

# Developing Zynq Software with Xilinx SDK

## Lab 6

### First Stage Boot Loader (FSBL)



November 2013  
Version 03

## Lab 6 Overview

In Lab 5, we used the SDK JTAG connection and a TCL script to initialize the ARM processor registers. This was convenient and worked well for debugging. However, a real product is not going to be tethered to a JTAG connection. We need a method to initialize the ARM in an embedded fashion. This is done with code called the First Stage Boot Loader or FSBL.

Xilinx SDK includes a template to automatically create the FSBL for us. By interpreting the provided hardware platform, SDK will generate a fully functional FSBL for us. This is one of the many useful things that SDK automates to make your job as a software developer a bit easier.

## Lab 6 Objectives

When you have completed Lab 6, you will know:

- How to generate the FSBL
- What the FSBL application includes
- How to recognize some of the initialization sequences

## Experiment 1: Generate the FSBL

Similar to the flow for creating the Test applications, SDK is used to generate a First Stage Bootloader application using a template.

### Experiment 1 General Instruction:

Create a new software application. Apply the FSBL template. Note the application size and target memory. Change to the Release configuration and note size.

### Experiment 1 Step-by-Step Instructions:

1. In SDK, select **File** → **New** → **Application Project**.
2. Name it **zynq\_fsbl\_0** and use the existing BSP. Click **Next >**.

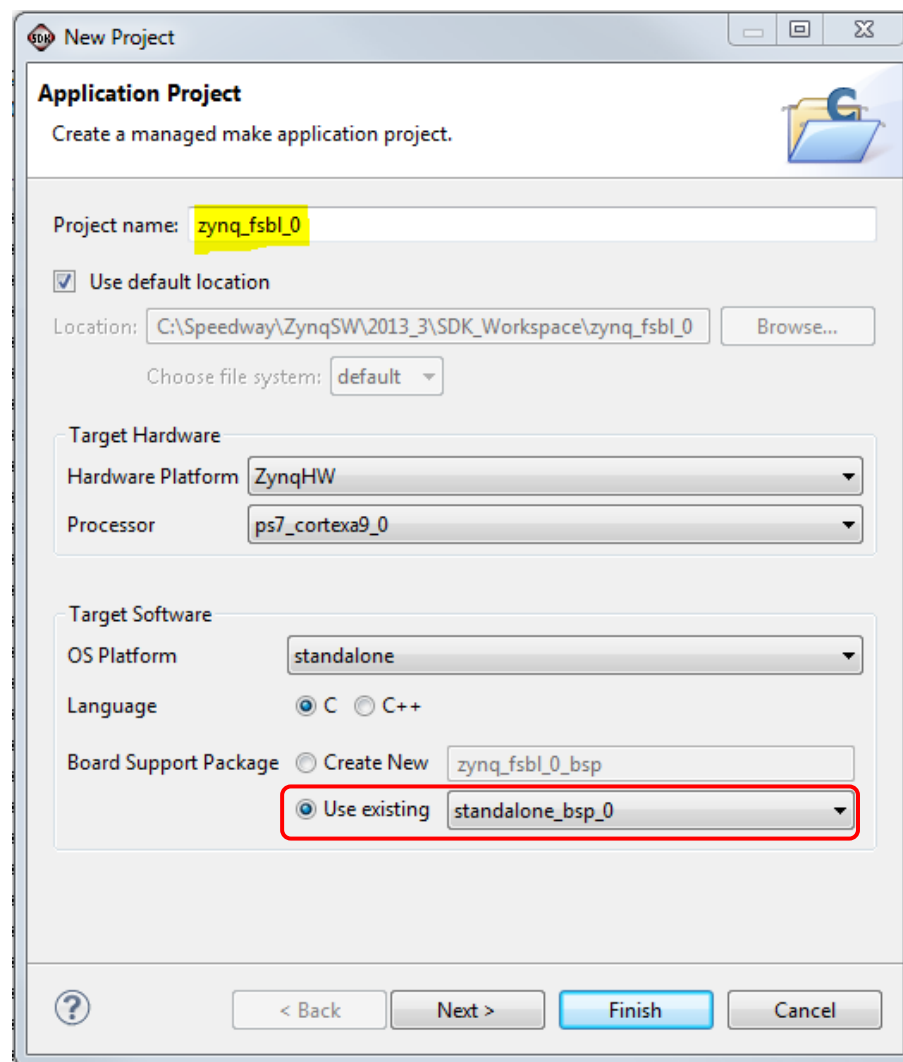
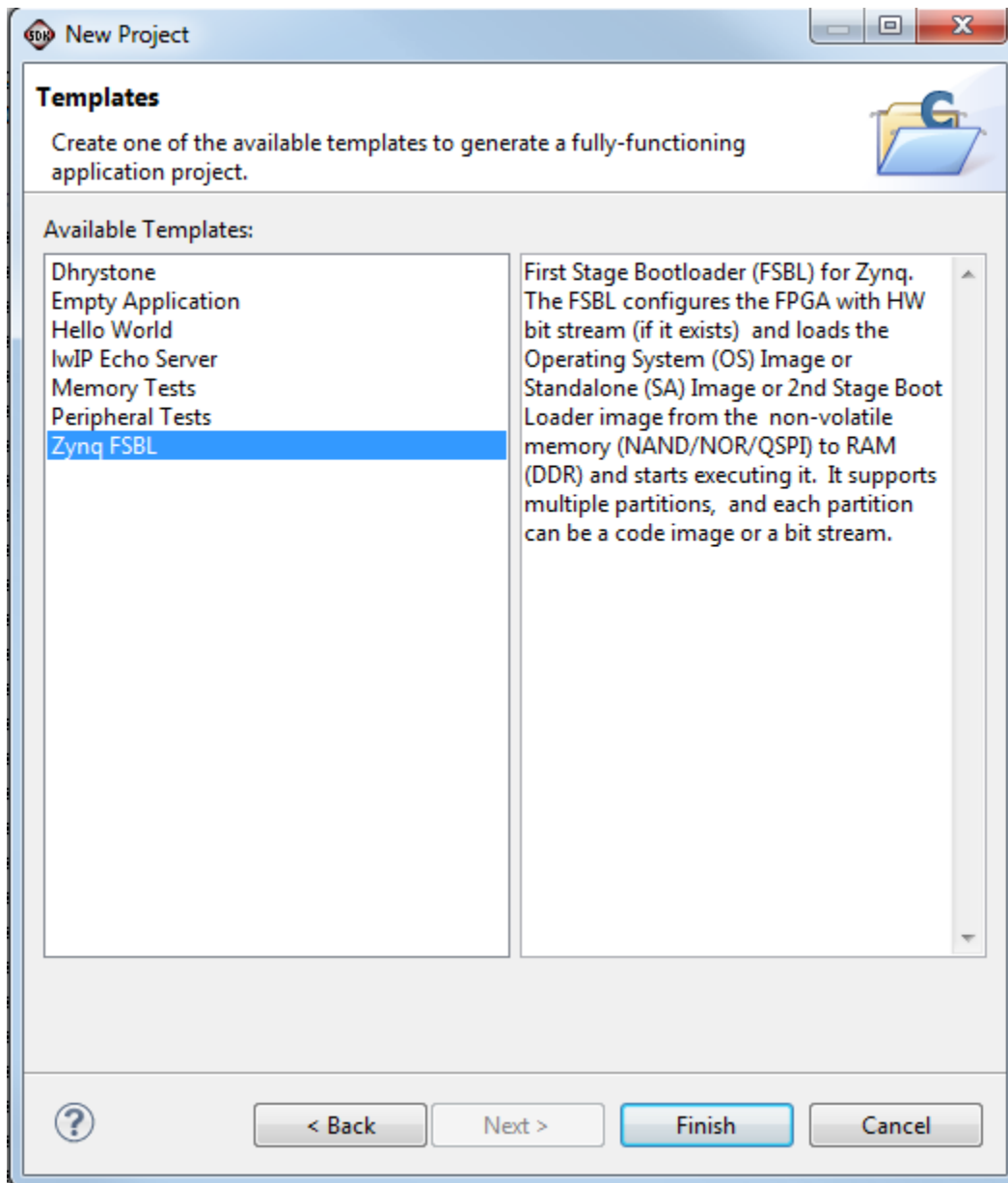


Figure 1 – FSBL Application

3. Select **Zynq FSBL** then click **Finish**.



**Figure 2 – Use the Zynq FSBL Template**

4. Notice this message at the top of the console (emphasis added):

Build of configuration **Debug** for project zynq\_fsbl\_0

5. Use the console output (or file zynq\_fsbl\_0.elf.size) to determine the Debug configuration build size. Use the linker script (lscript.ld) to determine the target memory. Note that the *Summary* tab for lscript.ld does not display the settings accurately for this design, so use the *Source* tab instead.

## Questions:

**Answer the following questions:**

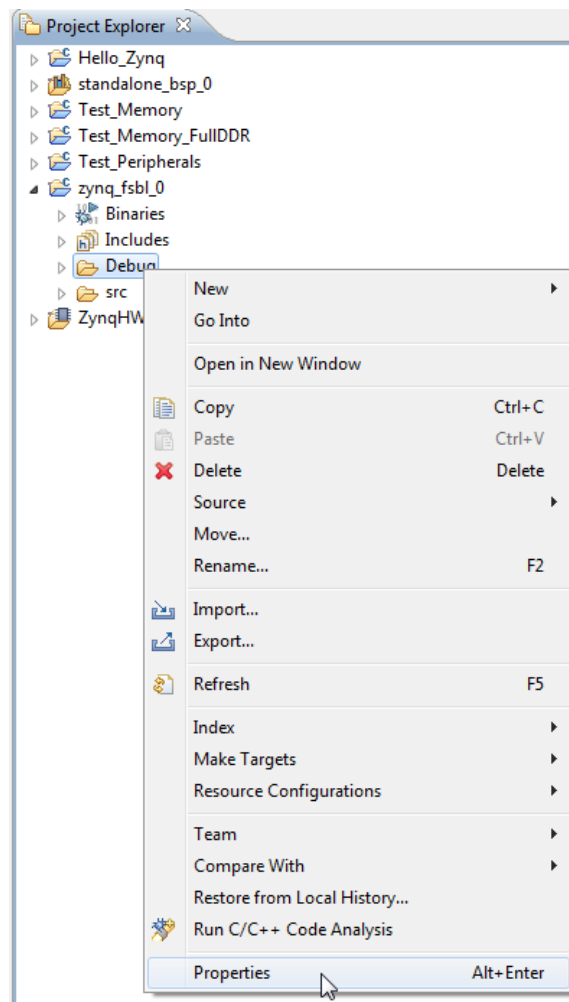
- What is the size of the FSBL application with the Debug configuration?

---

- What is the target memory for the FSBL?

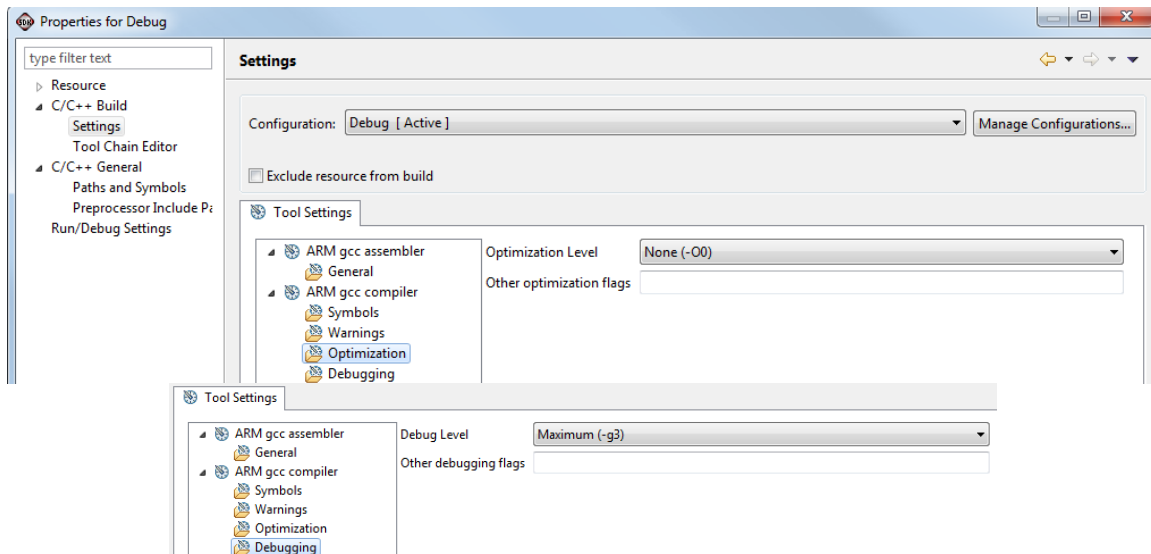
---

6. You can see what the compiler settings for the Debug configuration are by right-clicking on the **Debug** folder underneath the **zynq\_fsbl\_0** application and selecting **Properties**.



**Figure 3 – See Properties for Debug Configuration**

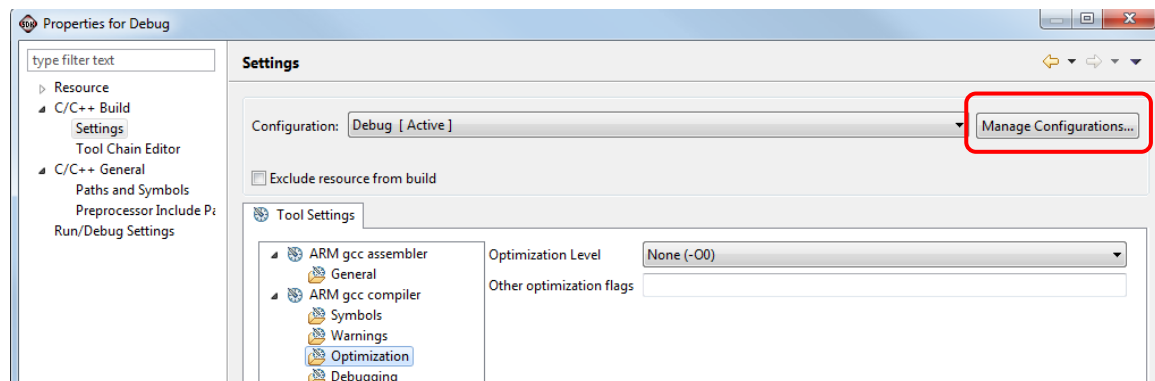
- Expand *C/C++ Build* and select **Settings**. Then under *Tool Settings*, expand *ARM gcc compiler* and look at **Optimization** and **Debugging**. Notice that the Optimization Level is at None and Debug Level is at Maximum.



**Figure 4 – Debug Configuration Optimization and Debugging**

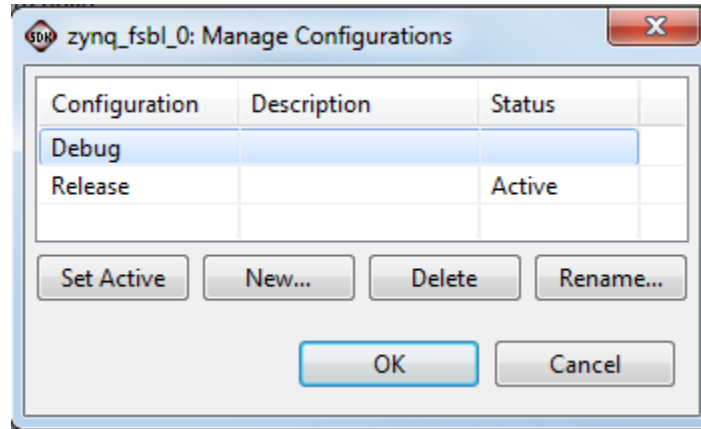
If we were going to debug this application, then these settings are perfect. However, the whole point of creating the FSBL is to move to an embedded delivery. Once we are no longer debugging, it is to our advantage to optimize the code compilation and remove the debug symbols.

- Click the **Manage Configurations** button. You'll find this button in the upper right corner of the *Properties for Debug* dialog



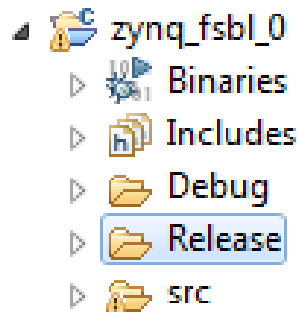
**Figure 5 – Manage Configurations Button**

9. 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**.
11. Click **Project → Build All**.
12. Notice that a **Release** sub-folder is now present in the zynq\_fsbl\_0 application project.



**Figure 7 – Release Configuration Added and Built**

**Question:**

**Answer the following question:**

- What is the size of the FSBL application with the Release configuration?

---

## Experiment 2: Investigate the FSBL

Now that we have the FSBL generated and built, it is worth taking a minute to examine what we have!

### Experiment 2 General Instruction:

Examine the FSBL sources. Open ps7\_init.c and review the code.

### Experiment 2 Step-by-Step Instructions:

1. Expand **zynq\_fsbl\_0** → **src**. Notice the various .c and .h files that SDK pulled together for the FSBL.

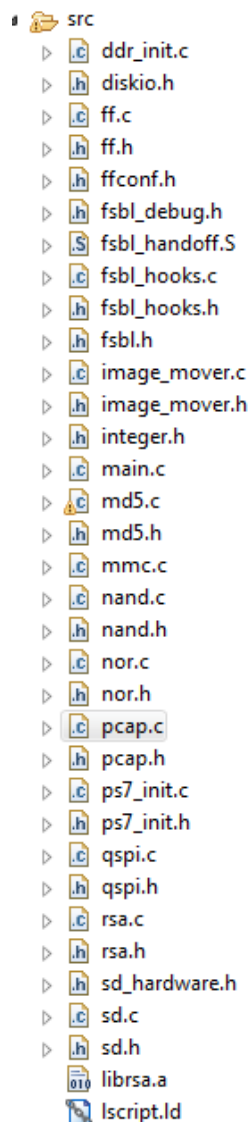


Figure 8 – FSBL Sources



2. Figuring out where function main is could be a challenge. We'll take advantage of the search function to find it. Select the **src** folder, then from the SDK pull-down menu select **Search** → **C/C++**. Type in 'main' into the *Search string*. Select the radio button for *Selected resources*. Click **Search**.

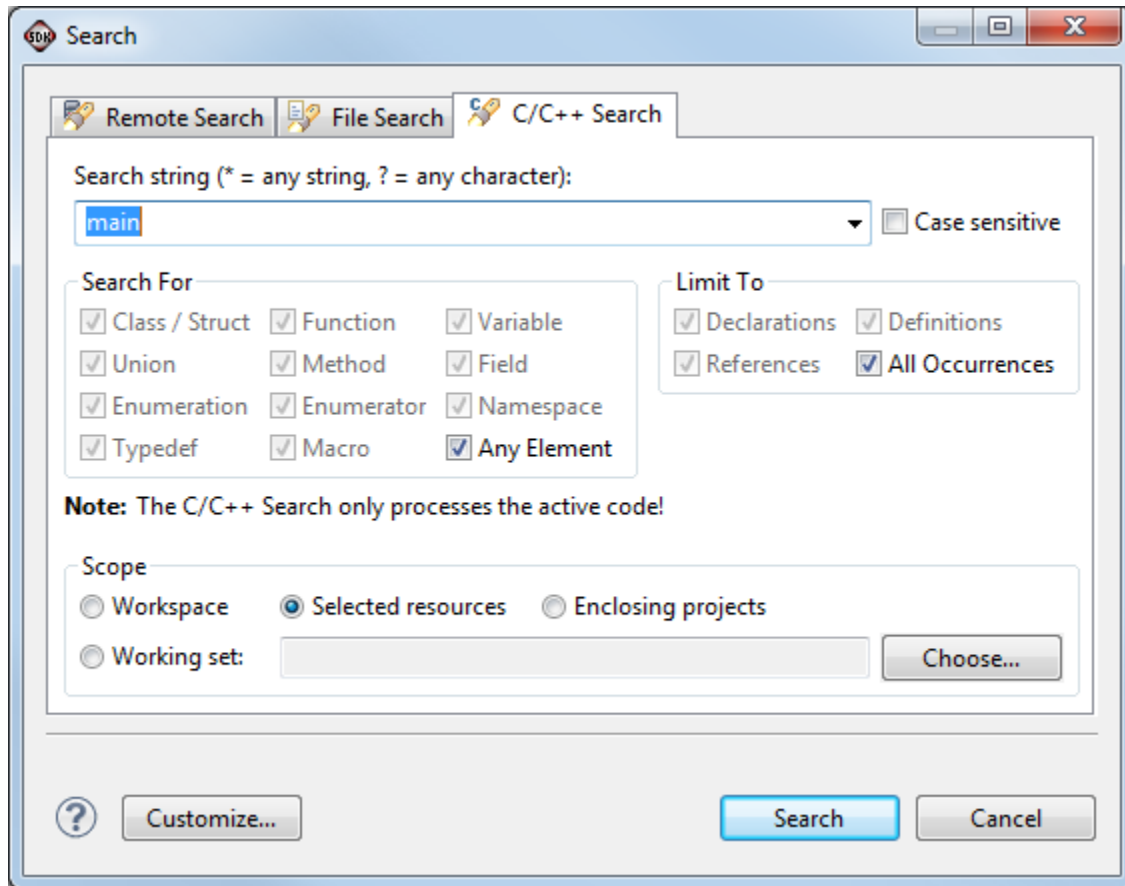


Figure 9 – Search for main

3. A new *Search* tab opens with the results. Notice that `main(void)` is in the file `main.c`.

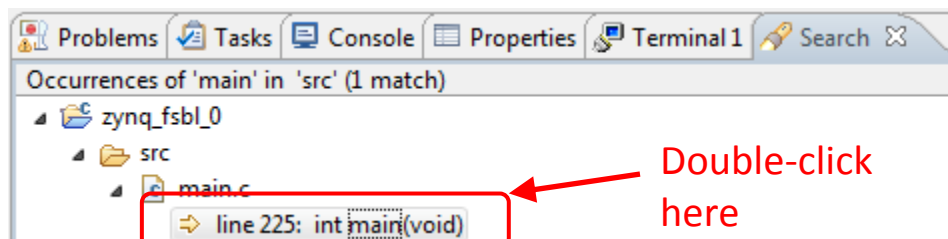


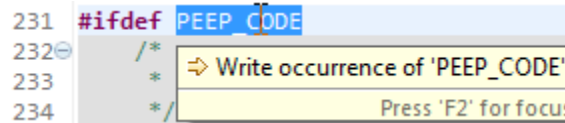
Figure 10 – Search Results

4. In the Search tab, double-click the result to open `main.c` to the `main` function.

5. Scroll down through the code to get an overview of what the code does. Make use of the hover and right-click features in the editor to determine definitions and also to jump to other function declarations.

If you didn't turn on line numbers previously, go to **Window → Preferences**, then **General → Editors → Test Editors** and check the box for **Show line numbers**.

- a. PEEP\_CODE is not defined, so skip lines 232-235



- b. Line 240 executes ps7\_init(). Select it, then right-click and select **Open Declaration**. Click **No**, as we won't change the scalability settings now. It is important to note that the ps7\_init.c file that was just opened is large.
- c. This has placed us on line 12303 of a 12359 line file, so very nearly the bottom. You can see the comments here to see what is going on:
  - Get the silicon version and make a few custom adjustments
  - Initialize the MIO
  - Initialize the PLLs
  - Initialize the clocks
  - Initialize the DDR
  - Configure the peripherals
- d. If you scroll back up to the top of the file, you will see very detailed, highly commented instances of register settings which is what all these initialize functions are performing.
- e. Close the ps7\_init.c file.
- f. Back in main.c, assuming ps7\_init() was successful, the next thing is to Unlock the System Level Control Registers (SLCR).
- g. Scroll down to line 293. Since XPAR\_PS7\_DDR\_0\_S\_AXI\_BASEADDR is defined, the DDR Initialization Check will be run.

```

293 #ifdef XPAR_PS7_DDR_0_S_AXI_BASEADDR
294
295 /* Definitions for peripheral PS7_DDR_0 */
296 * #define XPAR_PS7_DDR_0_S_AXI_BASEADDR 0x00100000
297 */
298

```

- h. The PCAP (Programming Configuration Access Port) then gets set up starting on line 314.
- i. Scroll down to Line 379. Here the boot mode is read and compared to a mask. This will read the MODE jumper settings on your board.
- j. Based on that BootMode setting, the FSBL then jumps to the appropriate Boot Media, which might be QSPI, NAND, NOR, SD or JTAG.
- k. Once the boot media is accessed, a bitstream will be programmed to the PL if it is found, and the application ELF will be copied into the appropriate memory.
- l. Lastly, a handoff address is determined, and then the FSBL hands off control to the application loaded from the Boot Media on Line 571.

### Questions:

**Answer the following questions:**

- In what file was the FSBL main() function?

---

- Which file included all of the ARM register settings? Where did it originate (prior to FSBL generation)?

---



---

## Exploring Further

If you have more time and would like to investigate more...

- Go back to Line 379 of main.c. What is the address of the BOOT\_MODE\_REG? Find this address in the [TRM](#). Which register is this?

This concludes Lab 6.

## 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

### Experiment 1

- *What is the size of the FSBL application with the Debug configuration?*

153,696 bytes

- *What is the target memory for the FSBL?*

ps7\_ram\_0\_S\_AXI\_BASEADDR, which is the 192 KB on-chip RAM

- *What is the size of the FSBL application with the Release configuration?*

140,916 bytes – 12,780 bytes (or 8.3%) smaller

### Experiment 2

- *In what file was the FSBL main() function?*

main.c

- *Which file included all of the ARM register settings? Where did it originate (prior to FSBL generation)?*

ps7\_init.c

The imported hardware platform. Remember Lab 1?