Peripherals.bif	11/7/2013 12:39 AM	BIF File	1 KB	
 Test_Peripherals.bin	11/7/2013 12:39 AM	VLC media file (.bi...	4,117 KB	

Figure 8 - bootimage Directory

13. 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.

14. Launch the **Create Zynq Boot Image** utility again.
15. This time, it is OK to import the existing .bif file. Also, notice that the utility remembers our previous selection of the Release ELF files.
16. For the *Output path*, click **Browse**. Browse to the `SDK_Workspace\Test_Peripherals\bootimage` folder, then change the *Save as type* to **.mcs**. Type in **Test_Peripherals** as the *File name*. Click **Save**.

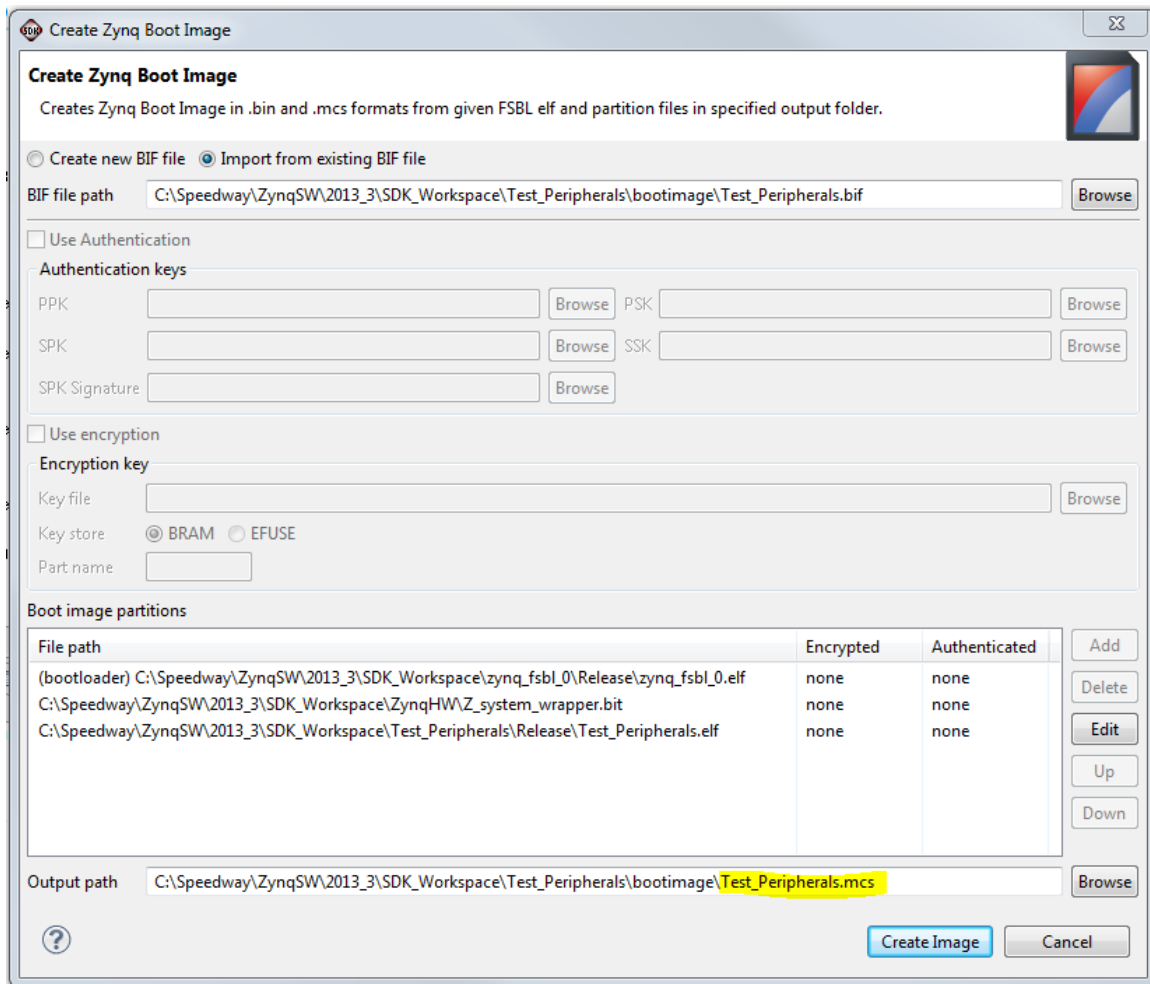


Figure 9 – Create the MCS Boot Image

17. Click **Create Image** and then check the bootimage folder again to ensure the .mcs has also now been created.




2013_3 ▶ SDK_Workspace ▶ Test_Peripherals ▶ bootimage				
folder				
	Name	Date modified	Type	Size
	 Test_Peripherals.bif	11/7/2013 12:39 AM	BIF File	1 KB
	 Test_Peripherals.bin	11/7/2013 12:39 AM	VLC media file (.bi...	4,117 KB
	 Test_Peripherals.mcs	11/7/2013 12:40 AM	MCS File	11,570 KB

Figure 10 – Test Peripherals Boot Images

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. Switch off power then change the MODE jumpers to QSPI. Power on the board and boot from QSPI.

Experiment 2 Step-by-Step Instructions:

1. Put ZedBoard in Cascaded JTAG mode by setting the MIO[6:2] each to the GND position.



Figure 11 – PLL Used, JTAG Boot, Cascaded JTAG: MIO[6:2] = 00000

2. Connect a micro-USB cable between the Windows Host machine and the ZedBoard connector J17 (JTAG).
3. Power on the ZedBoard.
4. In SDK, select **Xilinx Tools** → **Program Flash**.
5. 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**.
6. Click **Program**.

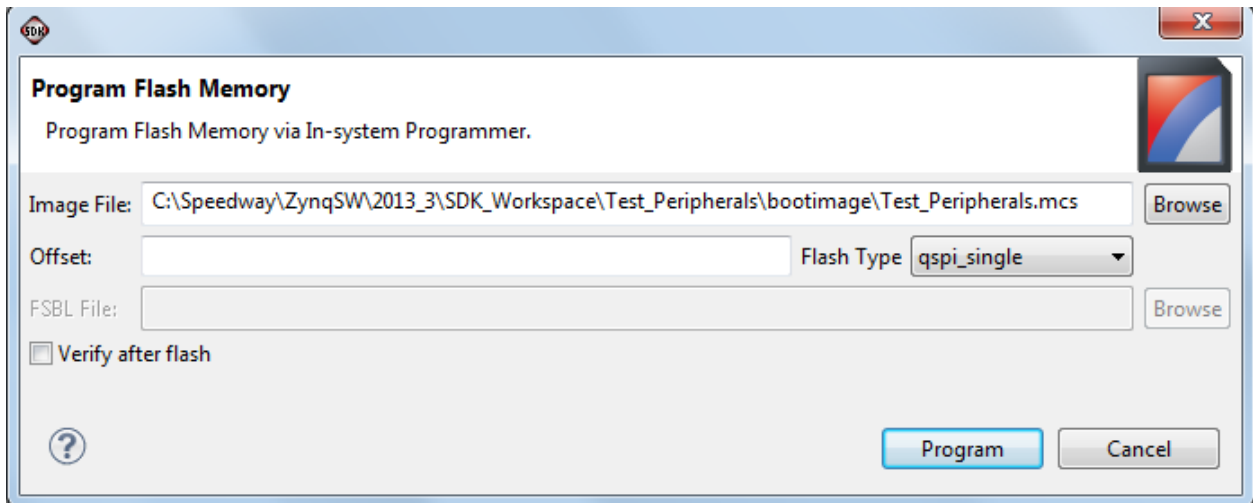


Figure 12 – Program QSPI Flash Memory

7. The operation should take approximately 1 minute. You should see the following in the Console window.

```
Setting PC with Program Start Address 0xffffc4800
0% Info:Performing Program operation.
.....3% .....6% .....9% .....12% .....15% .....18% .....21% .....24% .....
27% .....30% .....33% .....36% .....39% .....42% .....45% .....48% .....
51% .....54% .....57% .....60% .....63% .....66% .....69% .....72% .....
75% .....78% .....81% .....84% .....87% .....90% .....93% .....96% .....100% 100%

Info:Programmed flash successfully
Info:Closed cable successfully
```

Figure 13 – QSPI Programmed Successfully

8. Unplug the JTAG cable from the ZedBoard and turn the board OFF.
9. Set the ZedBoard Boot Jumpers to QSPI by moving MIO5 to the 3V3 position.

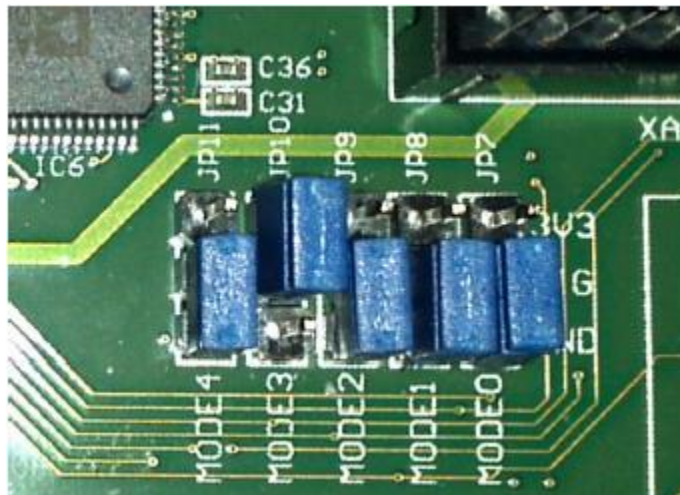
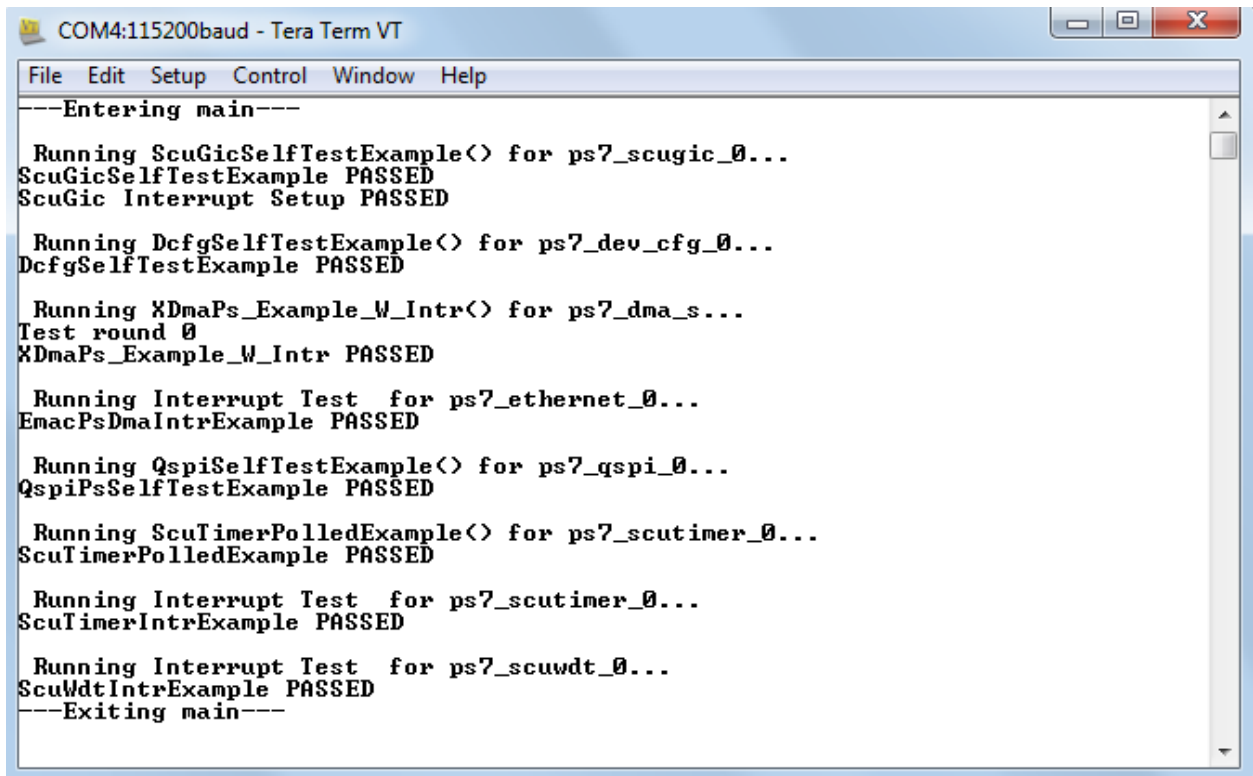


Figure 14 – QSPI Boot Mode

10. Close or disconnect the terminal that may have previously been open on your PC.
11. Plug in the USB-UART cable and then turn the board ON.
12. Launch a terminal program with the 115200/8/n/1/n settings.
13. Push the PS-RST button (BTN7). You should see the results in the terminal.



```
COM4:115200baud - Tera Term VT
File Edit Setup Control Window Help
---Entering main---

Running ScuGicSelfTestExample() for ps7_scugic_0...
ScuGicSelfTestExample PASSED
ScuGic Interrupt Setup 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 Interrupt Test for ps7_ethernet_0...
EmacPsDmaIntrExample 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
---Exiting main---
```

Figure 15 – Results from QSPI boot of Periph_Test

14. Close the terminal. Turn OFF the ZedBoard and unplug the USB-UART cable.

Experiment 3: Write and boot from the SD Card

Next, we will prepare the SD card with the bin file that SDK generated. Then we will boot the same test application from the SD card.

Experiment 3 General Instruction:

Make a copy of the .bin file, naming it boot.bin, on the SD card. Change the MODE jumpers to SD boot. Insert the card to ZedBoard and power on. Observe the results in the terminal.

Experiment 3 Step-by-Step Instructions:

1. To program a SD Card, insert the SD card into your PC (or a USB card reader plugged into your PC). Use Windows Explorer to browse to the SDK_Workspace\Test_Peripherals\bootimage folder. Copy the Test_Peripherals.bin image to the SD Card.
2. On the SD Card, rename the .bin file to **boot.bin**. THIS IS CRITICAL! The Stage 0 bootloader looks specifically for this file. Also, the name is not case sensitive.
3. Eject the SD card from the PC and insert it into the ZedBoard slot.
4. Set the ZedBoard Boot Jumpers to SD Card, with both JP2 and JP3 in the 2-3 position.

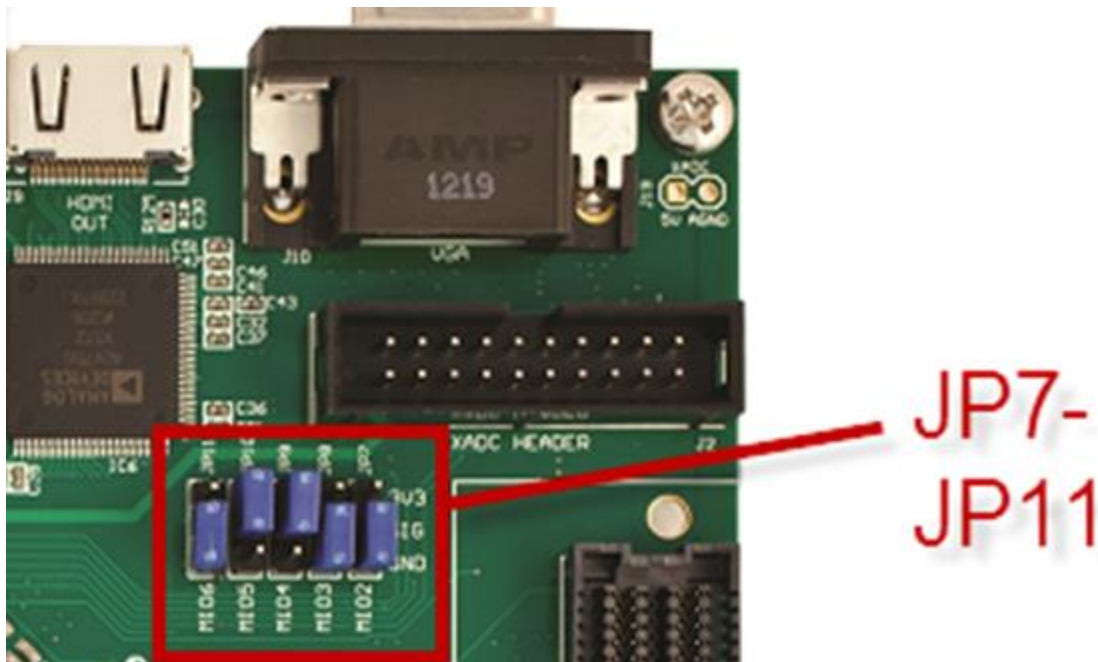


Figure 16 – SD Card Boot Mode

5. Close or disconnect the terminal that may have previously been open on your PC.
6. Plug in the USB-UART cable and turn the board ON.
7. Launch a terminal program with the 115200/8/n/1/n settings.
8. Push the PS-RST button. You should see the same results in the terminal that were shown during the QSPI test.

Questions:

Answer the following questions:

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

- *True or False? The Create Zynq Boot Image tool creates both the QSPI and SD card images in one operation.*

- *What is the required name for the .bin file on the SD card when booting?*

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	ZedBoard.org Training Course Release

Answers

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

YES! FSBL ELF, then bitstream, then application ELF

- *True or False? The Create Zynq Boot Image tool creates both the QSPI and SD card images in one operation.*

False. SDK used to operate that way, but in 2013.3, it requires two separate operations.

- *What is the required name for the .bin file copied to the SD card?*

boot.bin