Boot Loading from Flash Memory

Zynq All Programmable SoC

2016.3

Abstract

Booting a system is a critical part of delivering a viable system. Here you will learn how to create and customize a boot image.

Objectives

After completing this lab, you will be able to:

- Create a First Stage Bootloader (FSBL) that works with a custom boot image targeting the QSPI
- Load the image into the QSPI and observe its execution

Introduction

This lab illustrates the steps involved in booting an application from QSPI Flash.

You will be provided with an existing hardware design running on the Zynq® All Programmable SoC PS's Cortex™-A9 processor and a MicroBlaze™ processor. The presence of these two processors will highlight how code from a single Flash device can be configured for multiple processors.

Note: The details of this design can be found in the topic cluster named "Sharing Resources". The important element to remember is that there are multiple processors in this design: a Cortex-A9 and a MicroBlaze processor. Regardless of whether you are using multiple MicroBlaze processors embedded in an FPGA, multiple processors in a PS, or a mix of the two, the basic process remains the same.

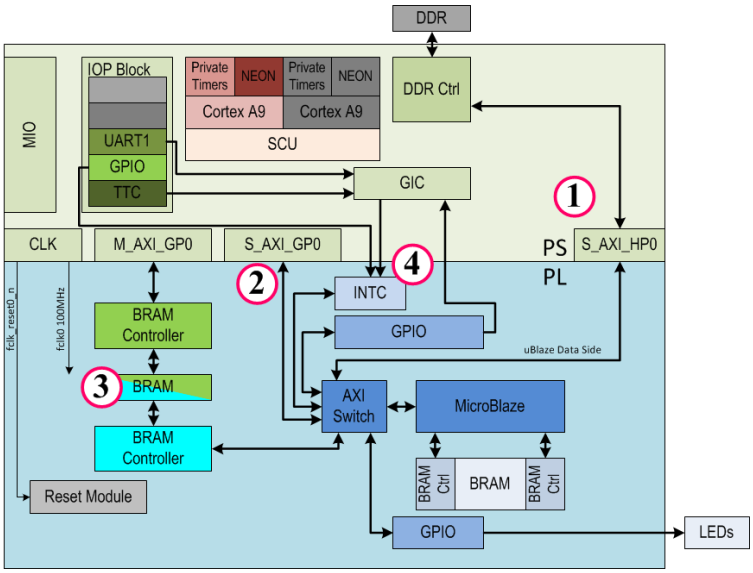


Figure 10-1: Hardware Block Diagram

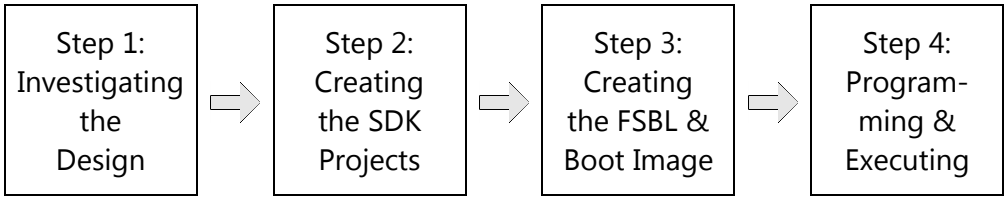
The applications for the processors are typically stored in non-volatile memory (such as QSPI) and is executed from a faster, volatile, off-chip memory resource like DDR or block RAM.

You have been provided with a script to create the hardware project, the pre-generated HDF files as well as the source code and a script to create the SDK projects required to build the boot image.

To simplify development of the boot image, Xilinx SDK provides a Create Boot Image utility. You will use the bootGen tool (available in SDK) to create a bootable image that you will then use to boot the system and verify that the applications have launched.

The applications for this lab are very simple. Both processors initialize themselves then enter a loop where they periodically emit a 'Z' (from the Cortex-A9 processor) or an 'M' (from the MicroBlaze processor). This is only to illustrate that both processors are running their intended code.

General Flow



Investigating the Hardware Design [2016.3] Step 1

A Tcl script has been provided for you to quickly build the hardware project. You will begin by launching the Vivado® Design Suite and running the provided Tcl script to assemble the project.

There are a number of ways to launch the Vivado Design Suite. The two most popular mechanisms are shown here.

1-1. Launch the Vivado Design Suite.

This can be done in two standard ways, use your preferred method.

1-1-1. Select **Start** > **All Programs** > **Xilinx Design Tools** > **Vivado 2016.3** > **Vivado 2016.3**.

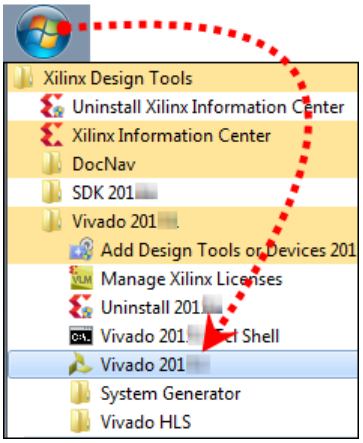


Figure 10-2: Launching the Vivado Design Suite from the Start Menu

-- OR --

Double-click the **Vivado Design Suite** shortcut icon () on the desktop.

The Vivado Design Suite opens to the Welcome window. From the Welcome window you can create a new project, open an existing project, or enter Tcl commands directly into the Vivado Design Suite as well as access documentation and examples.

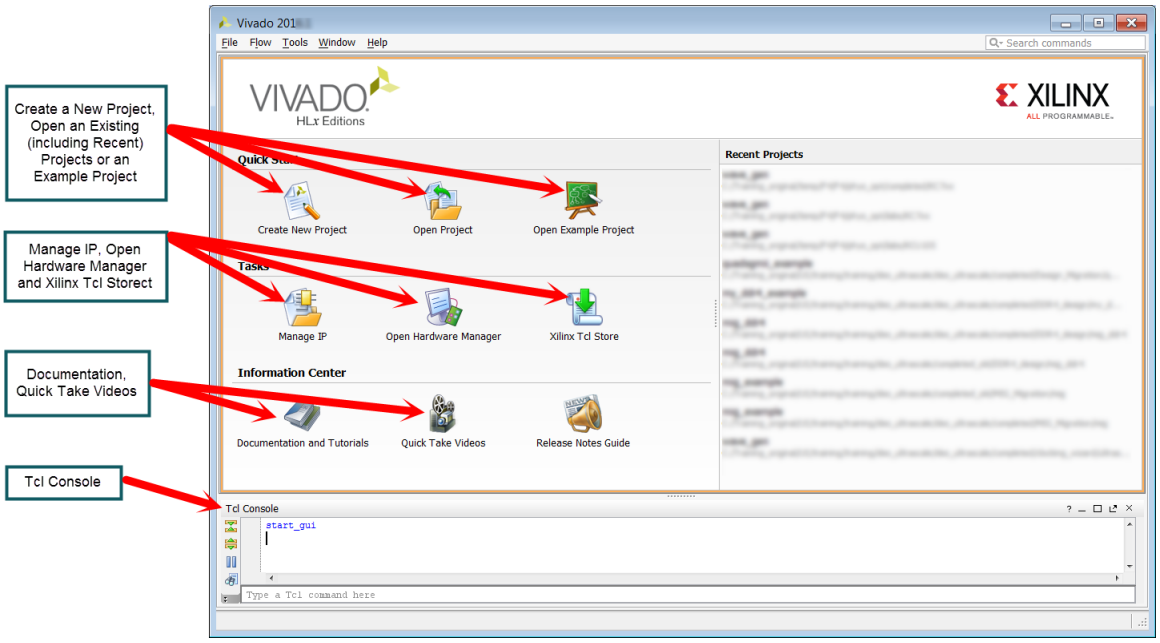


Figure 10-3: Vivado Design Suite Welcome Screen

You will now load the provided Tcl script and have it build the project.

The Vivado Design Suite offers both GUI and scripted control. Scripted control takes the form of Tcl commands. These Tcl commands can be entered directly into the tool one at a time, or an entire Tcl script can be loaded and executed.

1-2. Run a Tcl script.

1-2-1. Locate the Tcl command line entry.

The command line entry can be found either on the Welcome page prior to a project being opened, or once a project has been opened.

From the Welcome screen:

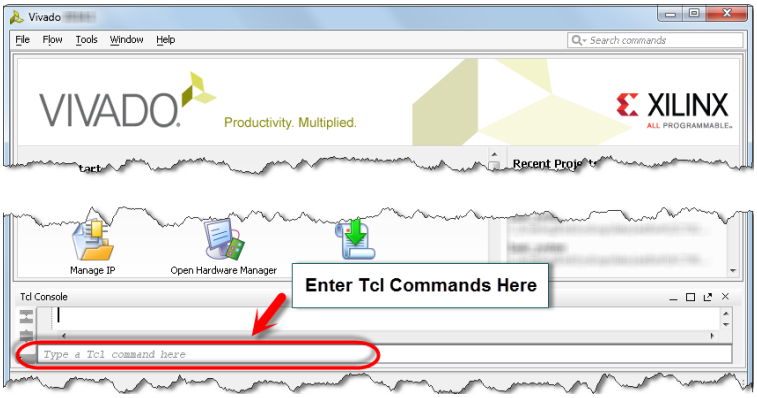


Figure 10-4: Accessing the Tcl Console from the Getting Started Page

From an opened project:

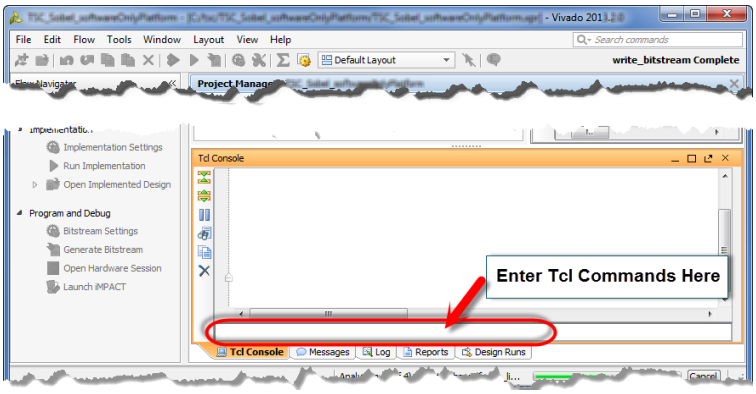


Figure 10-5: Entering Commands into the Tcl Console from an Open Project

The default directory for the Tcl environment is nested within the Xilinx installation directory. This placement, however, is often disadvantageous. In most cases, you will want to navigate to a more useful path. To do this, use the `cd` command to change directory to the user directory.

1-2-2. Change the current working directory to where the Tcl script is located by entering:

```
cd c:/training/bootLoading/support
```

Remember that the Tcl environment is based on Linux and requires the '/' character to delimit hierarchical paths.

1-2-3. Verify that you are now where you want to be by entering the following into the Tcl command line:

```
pwd
```

The current working directory is displayed. If you are not where you want to be, use the `cd` command to change to `c:/training/bootLoading/support`.

1-2-4. Enter the following Tcl command:

```
source bootLoading_completer.tcl
```

The Tcl script is run as though you typed each command included in the Tcl script into the Tcl command line. You can follow the execution of the script and monitor for any errors or warnings in the Tcl Console.

The Tcl script is now loaded and you have been invited to begin entering command and proc names into the Tcl command line.

1-3. Build the project.

1-3-1. Enter the following command to specify the board type, using either ZC702 or Zed as the argument to the `use` proc:

```
use [ZC702 | Zed]
```

You can now specify how the outputs are to be directed.

1-3-2. Specify `base` for using the LEDs on the board:

```
use base
```

1-3-3. Enter the following command to launch the proc that builds the entire project:

```
makeProject
```

The project builds in the Tcl window. When it completes, the Vivado Design Suite GUI will open.

1-4. Investigate the design for the following features:

- **Boot source**
- **Instruction memory for the PS**
- **MicroBlaze processor**
- **Instruction memory for the MicroBlaze processor**
- **Connection between the MicroBlaze processor and the PS**

1-4-1. Click **Open Block Design** in the Flow Navigator window to open the Vivado IP integrator design.

1-4-2. Click the **Show Interface Connections Only** icon to hide clock and reset signals so that you can focus on the AXI connections.

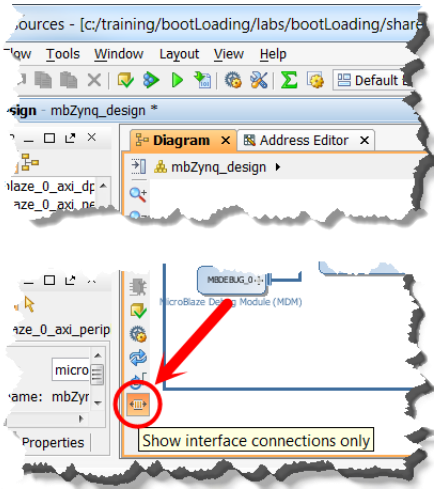


Figure 10-6: Locating the Show Interface Connections Only Icon

Question 1

Where can you find the boot source?

Question 2

What memory is available for the MicroBlaze processor?

Question 3

How are the PS and the MicroBlaze processor connected?

Since no changes should have been made, you can exit the Vivado Design Suite without saving.

1-5. Close the Vivado Design Suite.

1-5-1. Select **File** > **Exit**.

The Exit Vivado dialog box opens.

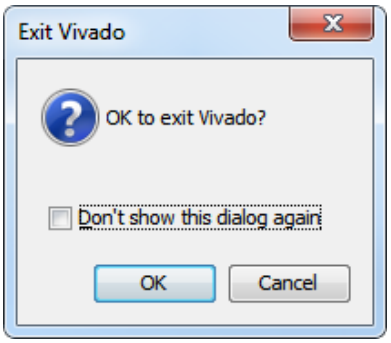


Figure 10-7: Exit Vivado Dialog Box

1-5-2. Click **OK**.

Creating the SDK Projects Step 2

With the hardware investigation now complete, you will move on to briefly investigating the software aspect of the project prior to creating the boot image file.

2-1. Launch the Xilinx Software Command Line Tool (XSCT).

2-1-1. Select **Start** > **All Programs** > **Xilinx Design Tools** > **SDK 2016.3** > **Xilinx Software Command Line Tool 2016.3** to launch the tool.

Alternatively, you can launch the tool from its desktop shortcut, if available.

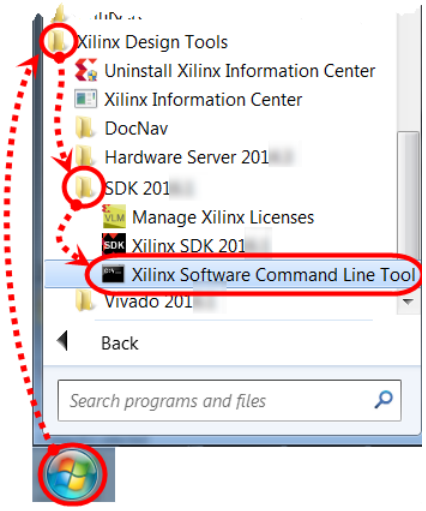


Figure 10-8: Launching XSCT

The Xilinx Software Command Line Tool opens.

Figure 10-9: Xilinx Software Command Line Tool

You can now enter Tcl commands into the tool.

2-2. Build the SDK projects.

2-2-1. From the Xilinx Software Command Line Tool, change directory to the support directory:

```
cd c:/training/bootLoading/support
```

2-2-2. Access the procs needed for the SDK build:

```
source helper.tcl
```

2-2-3. Select the ZC702 or Zed board:

```
use [ ZC702 | Zed ]
```

2-2-4. Create the SDK projects:

```
source SDK_builder.tcl
```

This will create and build the application projects for the Zynq and MicroBlaze processors.

After the script has completed, the Xilinx Software Command Line Tool has no further use and can be closed.

2-3. Launch the SDK tool and set the workspace.

2-3-1. Select **Start** > **All Programs** > **Xilinx Design Tools** > **SDK 2016.3** > **Xilinx SDK 2016.3** to launch the tool.

Alternatively, you can launch the tool from its desktop shortcut, if available.

The Workspace Launcher opens after a moment.

The SDK tool creates a workspace environment that initially only contains a thin structure that tracks tool settings and maintains the SDK tool log file. In SDK, as projects are added, this workspace will update to include hardware projects, BSPs, and your software applications. Workspaces can be switched from within the SDK tool (select **File** > **Switch Workspace**).

If it becomes necessary to move a software application to another location or computer, use the import and export features. Manually copying files is not recommended as workspace files are set to use absolute path names and this will cause the tool to become unstable.

The default location for the SDK software workspace (when launching from within the Vivado® Design Suite) is the root directory of your hardware project; however, a long path name can lead to problems on Windows-based machines. There is no default location for the tool projects. Placing your project at the root level or one hierarchical level below helps keep the path names as short as possible and is recommended.

Many of the Xilinx labs do not follow this guidance as it is important to keep a predictable structure through the various courses and labs. These labs have been tested to ensure that path name lengths do not cause problems.

2-3-2. Enter **C:\training\bootLoading\lab** into the Workspace field or use the Browse button when the Workspace Launcher opens.

Note that when you use the Browse button, you will need to select the **C:\training\bootLoading\lab** directory and click **OK**.

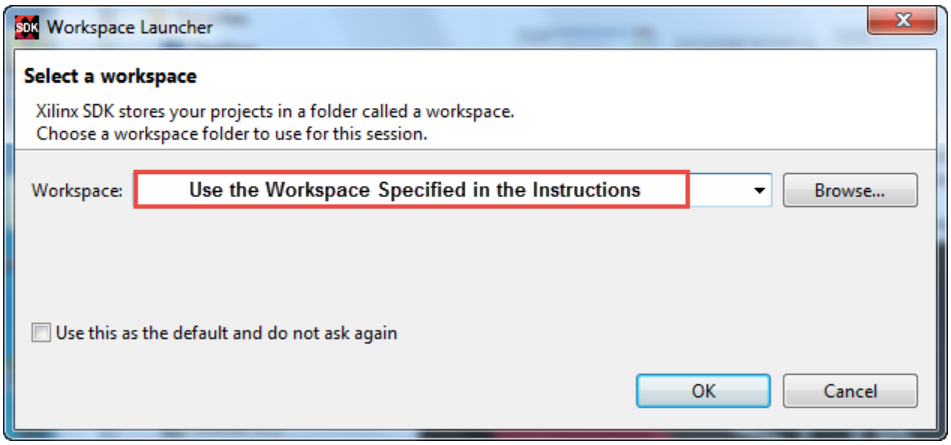


Figure 10-10: Setting Up the Workspace Environment Path

2-3-3. Click **OK** to close the Workspace Launcher dialog box and open the new workspace.

A workspace location and hardware platform are created when the **Export Hardware Design for SDK** command is performed from the Vivado Design Suite (or they can be created manually). While not a requirement, it is a good idea to keep the related files together.

Note that SDK must associate with a hardware system that has been previously exported so that an appropriate software platform or board support package can be built. However, the SDSoC™ development environment can take advantage of available platforms (for ZC702/ZedBoard). The hardware platform can be created for your custom hardware.

Usually, a platform provider builds the platform hardware using the Vivado Design Suite and IP integrator. For more information on platform creation, refer to the "SDSoC Platform Creation" topic cluster.

When the SDK tool is launched on its own, you must manually identify where you want the workspace and create (or import) the necessary hardware description to begin developing an application.

2-3-4. Close the **Welcome** tab if it appears.

This will give you more room to view your project. You may also want to maximize the SDK window, as there will be a lot to see.

The projects that were just built will appear in the Project Explorer view located in the left side of the SDK window.

Question 4

What projects are in this workspace? What are they for?

Question 5

What information from these projects needs to be included in the boot image?

2-4. Open the BSP settings.

2-4-1. Expand the **sharedResourcesQSPI_Zynq_bsp** project so that the *system.mss* file is shown (1).

The *system.mss* file provides an overview of the BSP and supports modification and regeneration of the BSP and its sources.

2-4-2. Double-click the **system.mss** file to open the Board Support Package Project settings dialog box (2).

2-4-3. Click the **Modify this BSP's Settings** button to open a dialog box that contains the BSP Settings (3).

2-4-4. Click **Overview** in the left window pane (4) if the Overview view is not selected.

2-4-5. Select **xilffs** under the Supported Libraries section (5).

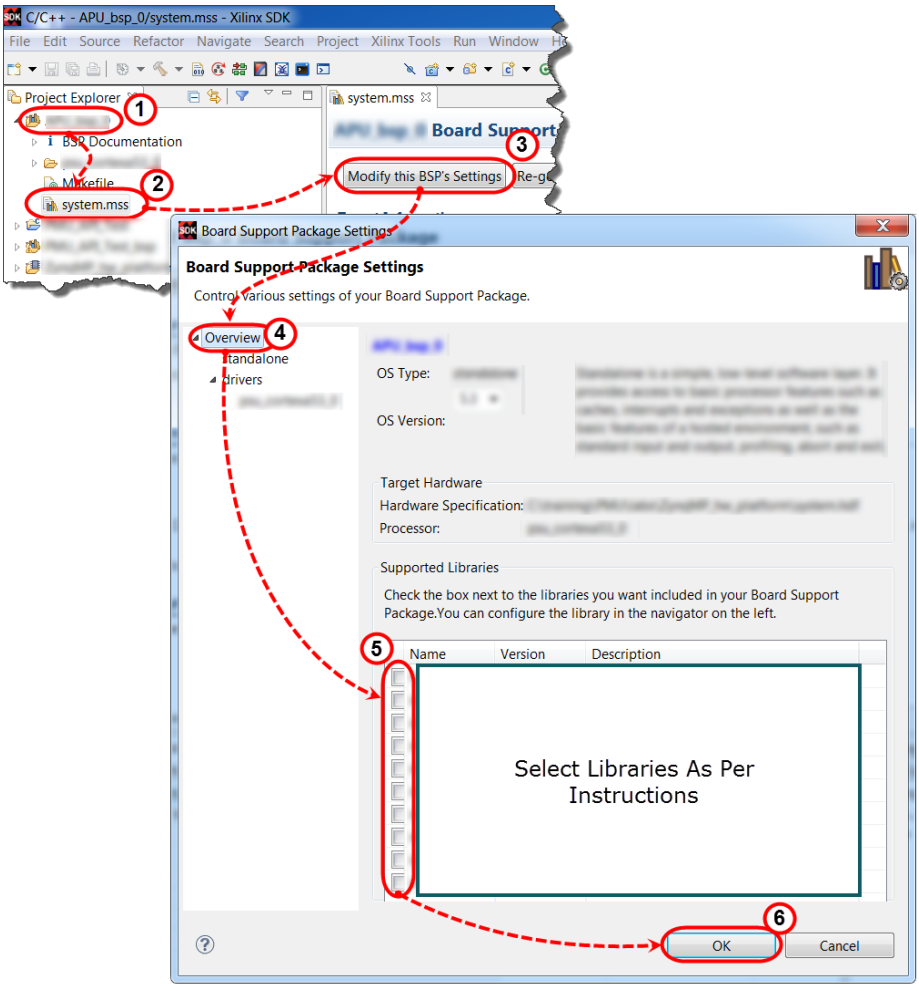


Figure 10-11: Accessing the BSP Customization Dialog Box

2-4-6. Click **OK** to modify the BSP and rebuild the BSP's sources and binaries (6).

Note: The xilffs library is needed by the first stage bootloader application.

Creating a Boot Image File Step 3

You will now create the First Stage Bootloader application that will execute on the Cortex A9_0 processor and take over booting from the bootrom program.

Using the Application Project Wizard is a quick way to set up a C or C++ software application project that targets an existing processor and OS platform (Standalone or Linux). You can automatically generate the board support package (BSP) or select an existing one. Based on the dialog box choices, the appropriate tool chain is selected for pre-processing, compiling, assembling, and linking.

3-1. Create a new C/C++ application project named *Zynq_FSBL_app*. Use the board support package named *Zynq_FSBL_bsp*.

3-1-1. Select **File** (1) > **New** (2) > **Application Project** (3) to open the New Project dialog box.

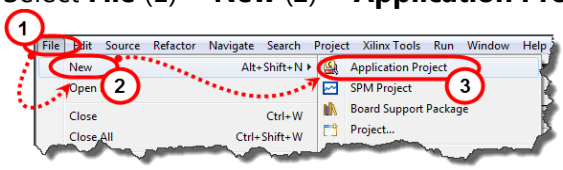


Figure 10-12: Creating an Application Project

3-1-2. Enter **Zynq_FSBL_app** as the project name.

3-1-3. Ensure that you have **sharedResourcesQSPI_hw** selected from the Hardware Platform drop-down list as the SDK tool can manage multiple platforms within a single workspace.

This will populate the Processor drop-down list accordingly.

3-1-4. Ensure that **ps7_cortexa9_0** is selected from the Processor drop-down list.

3-1-5. Ensure that **standalone** is selected from the OS Platform drop-down list.

3-1-6. Select **Use Existing** and choose an existing BSP from the drop-down list if you have already created a BSP for this hardware platform; otherwise, select **Create New** and enter the name for the BSP as **Zynq_FSBL_bsp**.

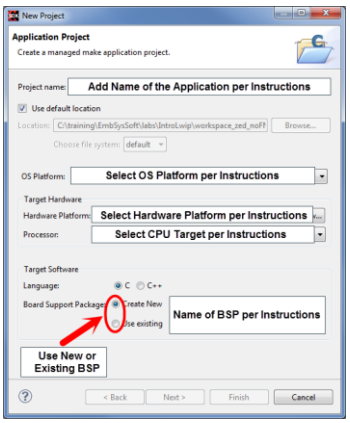


Figure 10-13: Entering Application Project Information

3-1-7. Click **Next** to select the template for this application.

3-1-8. Select **Zynq FSBL** (1).

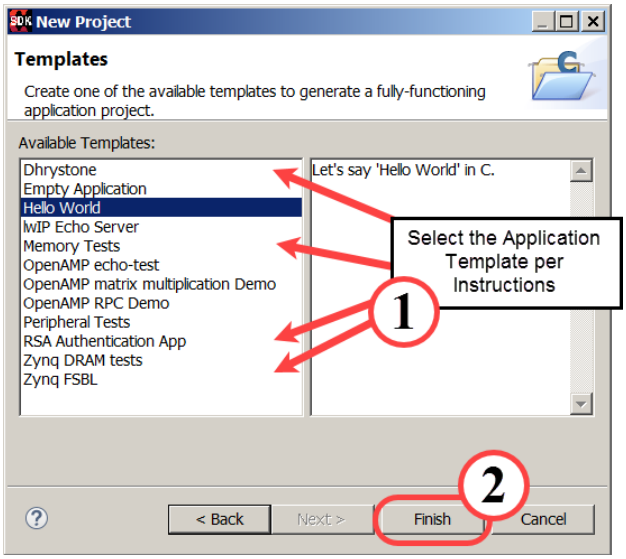


Figure 10-14: Selecting an Application Template (Selection in Figure May Not Match Instructions)

3-1-9. Click **Finish** to create the new project (2).

As the project is created, the new BSP is compiled and any sources from the templates are also compiled. The creation of the new project usually takes less than a minute.

3-2. Build all of the projects.

3-2-1. Force the building of all applications by selecting **Project** > **Build All**.

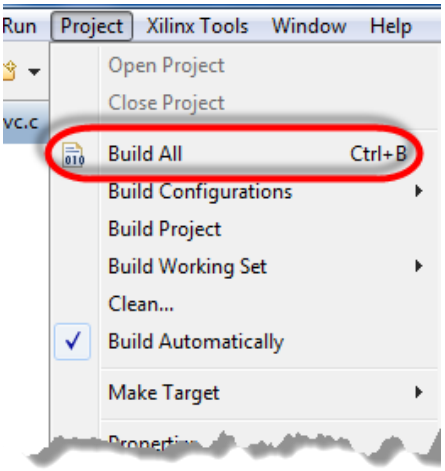


Figure 10-15: Selecting Build All

Note: In 2016.3 the SDK tool does not automatically build the FSBL project upon creation; hence, the need to build all projects.

After the FSBL application has been created, you will create a boot image file that contains the Zynq All Programmable SoC FSBL, the BIT file for programmable logic (PL) configuration, and the software application ELF file.

3-3. Create the boot image.

3-3-1. Right-click the **Zynq_FSBL_app** project from the Project Explorer view to open the context menu.

3-3-2. Select **Create Boot Image** to open the Create Boot Image Wizard.

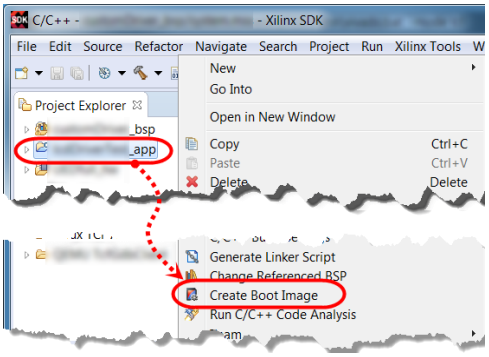


Figure 10-16: Selecting Create Boot Image

The Create Zynq Boot Image dialog box opens.

3-3-3. Make sure that **Create a new BIF file** is selected (1).

The *Output BIF file path* and the *Output path* field will auto-populate (2, 3).

The Boot image partitions section will auto-populate based on the bitstream provided in the hardware specification project and the ELF from the *Zynq_FSBL_app* application project.

Note: If there is a bitstream file in the design, it must be the next entry after the bootloader and any other files should be after the bitstream file.

You will now have to modify the bitstream image and add the remaining file to the boot image partitions.

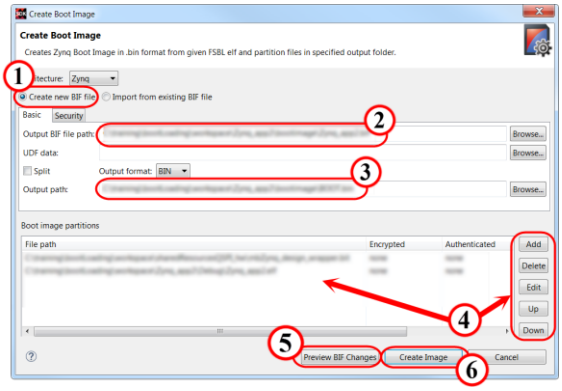


Figure 10-17: Creating the Boot Image File Using the Create Boot Image Wizard

3-3-4. Select the second entry in the *Boot image partitions* box *C:\training\bootLoading\lab\sharedResourcesQSPI_hw\mbZynq_design_wrapper.bit* (4).

3-3-5. Click the **Edit** button (4) to edit the bitstream entry.

3-3-6. Using the **Browse** button next to the *File path* field, browse to *C:\training\bootLoading\lab\sharedResourcesQSPI_hw*.

3-3-7. Double-click **mbZynq_design_wrapper_download.bit** to select and add the file.

3-3-8. Click **OK** to add this modified partition to the boot image.

Note: This bitstream has been modified to preload the MicroBlaze processor with the MicroBlaze processor application. If this were not done, the MicroBlaze processor block RAM would be loaded with all zeros and do nothing. As this is a simple design, the functionality to load the MicroBlaze processor code from the PS is not available. Details of how this bitstream is generated can be seen in the last line of the *SDK_builder.tcl* script.

3-3-9. Click **Add** to begin adding the next ELF file (4).

This will open an *Add new boot image partition* window.

3-3-10. Using the **Browse** button next to the *File path* field, browse to *C:\training\bootLoading\lab\Zynq_app\Debug*.

3-3-11. Double-click **Zynq_app.elf** to select and add the file.

While there are other options including encryption, alignment, offset, etc., you can leave these blank for this lab.

3-3-12. Click **OK** to add this partition to the boot image.

3-3-13. [Optional] Click **Preview BIF Changes** (5).

If you had a previous BIF file, this would compare the old BIF file (left-hand pane) with the newly generated BIF file (right-hand pane).

Question 6

How does the boot image file indicate what gets loaded first and where?

Question 7

How might you add a data file to the boot image?

3-3-14. If you examined the BIF file, click **OK** to close the comparison window.

3-3-15. Click **Create Image** to build the boot image (6).

3-4. Review the generated file.

3-4-1. Expand the **Zynq_FSBL_app** folder using the Project Explorer tab.

Note that the *bootimage* folder has been created.

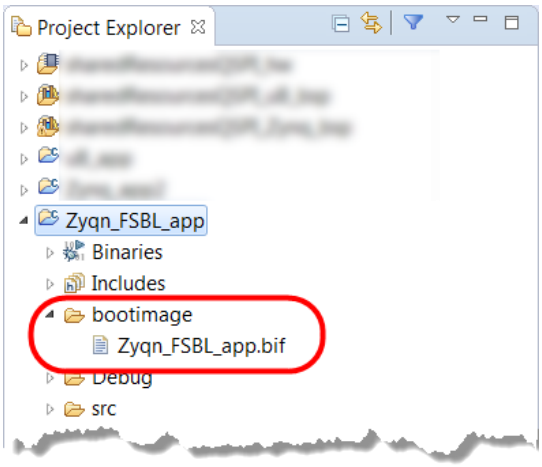


Figure 10-18: bootimage Directory

The *BOOT.bin* image is used for booting the application from Flash and will appear under *bootimage*.

3-4-2. Double-click **Zynq_FSBL_app.bif** and review the contents.

The image shows the partitions and order in which the files will be executed. This is the same information you saw when you clicked the **Preview BIF Changes** button.

Note: The boot image file can also be created manually, the details of which are outside the scope of this lab.

Programming the Flash and Executing the Application from Flash — Step 4

The boot image file created in the previous step will be used to program the Flash. You will then power-cycle the board and boot the system from the Flash device.

For this step, ensure that the board is properly connected to the computer, that there is no SD Card inserted, and that the development board is powered on.

4-1. Select the boot image file to program the Flash memory.

4-1-1. Select **Xilinx Tools** > **Program Flash**.

4-1-2. Click **Browse** to locate the boot image.

4-1-3. Browse to the *C:\training\bootLoading\lab\Zynq_FSBL_app\bootimage* directory.

4-1-4. Select the **BOOT.bin** image file.

4-1-5. Click **Open** to use this file as the source.

For this lab, there is no offset that is required.

4-1-6. Select **qspi_single** from the Flash Type drop-down list.

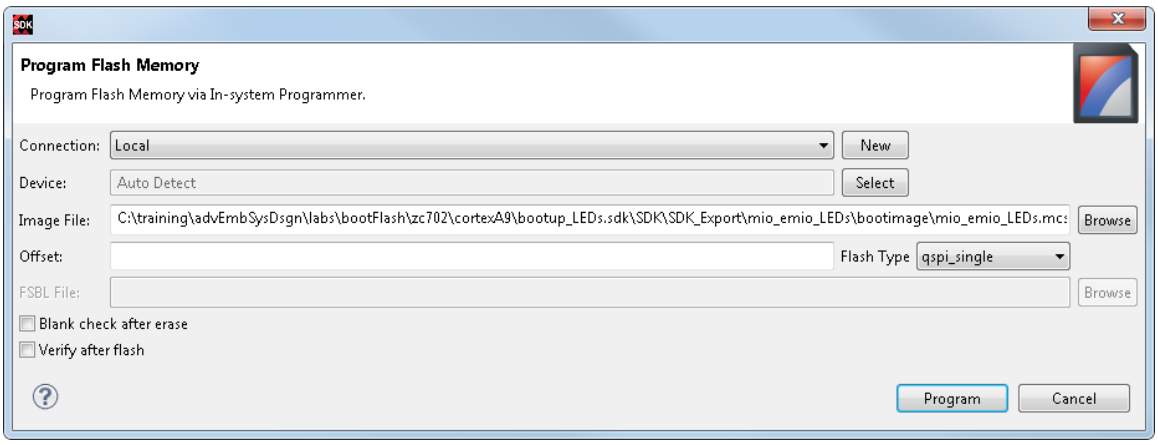


Figure 10-19: Programming the Flash

4-1-7. Click **Program**.

This process will take a few minutes to complete.

You can see the programming status in the Console window.

You will see the following message once the programming is completed:

Flash Operation Successful

4-2. Determine the COM port number for the USB serial port used on the board and configure the related terminal emulator program for that COM port at 115200-N-8-1.

If you do not recall how to perform this task, refer to the "Configuring the SDK Terminal" section under SDK Operations in the *Lab Reference Guide*.

4-3. Set and verify the jumper settings for booting from flash.

4-3-1. Power off the board now that you have programmed the flash, as you need to change the jumper settings for the flash boot.

4-3-2. **ZC702 board users:** Make sure that the jumper settings for booting from flash are set as shown below.

If the jumpers have not been populated, set the switch SW16 to 00010. The jumpers and switch are connected to the same nets.

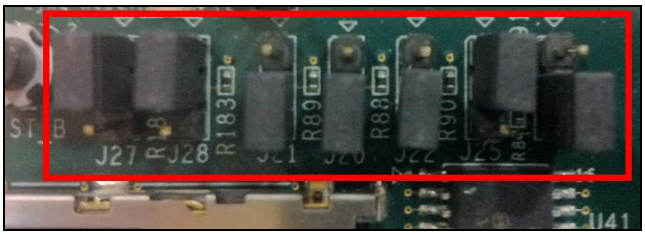


Figure 10-20: Jumper Settings – Flash Boot Mode

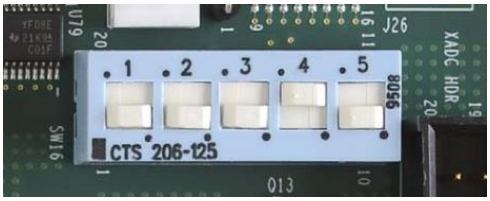


Figure 10-21: SW16 Configured to Boot from QSPI

ZedBoard users: Make sure that the jumper settings for booting from flash are set as shown below.

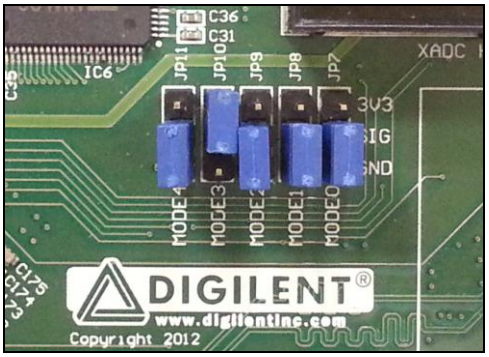


Figure 10-22: Jumper Setting for QSPI Boot (ZedBoard)

4-4. Connect the ZC702 or ZedBoard to your machine.

4-4-1. Make sure that both the UART/USB and programming USB cables are connected to your machine.

4-4-2. Power the board off now if it is powered on.

4-5. Verify the running software application program.

4-5-1. Power on the board.

You should see the DONE LED go ON. The application will now be loaded.

4-5-2. Verify the messages on the serial port console.

You should see a mix of 'Z' (indicating that *Zynq_app* is running) and 'M' (indicating that *uB_app* is running). These may take a up to 10 seconds to appear.

Question 8

What is the ratio of 'Z's to 'M's?

_____

_____

_____

4-6. Exit the SDK tool and power off the board.

4-6-1. Select **File** > **Exit**.

4-6-2. Power off the development board.

Summary

You just reviewed a hardware and software embedded project and created a bootable image that you burned into the target board's Flash memory. You learned how to create various boot image sections and identify the sections as ELF, BIT, data, and bootloader.

This process can be applied to significantly more complex systems, including those with Second Stage Boot Loaders and operating systems.

Answers

1. Where can you find the boot source?

   Since this is a Zynq device, booting is controlled from a series of jumpers that is read when the device is powered up. The board should be configured to boot from the QSPI device. This means that the pins in the MIO should be connected to the QPSI device and should be enabled.

   You can quickly check to see how the MIO is configured by double-clicking the PS block and selecting either Peripheral I/O Pins or MIO Configuration.

2. What memory is available for the MicroBlaze processor?

   The MicroBlaze processor is connected to the HP port of the PS, giving it access to DDR memory, as well as being equipped with its own local block RAM memory for cache and low-latency access.

3. How are the PS and the MicroBlaze processor connected?

   There are two points of connection: an AXI path to the HP port of the PS for access to the DDR and a master AXI connection into the PS so that the MicroBlaze processor can access the IOP block within the PS. This is how the MicroBlaze processor will issue serial characters.

4. What projects are in this workspace? What are they for?

   - *sharedResourcesQSPI_hw:* Hardware project that describes the environment in which the software will operate. This includes the number of types of processors, types and quantities of memory, types and quantities of peripherals, and a memory map placing all devices within the memory scheme.
   - *sharedResourcesQSPI_uB_bsp:* Board support package derived from the hardware project specific to the MicroBlaze processor. This includes device drivers for the peripherals and processors, utilities, and other important support features.
   - *sharedresourcesQSPI_Zynq_bsp:* Board support package derived from the hardware project specific to the Cortex-A9 processor in the PS. As with the MicroBlaze processor's BSP, this includes device drivers, utilities, and other necessary support.
   - *uB_app:* Application that runs on the MicroBlaze processor.
   - *Zynq_app:* Application that runs on the Cortex-A9 processor in the PS.
   - *Zynq_FSBL_app:* First Stage Boot Loader that will run on the Cortex-A9 processor in the PS. The job of this application is to load the ELF files into their appropriate places in memory and load the bitstream in the PL.

5. What information from these projects needs to be included in the boot image?

   Typically the bitstream for the PL is included in the hardware project and will be needed for the boot image. When the applications are compiled, the necessary information is extracted from the BSP, so the BSP projects are not explicitly used. The BSP combined with the application for the Zynq device and MicroBlaze processors produces a pair of ELF files: one for the Cortex-A9 processor and the other for the MicroBlaze processor.

   A mapping of how the two ELF files and bitstream is also required and will be created in the next step as well as a First Stage Boot Loader to extract the various types of files from the NV-RAM device and place them in memory.

6. How does the boot image file indicate what gets loaded first and where?

   The BIF file tags the bootloader with a [bootloader] stamp. This file must be the first ELF file in the Boot Image Partitions list. This was automatically determined by the tools. The partition type could also be specified in the Edit Partition dialog window. The FSBL contains information regarding how to handle BIT files and ELF files.

7. How might you add a data file to the boot image?

   Click **Add** to create a new boot image partition, browse to your data file, and select datafile from the Partition type drop-down list.

8. What is the ratio of 'Z's to 'M's?

   There should be two 'M's appearing for every one 'Z' because *uB_app* emits an 'M' every five seconds and *Zynq_app* emits a 'Z' every 10 seconds.