Overview

Thus far, we have relied on the tools to configure the Zynq PS properly. Although it wasn't explicitly pointed out previously, performing a *Run As* or *Debug As* operation in SDK first sources the `ps7_init.tcl` file that was created during the export to SDK. You can see this in the *Configurations* dialog under the *Target Setup* tab.

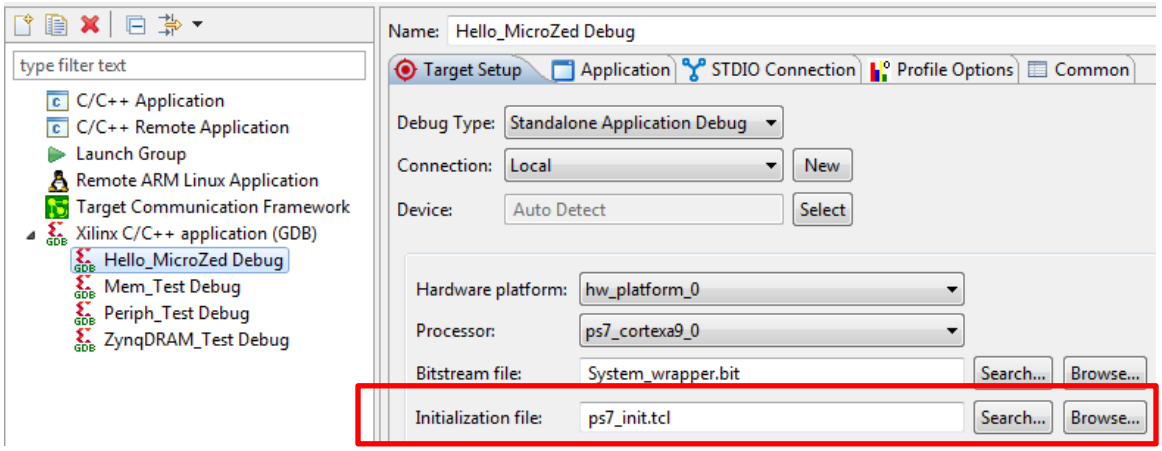


Figure 1 – TCL Used to Initialize PS

In a true, embedded application, you will not have a JTAG cable connected that can transfer these settings. Your code must be able to do this before transferring control to an application. The code that sets up the Zynq PS is called the First Stage Boot Loader (FSBL).

Objectives

When this tutorial is complete, you will be able to:

- Create the FSBL
- Prepare the boot image
- Write and boot from QSPI
- Write and boot from the microSD Card

© 2014 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Experiment Setup

Software

The software used to test this reference design is:

- Windows-7 64-bit
- Xilinx SDK 2014.2
- Silicon Labs CP201x USB-to-UART Bridge Driver
  - www.microzed.org → Support → Documentation → MicroZed Silicon Labs CP210x USB-to-UART Setup Guide
- Tera Term or another terminal emulator
  - We will be booting outside of the SDK environment, so the internal terminal will no longer work.
  - For additional information on setting up Tera Term, see the *USB-to-UART Setup Guide* listed above.

Hardware

The hardware setup used to test this reference design includes:

- Win-7 PC with a recommended 1.6 GB RAM available for the Xilinx tools to complete a XC7Z010 design[1]
- Avnet MicroZed
- USB cable (Type A to Micro-USB Type B) – *one included in kit*
- JTAG Programming Cable (Platform Cable, Digilent HS1 or HS2 cable)
  - If you don't already have a JTAG Cable, Avnet recommends the Digilent HS2 Cable
  - http://www.em.avnet.com/en-us/design/drc/Pages/Digilent-JTAG-HS2-Programming-Cable.aspx
- microSD card – *one included in kit*
- microSD card adapter – *SD form factor adapter included in kit*

[1] Refer to www.xilinx.com/design-tools/vivado/memory.htm

Experiment 1: Create the FSBL

The first step is to create the FSBL application. This is a C program that embeds all the Zynq internal register settings that were established during the Vivado Block Design.

Similar to the flow for creating the Hello_World application, in SDK create a First Stage Bootloader application.

1. **File → New → New Application Project**

2. Name it something like MZ_FSBL and **Create New** BSP. The reason for creating a new BSP is that the FSBL BSP requires a library for the Flash, and the tools will automatically include this for us if we allow it to create the BSP.

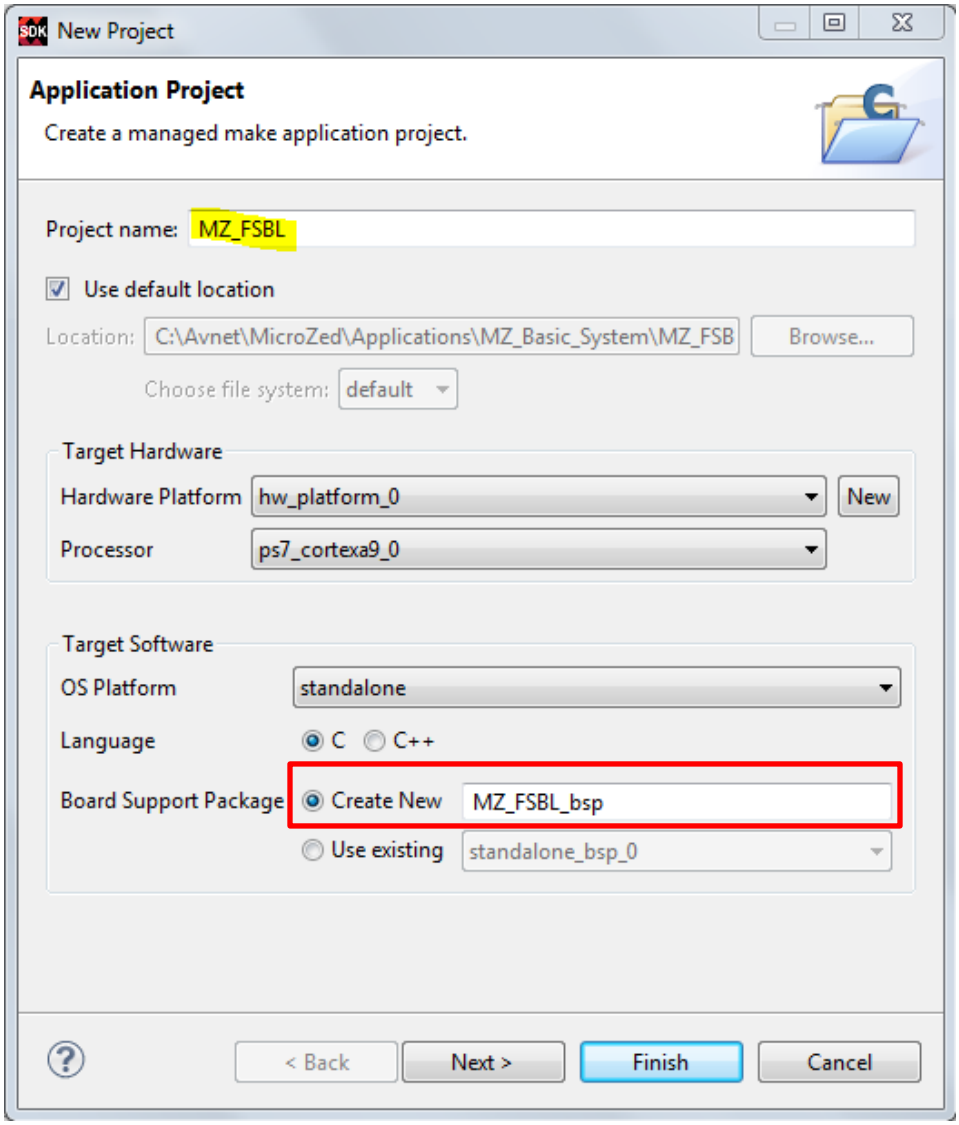


Figure 2 – FSBL Application

3. Click **Next**

4. Select **Zynq FSBL**

5. Click **Finish**

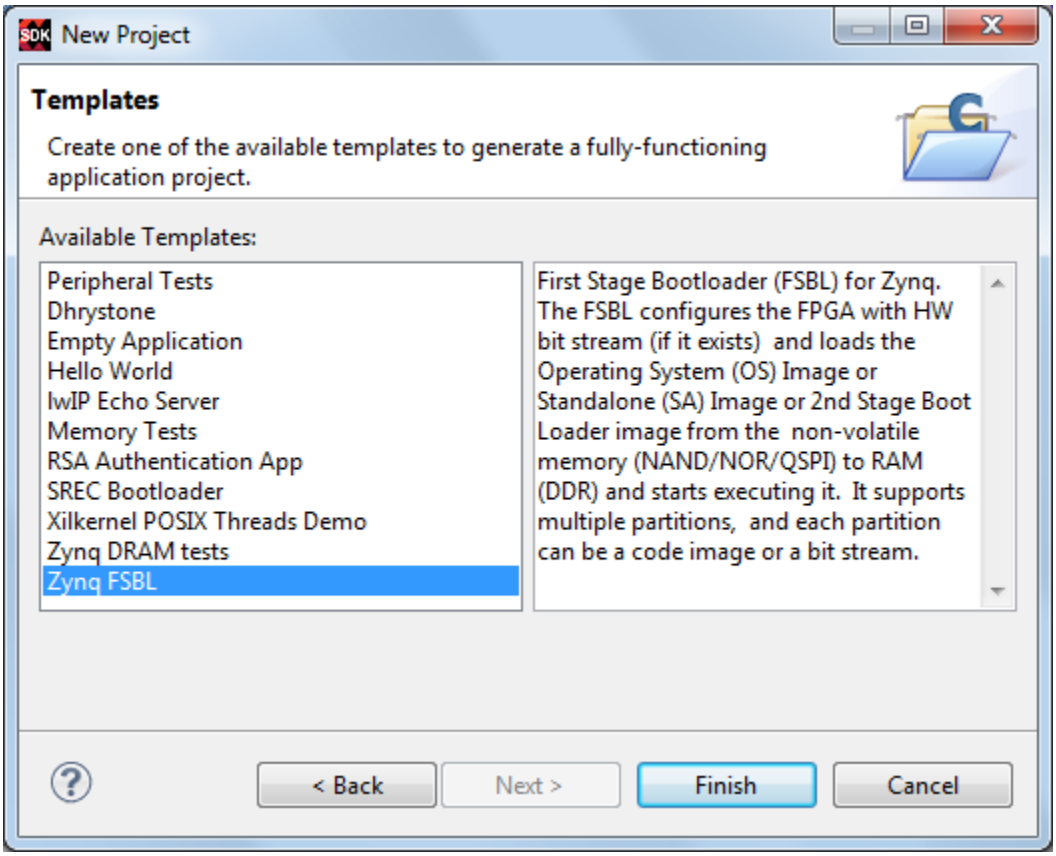


Figure 3 – Zynq FSBL Template for Application

6. Notice this message at the top of the console:

```
12:41:03 **** Build of configuration Debug for project MZ_FSBL ****
```

7. You can see what the compiler settings for the Debug configuration are by right-clicking on the **Debug** folder underneath the MZ_FSBL application.

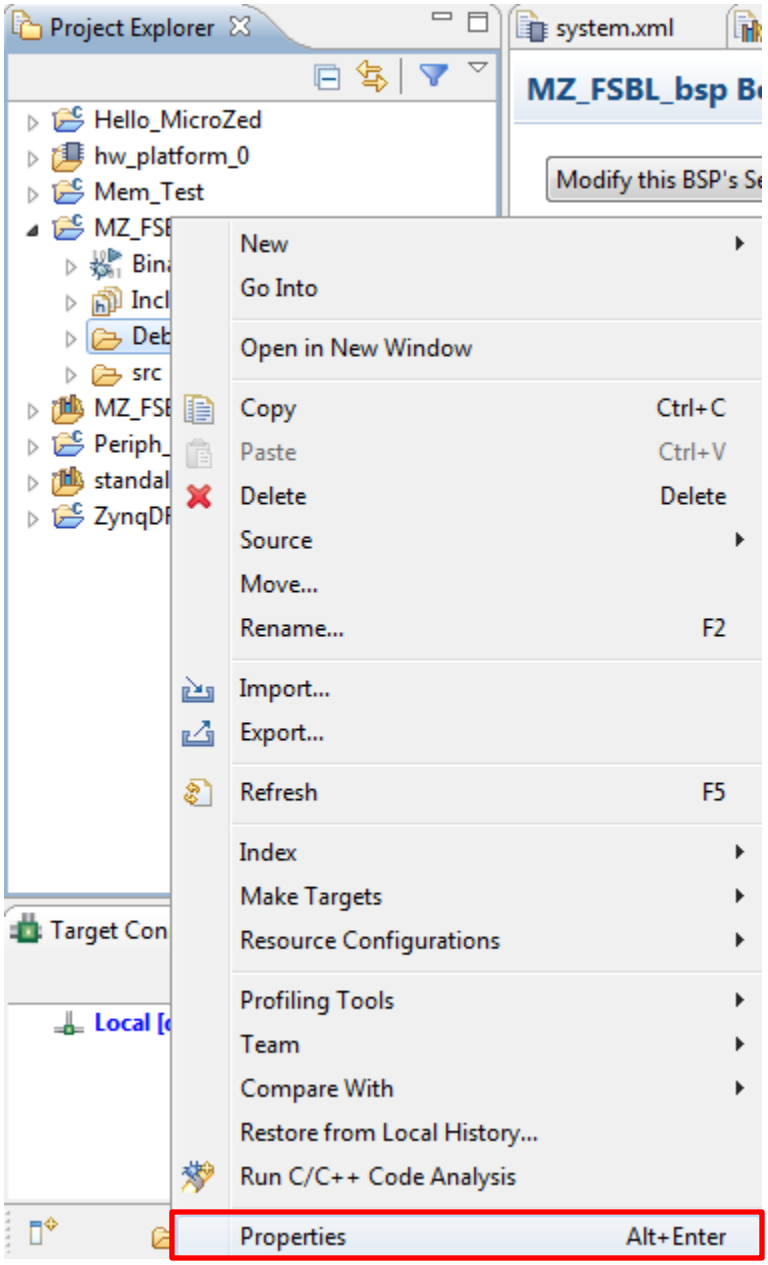


Figure 4 – See Properties for Debug Configuration

8. Expand *C/C++ Build* and select **Settings**. Then under *Tool Settings*, expand *ARM gcc compiler* and look at **Optimization** and **Debugging**. Notice that the Optimization Level is at None and Debug Level is at Maximum.

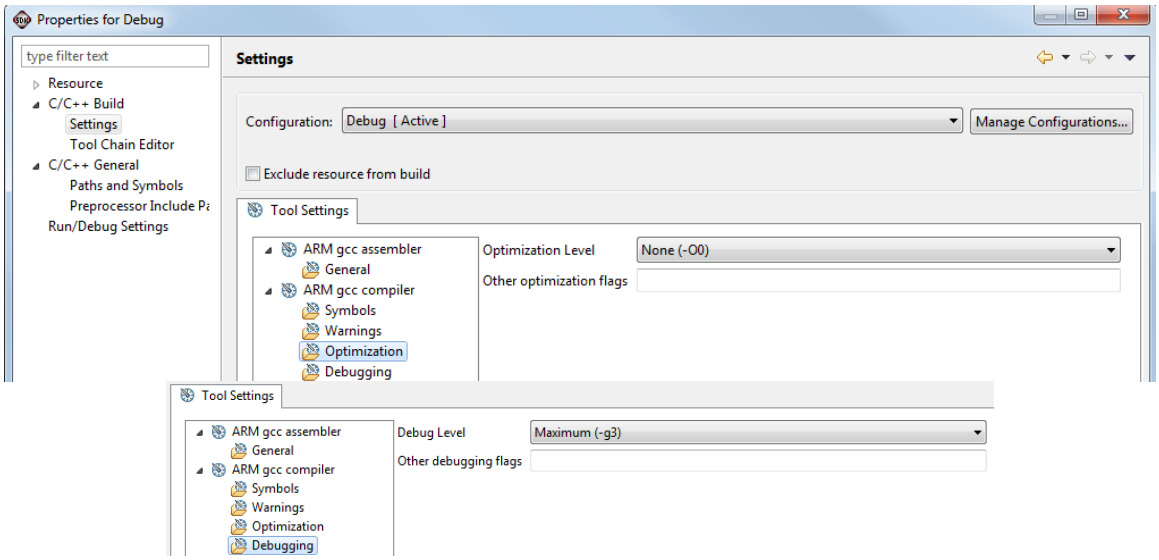


Figure 5 – Debug Configuration Optimization and Debugging

9. Click the **Manage Configurations** button. Select **Release** and then click the **Set Active** button. Click **OK**.

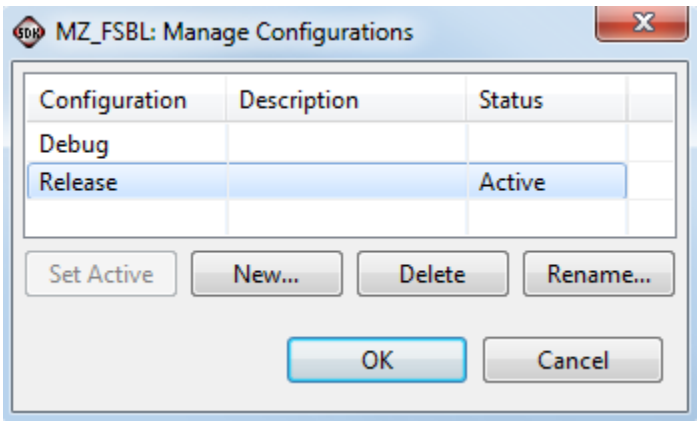


Figure 6 – Set Active Configuration to Release

10. Change the pull-down for *Configuration* to **Release [Active]**. Check the Optimization and Debugging settings for this configuration as a comparison. Click **OK**. The Release configuration should automatically be built.

11. You can click on the MZ_FSBL.elf.size file under each configuration in *Project Explorer* to compare sizes.

    a. Debug = 0x26004

    b. Release = 0x2480c

12. Since we will also not be debugging the actual application either, you should also change the application active configuration to **Release**. Right-click on the Hello_MicroZed and Periph_Test applications one at a time, then select **Build Configurations → Set Active Release**.

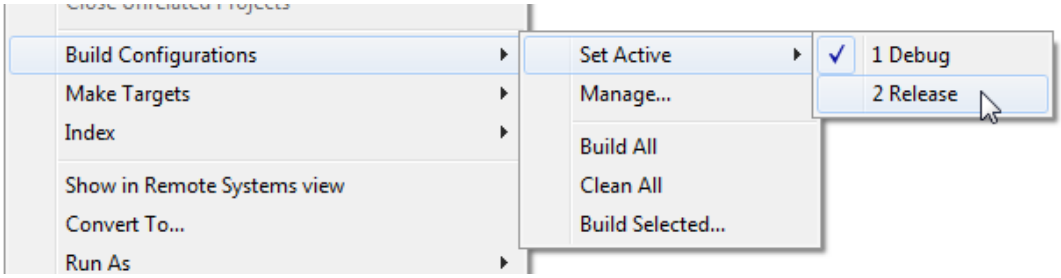


Figure 7 – Change to Release Configuration

13. Select **Project → Build All** to force the new configurations to build.

Experiment 2: Prepare the Boot Image

The next step is to create a non-volatile boot image for MicroZed. MicroZed has two non-volatile bootable sources, QSPI and SD Card.

1. In SDK, select either Hello_MicroZed or Periph_Test (Mem_Test in its default state will not successfully boot). Periph_Test is a bit more interesting, so that is recommended.

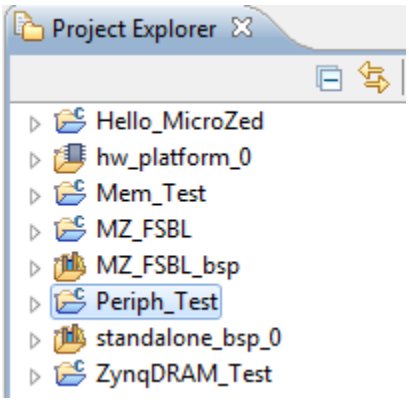


Figure 8 – Select Application to Boot

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

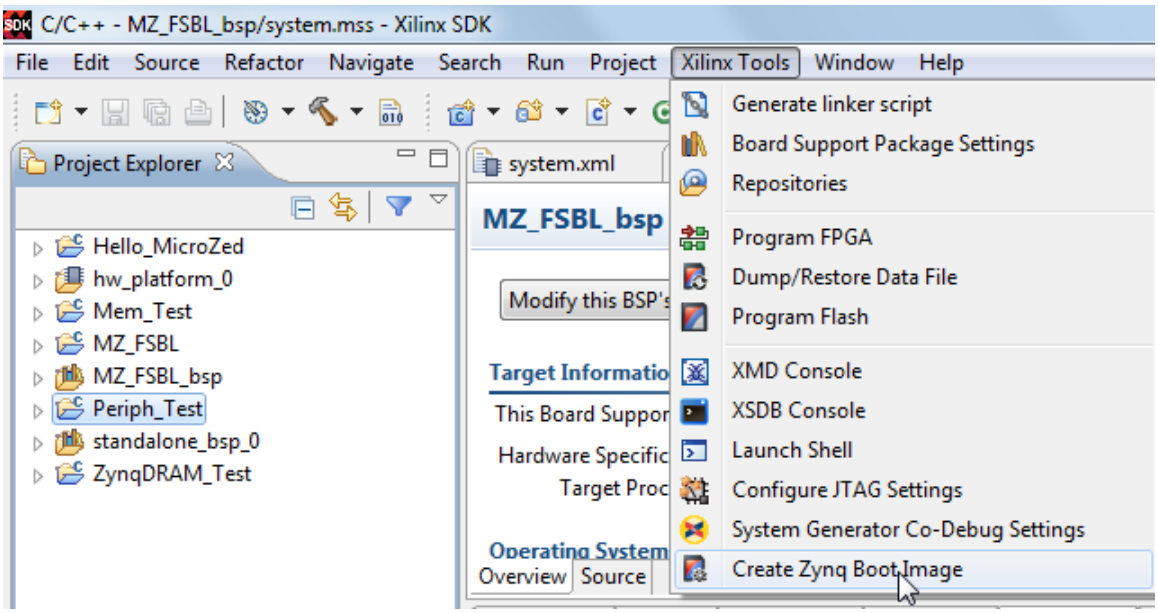


Figure 9 - Create Zynq Boot Image

This will preload the FSBL ELF and Application ELF images, but it does not find the bitstream by default. The order of the files is important. The FSBL is first, followed by the bitstream, followed by the Application. One function of the FSBL is to program the PL. After the PL is configured, the application is loaded. The Boot Image dialog has successfully detected that the Release configuration is Active.

3. Next to *Boot image partitions*, click the **Add** button.

4. Click **Browse** next to *File path***.** Browse to

   `C:\Avnet\MicroZed\Applications\MZ_Basic_System\hw_platform_0`

5. Select **`System_wrapper.bit`** and click **Open**. Click **OK**.

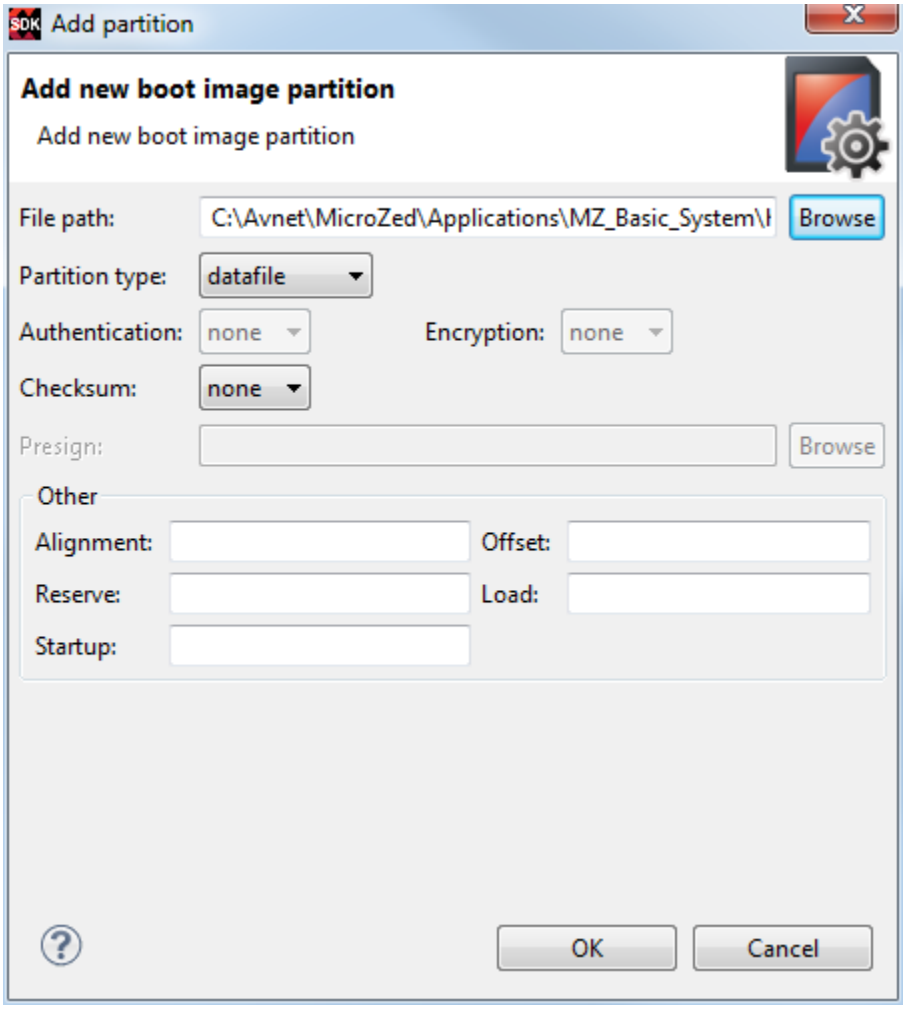


Figure 10 – Add Partition for Bitstream

6. Now select the .bit file and click **Up**. The *Boot image partitions* should looks as follows:

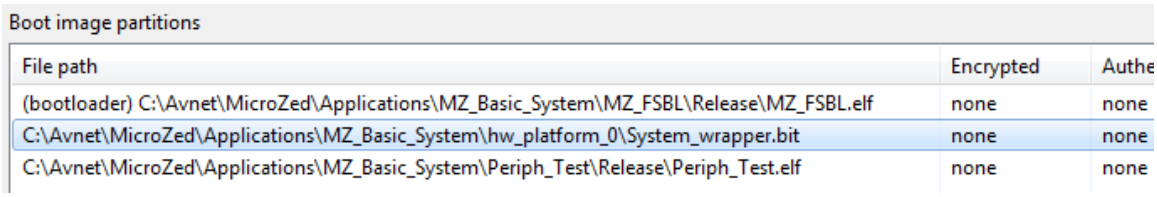


Figure 11 – Boot Image Partitions After Adding and Moving Bitstream

Notice that by default, the output path points to BOOT.bin, which is the bootimage required for SD boot. We will create this one first, and then we will need to repeat the operation for the QSPI bootimage (mcs).

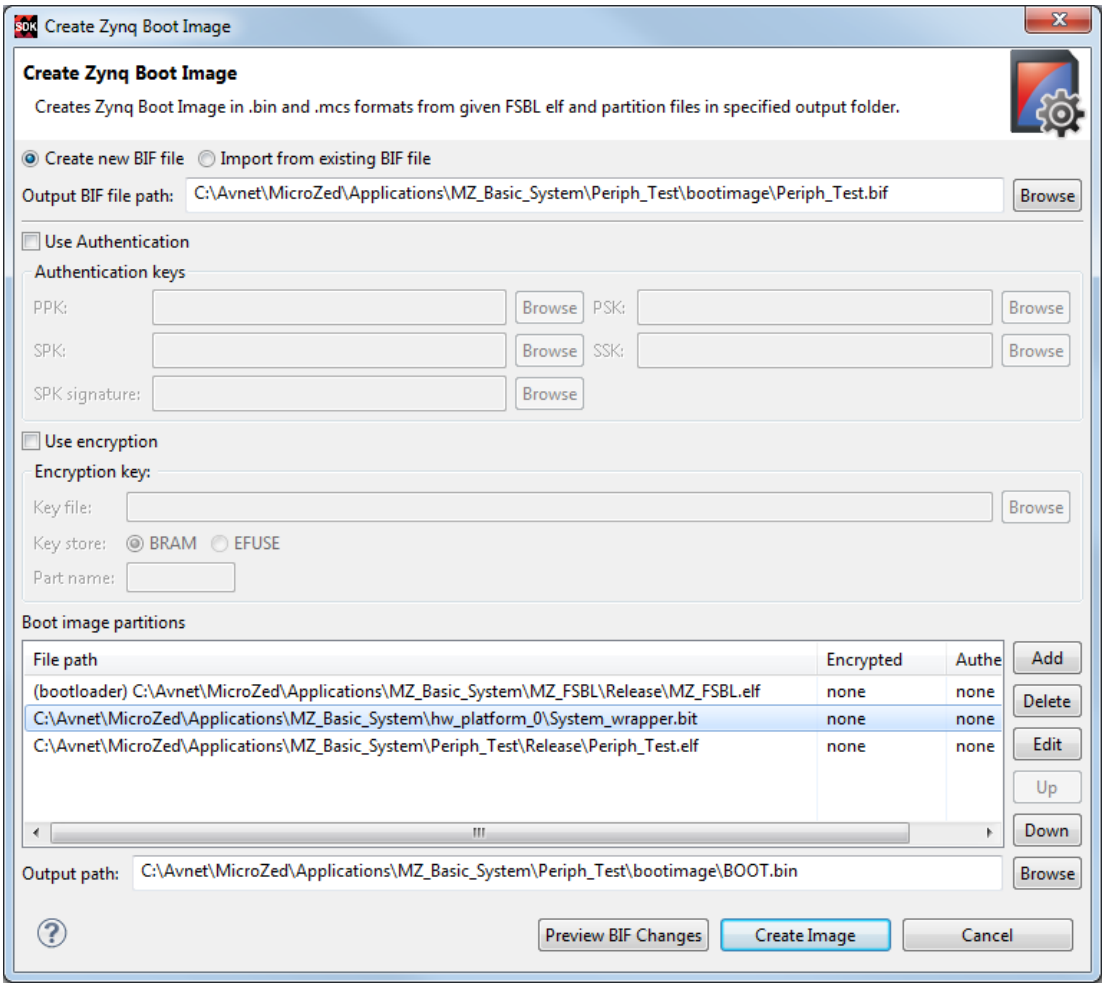


Figure 12 – Create Image

7. Click **Create Image**.

8. Select **Xilinx Tools → Create Zynq Boot Image** again.

9. Click **Browse** next to the *Output Path*. To keep things consistent, browse and select the previously used bootimage directory.

```
C:\Avnet\MicroZed\Applications\MZ_Basic_System\Periph_Test\bootimage
```

10. Change the *Save as type* to ***.mcs**. Type in **`Periph_Test`** for the *File name* and then click **Save**.

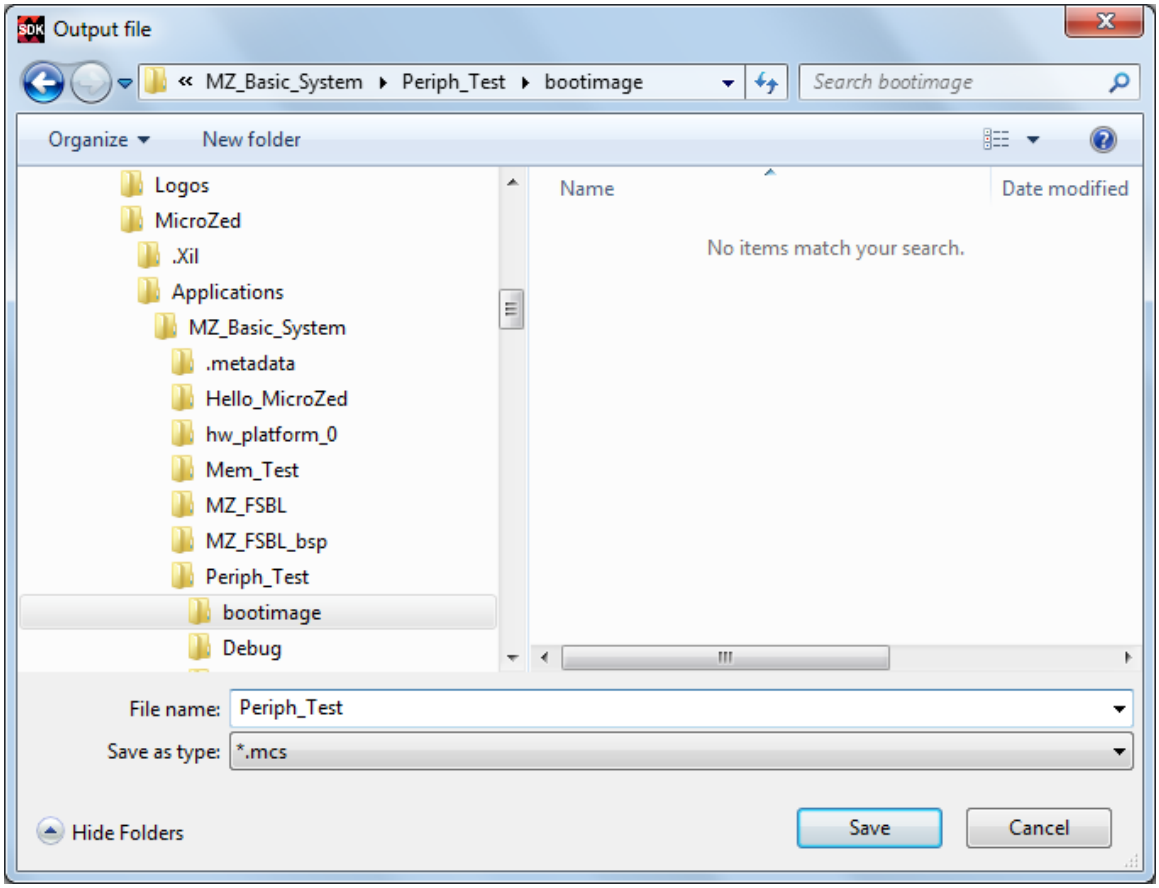


Figure 13 – Set Bootimage Output File to Periph_Test.mcs

The resulting Bootimage dialog is shown below. Notice that we do not need to re-generate the BIF file since it was created when BOOT.bin was generated.

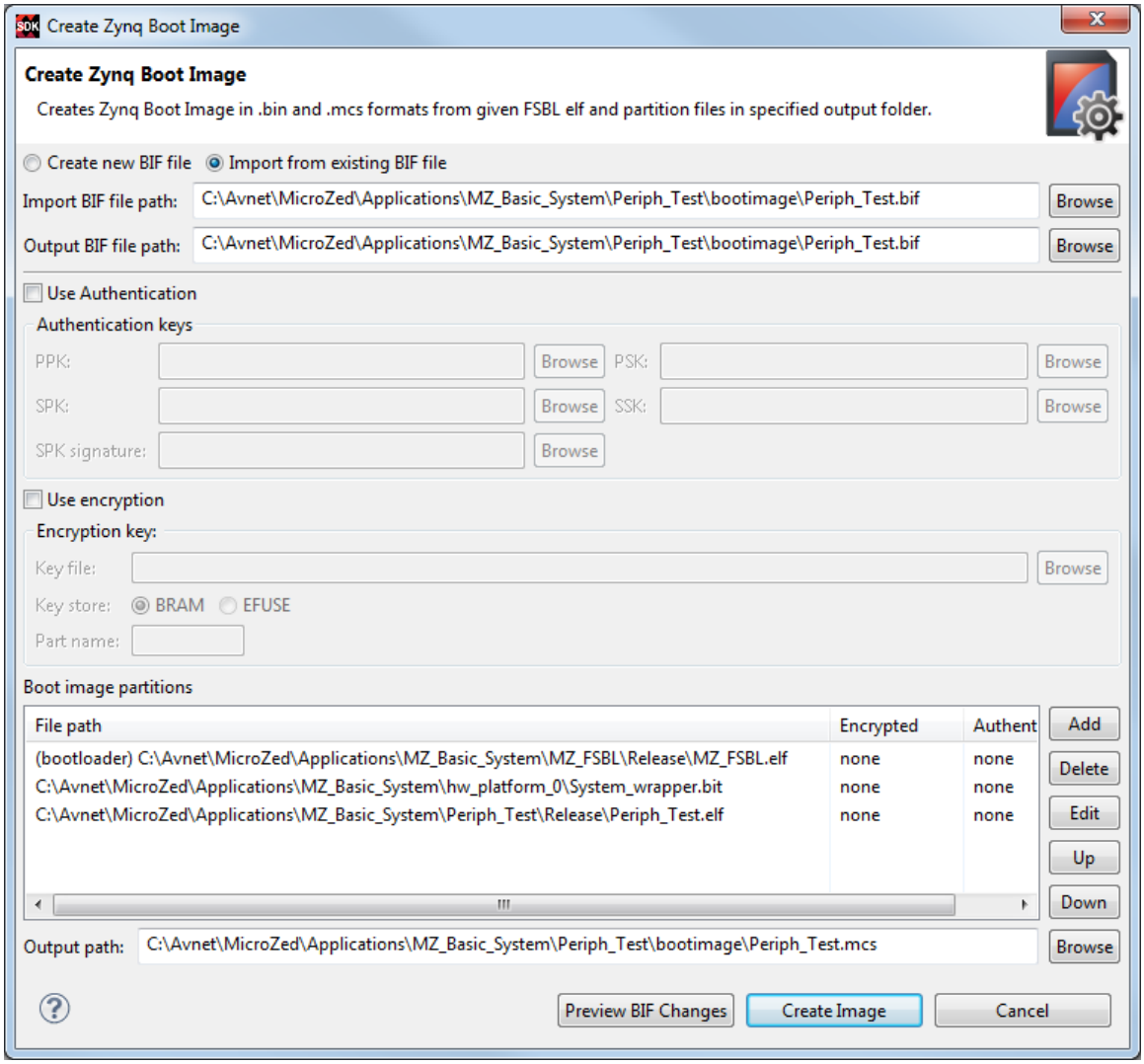


Figure 14 – Creating the QSPI Bootimage

11. Click **Create Image**.

12. Using Windows Explorer, navigate to the application directory, then into `Periph_Test`, then into the newly created **bootimage** directory. Notice that three files have been created: .bif, .bin, and .mcs. These are all the required files to program either the QSPI or SD Card.

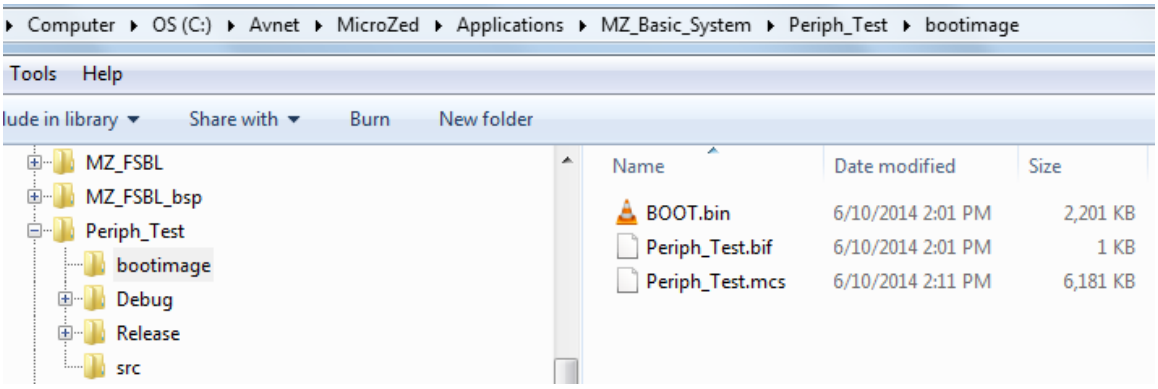


Figure 15 - bootimage Directory

Experiment 3: Write and boot from QSPI

First, we will program the QSPI with the MCS file from within SDK.

1. Unplug the USB-UART cable from MicroZed to power it down.
2. Put MicroZed in Cascaded JTAG mode by setting JP1-3 each to the 1-2 position.

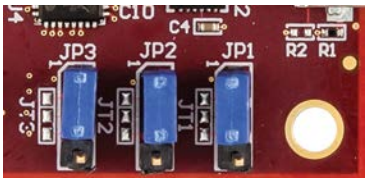


Figure 16 – Cascaded JTAG Boot Mode

3. Attach the JTAG cable.
4. Power on the MicroZed by plugging in the USB-UART cable.
5. In SDK, select **Xilinx Tools → Program Flash**.
6. Click the **Browse** button, and browse to the MCS image for the application you wish to program. Select it and click **Open**. You can selectively enable the *Blank check after erase* and/or the *Verify after flash* functions although each of these will add time to the procedure. Neither is required to program the Flash.
7. Click **Program**.

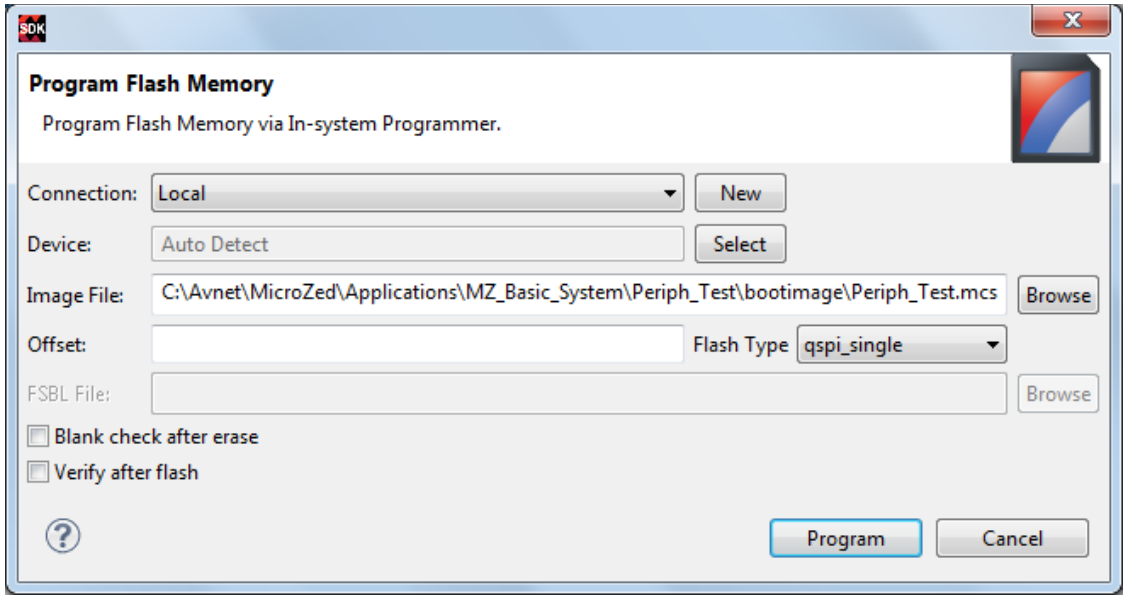


Figure 17 – Program QSPI Flash Memory

8. The operation took approximately 75 seconds on a Win7 PC with a Digilent HS2 JTAG Cable. You should see the following in the Console window.

```
CortexA9 Processor Configuration
-------------------------------------
Version.............................0x00000003
User ID.............................0x00000000
No of PC Breakpoints................6
No of Addr/Data Watchpoints.........4
Processor Reset .... DONE
.................................................

Flash Operation Successful
```

Figure 18 – QSPI Programmed Successfully

9. Unplug the USB-UART cable to turn off power to the board. The JTAG cable may also be unplugged.

10. Set the MicroZed Boot Jumpers to QSPI by moving JP3 to the 2-3 position.



Figure 19 – QSPI Boot Mode

11. Close or disconnect the terminal that may have previously been open on your PC.

12. Plug in the USB-UART cable. The green power good LED and the blue DONE LED should both be lit.

13. Launch a terminal program with the 115200/8/n/1/n settings.

14. Push the RST button. You should see the results in the terminal.

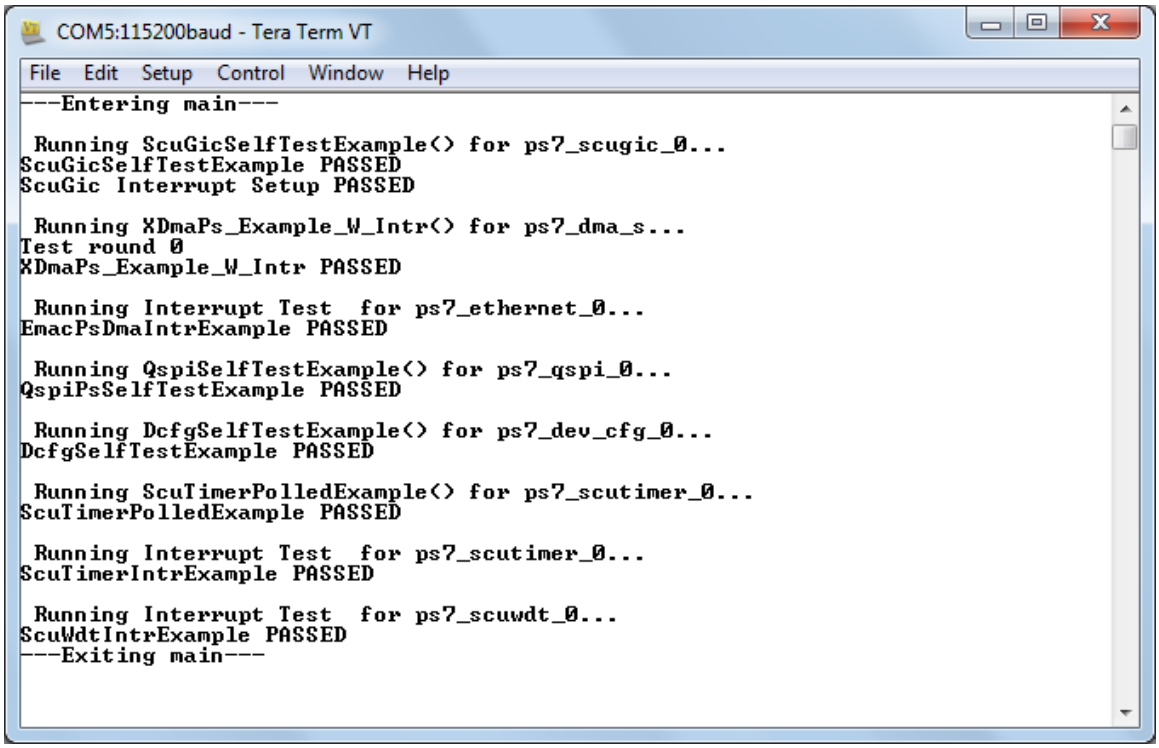


Figure 20 – Results from QSPI boot of Periph_Test

15. Close the terminal and unplug the USB-UART cable from MicroZed.

Experiment 4: Write and boot from the microSD Card

Next, we will prepare the microSD card with the bin file that SDK generated.

1. To program a microSD Card, insert the microSD card into an adapter and plug into your PC. Copy the BOOT.bin image to the microSD Card.

2. Be aware that the name "boot.bin" is essential. If you changed the name of the .bin file, rename it on the microSD card to **`boot.bin`** or **`BOOT.bin`**. THIS IS CRITICAL!

3. Eject the microSD card from the PC.

4. Insert the microSD card into the MicroZed slot.

5. Set the MicroZed Boot Jumpers to SD Card, with both JP2 and JP3 in the 2-3 position.

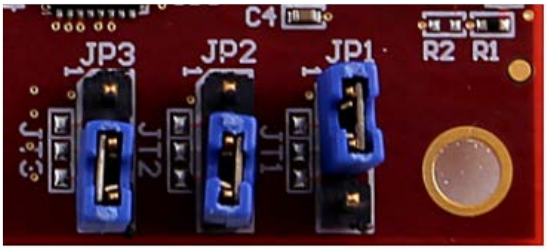


Figure 21 – SD Card Boot Mode

6. Close or disconnect the terminal that may have previously been open on your PC.

7. Plug in the USB-UART cable.

8. Launch a terminal program with the 115200/8/n/1/n settings.

9. Push the RST button. You should see the same results in the terminal that were shown during the QSPI test.

Revision History

Date	Version	Revision
26 Aug 2013	2013_2.01	Initial Avnet release for Vivado 2013.2
06 Sep 2013	2013_2.02	Switching to Release build rather than Debug
10 Jun 2014	2014_1.01	Update to 2014.1. Added instructions that .bin and .mcs must each be generated individually now.
11 Jun 2014	2014_2.01	Update to 2014.2. Bitstream must manually be added to the Boot Image dialog now.