Configuration and Booting

XUP Lab5 Advanced Embedded System Design on Zynq using Vivado

Introduction

This lab guides you through creating a bootable system capable of booting from the SD card or the QSPI flash memory located on the board. It also demonstrates how different bitstreams can be loaded in the PL section after the board is booted up and the corresponding application can be executed.

Objectives

After completing this lab, you will be able to:

- Create a bootable system capable of booting from the SD card
- Create a bootable system capable of booting from the QSPI flash
- Load the bitstream stored on the SD card or in the QSPI flash memory
- Configure the PL section using the stored bitstream through the PCAP resource
- Execute the corresponding application

Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

Design Description

In this lab, you will design just the PS based embedded system consists of ARM Cortex-A9 processor SoC. The SDIO and QSPI interfaces are included in the base design. The base design will then load the user selected design, consisting of both different hardware and software, and execute it. The following diagram represents the completed design (**Figure 1**).

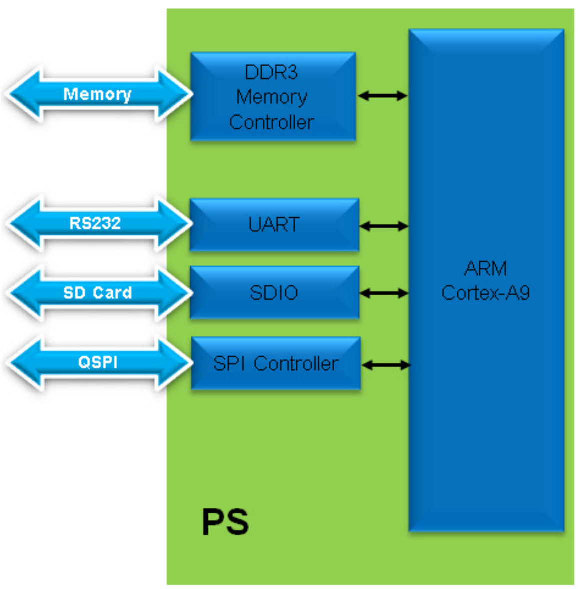
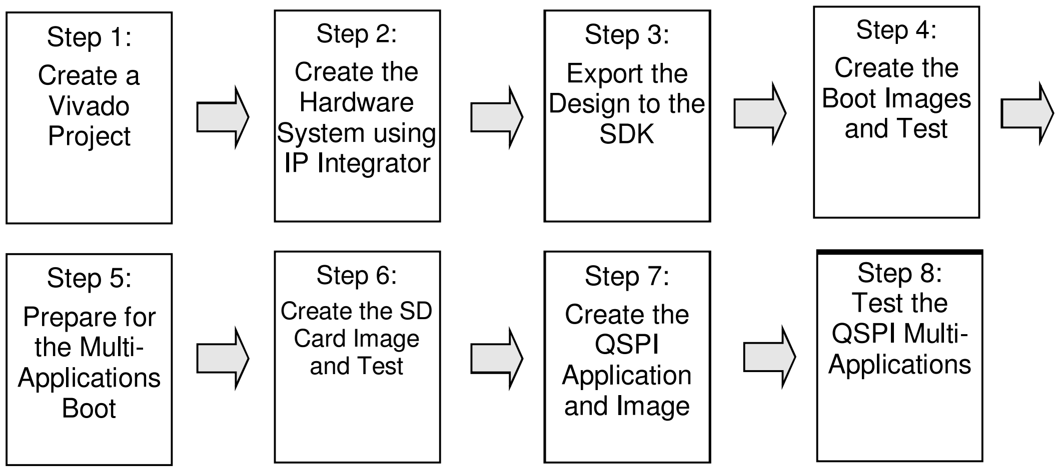


Figure 1 Completed Design

General Flow for this Lab



Create a Vivado Project Step 1

1-1. Launch Vivado and create an empty project, called lab5, and using the VHDL language.

1-1-1. Open Vivado and click **Create New Project** and click **Next**.

1-1-2. Click the Browse button of the *Project Location* field of the **New Project** form, browse to **c:\xup\adv_embedded\labs**, and click **Select**.

1-1-3. Enter **lab5** in the *Project Name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.

1-1-4. Select the **RTL Project** option in the *Project Type* form, and click **Next**.

1-1-5. Select **Verilog** as the *Target Language* in the *Add Sources* form, and click **Next**.

1-1-6. Click **Next** two times.

1-1-7. Click *Boards*, and select the **Zybo** or **ZedBoard** (choose your revision) and click **Next**.

1-1-8. Click **Finish** to create an empty Vivado project.

Creating the Hardware System Using IP Integrator Step 2

2-1. Create a block design in the Vivado project using IP Integrator to generate the ARM Cortex-A9 processor based hardware system.

2-1-1. In the Flow Navigator, click **Create Block Design** under IP Integrator.

2-1-2. Name the block **system** and click **OK**.

2-1-3. Click on *Add IP* in the message at the top of the *Diagram* panel.

2-1-4. Once the IP Catalog is open, type zy into the Search bar, and double click on **ZYNQ7 Processing System** entry to add it to the design.

2-1-5. Click on *Run Block Automation* in the message at the top of the *Diagram* panel. Leave the default option of *Apply Board Preset* checked, and click **OK**.

2-1-6. Double click on the Zynq block to open the *Customization* window.

2-1-7. A block diagram of the Zynq should now be open, showing various configurable blocks of the Processing System.

2-2. Configure the I/O Peripherals block to only have QSPI, UART 1 and SD 0 support.

2-2-1. Click on the *MIO Configuration* panel to open its configuration form.

2-2-2. Expand the *IO Peripherals* on the right.

2-2-3. Uncheck *ENET 0*, *USB 0*, and *GPIO (GPIO MIO)*, leaving *UART 1* and *SD 0* selected.

2-3. Deselect TTC device and M_AXI_GP0 interface. Also de-select FCLK_RESET0_N port and FCLK_CLK0.

2-3-1. In the **MIO Configuration** panel, expand the **Application Processing Unit** and uncheck the **Timer 0**.

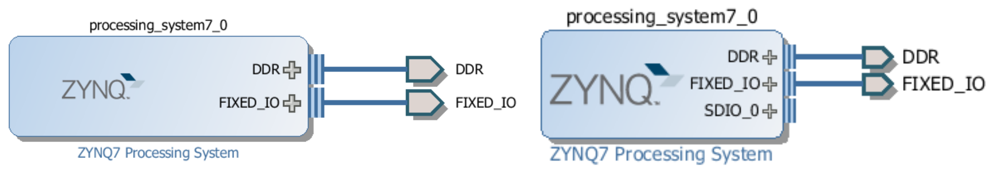
2-3-2. In the **PS-PL Configuration**, expand the *GP Master AXI* Interface, and uncheck the **M AXI GP0 Interface**.

2-3-3. In the **PS-PL Configuration**, expand the *General > Enable Clock Resets*, and uncheck the **FCLK_RESET0_N**.

2-3-4. In the **Clock Configuration**, expand the *PL Fabric Clocks*, and uncheck the **FCLK_CLK0.**

2-3-5. Click **OK**.

The configuration form will close and the block diagram will be updated as shown below.



(a) ZedBoard **(b) Zybo**

Figure 2 ZYNQ7 Processing System configured block

2-3-6. Select **Tools > Validate Design** to run the design rule checker and to make sure that there are no design errors.

2-3-7. Select **File > Save Block Design** to save the design.

Export the Design to the SDK and create the software projects Step 3

3-1. Create the top-level HDL of the embedded system, and generate the bitstream.

3-1-1. In Vivado, select the *Sources tab*, expand the *Design Sources,* right-click the *system.bd* and select **Create HDL Wrapper** and click **OK.**

3-1-2. Right-click on the system.bd, and select **Generate Output Product** and click **Generate**. Click **OK** again when the generation completes.

3-2. Export the design to the SDK and create the Hello World application.

3-2-1. Export the hardware configuration by clicking **File > Export > Export Hardware…**

3-2-2. Do not click the box to *Include Bitstream*, (because only the PS has been configured), then click **OK**

3-2-3. Launch SDK by clicking **File > Launch SDK** and click **OK**

3-2-4. In SDK, select **File** > **New** > **Application Project.**

3-2-5. Enter **hello_world** in the project name field, and leave all other settings as default.

3-2-6. Click **Next** and make sure that the *Hello World* application template is selected, and click **Finish** to generate the application.

3-2-7. Right click on hello_world_bsp and click **Board Support Package Settings**

3-2-8. Tick to include *xilffs* and *xilrsa* click **OK** (This is required for the next step to create the FSBL).

3-3. Create a first stage bootloader (FSBL).

3-3-1. Select **File** > **New > Application Project.**

3-3-2. Enter **zynq_fsbl** as the project name, select the *Use existing* standalone Board Support Package option with **hello_world_bsp**, and click **Next**.

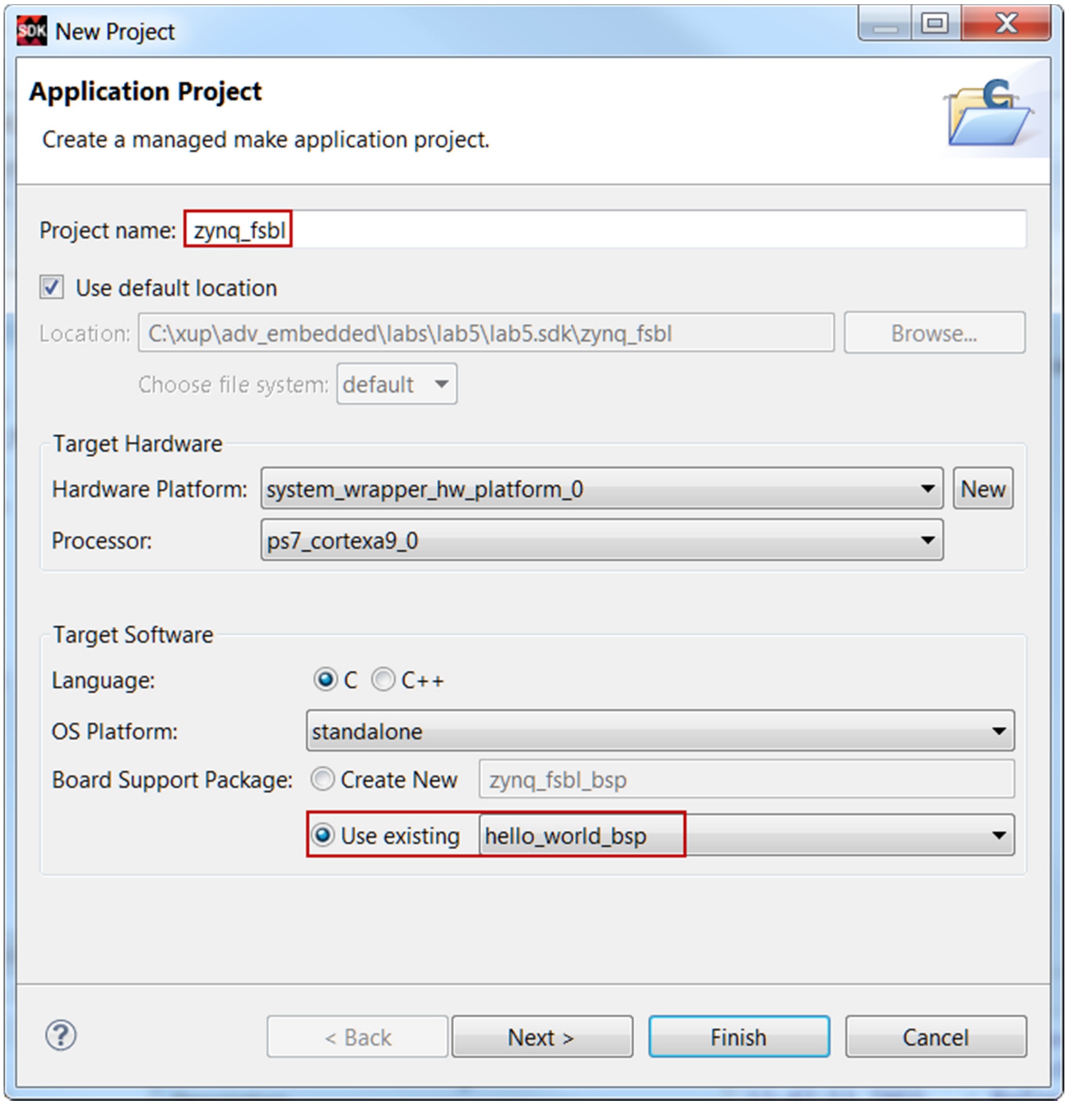


Figure 3 Creating the FSBL Application

3-3-3. Select *Zynq FSBL* in the **Available Templates** pane and click **Finish**.

A zynq_fsbl project will be created which will be used in creating the BOOT.bin file. The BOOT.bin file will be stored on the SD card which will be used to boot the board.

Create the Boot Images and Test Step 4

4-1. Using the Windows Explorer, create a directory called *image* under the lab5 directory. You will create the BOOT.bin file using the FSBL and hello_world.elf files.

4-1-1. Using the Windows Explorer, create a directory under the *lab5* directory and call it **image**.

4-1-2. In the SDK, select **Xilinx Tools > Create Zynq Boot Image**.

Click on the Browse button of the **Output BIF** field, browse to **c:\xup\adv_embedded\labs\lab5\image** and click **Save** (leaving the default name of output.bif)

4-1-3. Click on the **Add** button of the *Boot image partitions,* click the Browse button in the Add Partition form, browse to **C:\xup\adv_embedded\labs\lab5\lab5.sdk\zynq_fsbl\Debug** directory (this is where the FSBL was created), select *zynq_fsbl.elf* and click **Open**.

Note the partition type is bootloader, then click **OK.**

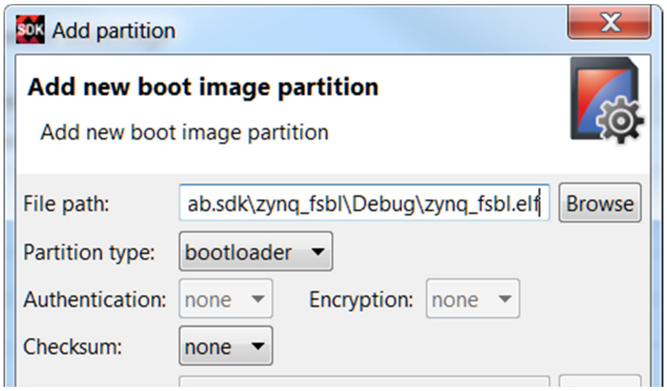


Figure 4 Adding FSBL partition

4-1-4. Click on the **Add** button of the *Boot image partitions* and add the software application, **hello_world.elf**, from the **C:\xup\adv_embedded\labs\lab5\lab5.sdk\hello_world\Debug** directory and click **OK**.

Since there is no hardware in the PL section, there is no bit file for the design. If a .bit file was required, it would be added before adding the application.

4-1-5. Click the **Create Image** button.

The BOOT.bin and the output.bif files will be created in the **lab5\image** directory. We will use the BOOT.bin for the SD card boot up.

4-1-6. Insert a blank SD/MicroSD card (FAT32 formatted) in a Card reader, and using the Windows Explorer, copy the BOOT.bin file from the **image** folder in to the SD/MicroSD card.

4-2. Set the board in SD card boot mode. Test the functionality by starting a Terminal emulator program and powering ON the board.

4-2-1. Set the board in the SD card boot mode (For Zedboard, set the mode pins JP7-JP11 as GND-SIG, GND-SIG, SIG-3V3, SIG-3V3, GND-SIG ([right-to-left), and for Zybo set the JP5 jumper to SD).

4-2-2. Insert the SD/MicroSD card into the board.

4-2-3. Power ON the board.

4-2-4. Connect your PC to the UART port with the provided micro-USB cable, and start a Terminal emulator program setting it to the current COM port and 115200 baud rate.

www.xilinx.com/university
xup@xilinx.com

4-2-5. You should see the **Hello World** message in the terminal emulator window. If you don't see it, then press the PS_RST/PS_SRST (Red button) button on the board.

4-2-6. Once satisfied power OFF the board and remove the SD card.

4-3. Make sure that the board is in the JTAG boot mode. Power ON the board, Program the QSPI using the Flash Writer utility.

4-3-1. Power ON the board.

4-3-2. Select **Xilinx Tools** > **Program Flash**.

4-3-3. In the *Program Flash Memory* form, click the **Browse** button, and browse to the **c:\xup\adv_embedded\labs\lab5\image** directory, select **BOOT.bin** file, and click **Open**.

4-3-4. In the *Offset* field enter **0** as the offset and click the **Program** button.

The QSPI flash will be programmed.

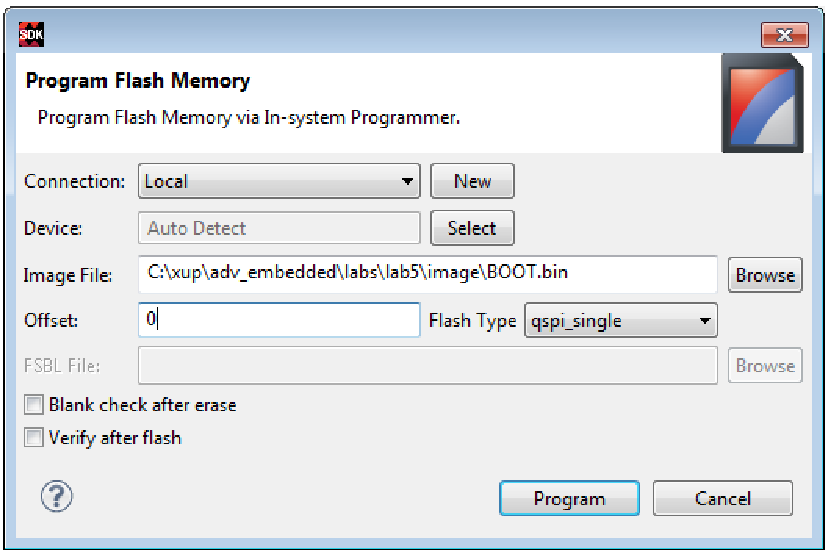


Figure 5 The Program Flash Memory Form

4-4. Power OFF the board and set the board in the QSPI boot mode. Test the functionality by starting a Terminal emulator program and powering ON the board.

4-4-1. Power OFF the board.

4-4-2. Set the board in the QSPI mode (For Zedboard set the mode pins JP7-JP11 as GND-SIG, GND-SIG, GND-SIG, SIG-3V3, GND-SIG ([right-to-left) and for Zybo set JP5 to QSPI).

4-4-3. Power ON the board.

4-4-4. Connect your PC to the UART port with the provided micro-USB cable, and start a Terminal emulator program setting it to the current COM port and a 115200 baud rate.

4-4-5. Press the PS_RST (Red button) button on the board to see the **Hello World** message in the terminal emulator window.

4-4-6. Once satisfied, power OFF the board.

Prepare for the Multi-Applications Boot Step 5

5-1. Create the lab5_sd application using the provided lab5_sd.c and devcfg.c, devcgf.h, load_elf.s files.

5-1-1. Select **File** > **New** > **Application Project.**

5-1-2. Enter **lab5_sd** as the project name, click the *Use existing* option in the *Board Support Package* (BSP) field and select *hello_world_bsp*, and then click **Next.**

5-1-3. Select *Empty Application* in the **Available Templates** pane and click **Finish**.

5-1-4. Select **lab5_sd > src** in the project view, right-click, and select **Import.**

5-1-5. Expand the General folder and double-click on **File system,** and browse to the **c:\xup\adv_embedded\sources\lab5** directory.

5-1-6. Select **devcfg.c, devcfg.h, load_elf.s**, and **lab5_sd.c,** and click **Finish.**

The program won't compile successfully indicating LAB1_BITFILE_LEN and LAB3_BITFILE_LEN are not defined.

5-1-7. Select *lab5 _sd > C/C++ Build Settings*. Click on *Symbols* under the **ARM gcc compiler group**, click the **+** button and enter either **ZED** or **ZYBO** depending on the board. Click OK twice.

The program should compile successfully and generate the lab5_sd.elf file.

> The next two steps 5-2 to 5-5 are optional. They convert the lab1 and lab3 executable files to the required (.bin) format for copying to the SD card later in step 6. The area in memory allocated for each application will be modified so that they do not overlap each other, or the main application. One or both of these steps, (5-2 and 5-3, and 5-4 and 5-5), can be skipped, and the prepared bin files provided in the directory: c:\xup\adv_embedded\sources\lab5\ [zybo | zedboard]\SDCard can be used for copying to the SD card. If the two steps are skipped, proceed to 6-1.

5-2. (OPTIONAL) Start another instance of the SDK program. Open the lab1 project, change the ps7_ddr_0_S_AXI_BASEADDR to 0x00200000 and the Size to 0x1FE00000 in the lscript.ld (linker script) file. Recompile the lab1.c file. Use objcopy command to convert the elf file into the binary file and note the size of the binary file as well as the program entry point (main()).

5-2-1. Start the **SDK** program and browse to the workspace pointing to **c:\xup\adv_embedded\labs\lab1\lab1.sdk**\ and click **OK.**

 www.xilinx.com/university
xup@xilinx.com

5-2-2. Right-click on the **lab1** project, select the *Generate Linker Script* option, change the *code, data, heap, and stack* sections to use the *ps7_ddr_0_AXI_BASEADDR*, and click **Generate**. Click **Yes** to overwrite the linker script.

5-2-3. Expand the **lab1 > src** entry in the Project Explorer, and double-click on the **lscript.ld** to open it.

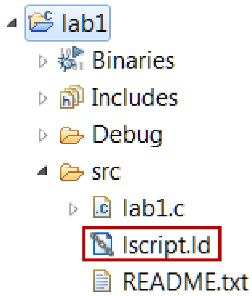


Figure 6 Accessing the linker script to change the base address and the size

5-2-4. In the lscript editor view, change the Base Address of the *ps7_ddr_0_AXI_BASEADDR* from **0x00100000** to **0x00200000**, and the Size from **0x1FF00000** to **0x1FE00000**.

Name	Base Address	Size
ps7_ram_0_S_AXI_BASEADDR	0x00000000	0x00030000
ps7_ddr_0_S_AXI_BASEADDR	0x00200000	0x1FE00000
ps7_ram_1_S_AXI_BASEADDR	0xFFFF0000	0x0000FE00

Figure 7 Changing the Base address and the size

5-2-5. Press **Ctrl-S** to save the change.

The program should compile.

5-2-6. In the SDK of the **lab1** project, select **Xilinx Tools > Launch Shell** to open the shell session.

5-2-7. In the shell window, change the directory to **lab1\Debug** using the **cd** command.

5-2-8. Convert the *lab1.elf* file to *lab1elf.bin* file by typing the following command.

arm-xilinx-eabi-objcopy -O binary lab1.elf lab1elf.bin

5-2-9. Type **ls –l** in the shell window and note the size of the file. In this case, it is **32780**, which is equivalent to 0x800c bytes.

5-2-10. Determine the entry point “main()” of the program using the following command in the shell window.

arm-xilinx-eabi-objdump -S lab1.elf | grep main

It should be in the **0x00200558**.

5-2-11. Close the Shell window.

5-3. Define MULTIBOOT symbol, create Zynq_fsbl application, and change the lab1 BSP reference to zynq_fsbl_bsp. Create the image using the zynq_fsbl.elf, system_wrapper.bit, and lab1.elf files and naming it as lab1.bin in the lab1 directory.

5-3-1. Right-click on the **lab1** entry, select the *C/C++ Build Settings* option.

5-3-2. Select *Symbols* in the left pane under the *ARM gcc compiler* group, click the + button on the right, enter **MULTIBOOT** in open form, click **OK** and click **OK** again.

The application will re-compile as the MULTIBOOT related code is now included.

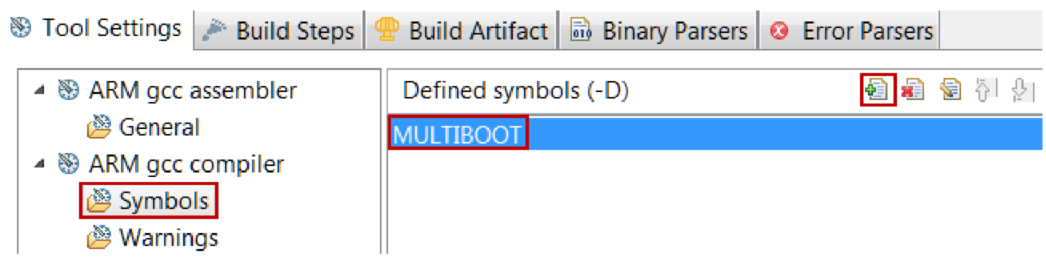


Figure 8 Setting user-defined symbol

5-3-3. Select **File** > **New** > **Application Project**

5-3-4. Enter **zynq_fsbl** as the project name, select the *Create New* option with **zynq_fsbl_bsp**, and click **Next**.

5-3-5. Select *Zynq FSBL* in the **Available Templates** pane and click **Finish**.

5-3-6. Select *lab1* in the Project Explorer pane, right-click, and select *Change Referenced BSP*.

5-3-7. In the displayed form, select *zynq_fsbl_bsp* and click **OK**.

The lab1 will be re-compiled using the zynq_fsbl_bsp.

5-3-8. In the SDK, select **Xilinx Tools > Create Zynq Boot Image**.

5-3-9. Select **Create new BIF file** option, click the *Browse* button of the BIF file path field and browse to *c:\xup\adv_embedded\labs\lab1* directory, set filename as *lab1*, and click **Save**.

5-3-10. Make the window bigger (taller), if necessary, so that you can see the **Output** folder field,

5-3-11. Add the three files, zynq_fsbl.elf, system_wrapper.bit, and lab1.elf with the correct path and order.

5-3-12. Change the output filename to **lab1.bin** making sure that the output directory is *lab1.*

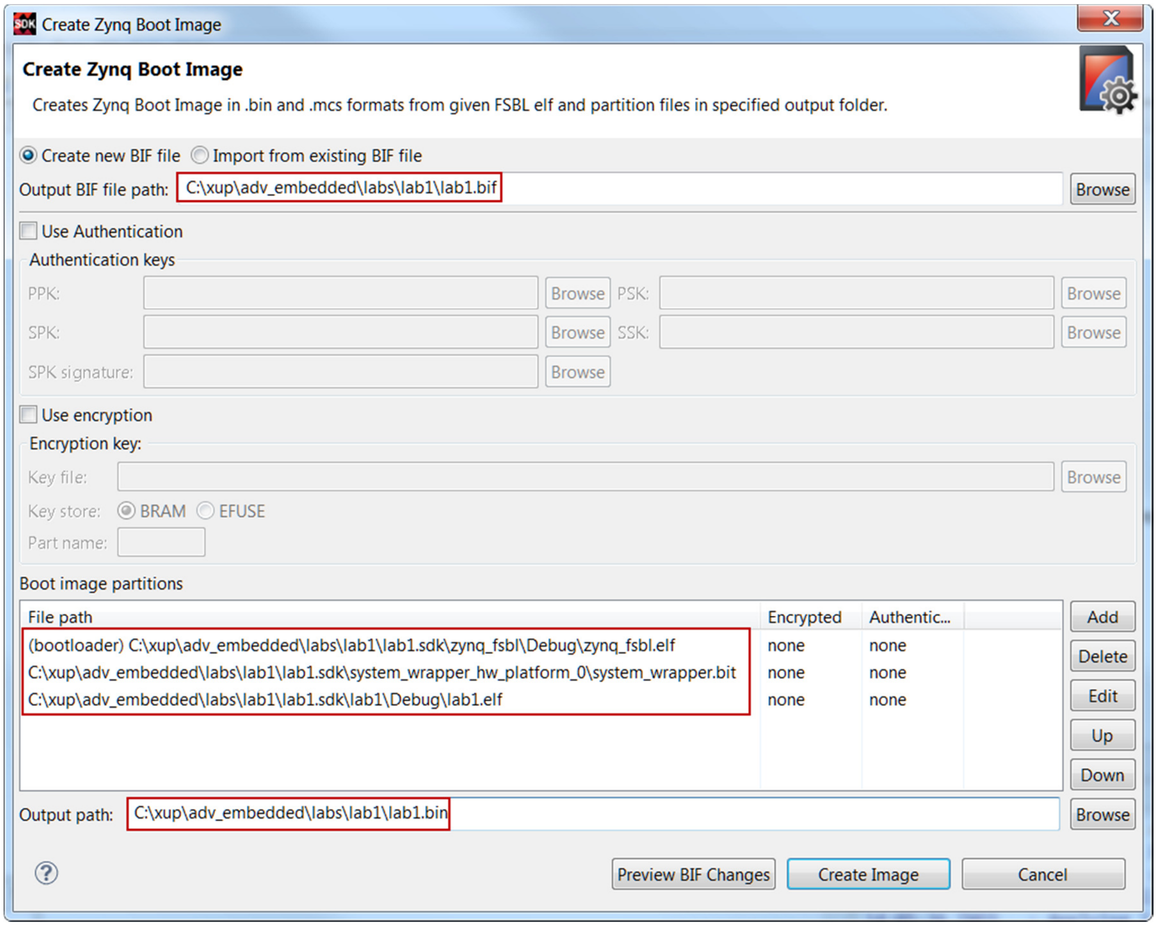


Figure 9 Creating the bin file of the lab1 project for the multiboot application

5-3-13. Click the **Create Image** button.

The lab1.bin will be created in the lab1 directory.

5-4. (OPTIONAL) Switch the workspace to lab3's SDK project. Assign all sections to ps7_ddr_0_S_AXI_BASEADDR and generate the linker script. Change the ps7_ddr_0_S_AXI_BASEADDR to 0x00600000 and the Size to 0x1FA00000 in the lscript.ld (linker script) file as you did in the previous step. Recompile the lab3.c file. Use the objcopy command to convert the elf file into the binary file and note the size of the binary file as well as the program entry point "main()".

This step is optional. If this step is skipped, proceed to 6-1.

5-4-1. In the **SDK** program switch the workspace by selecting **File > Switch Workspace > Other…** and browse to the workspace pointing to **c:\xup\adv_embedded\labs\lab3\lab3.sdk**\ and click **OK.**

5-4-2. Right-click on the **lab3** entry, select the *Generate Linker Script* option, change the *code, data, heap, and stack* sections to use the *ps7_ddr_0_AXI_BASEADDR*, and generate the linker script.

5-4-3. Expand the **lab3 > src** entry in the Project Explorer, and double-click on the **lscript.ld** to open it.

5-4-4. In the lscript editor view, change the Base Address of **ps7_ddr_0_AXI_BASEADDR** from **0x00100000** to **0x00600000**, and the Size from **0x1FF00000** to **0x1FA00000**.

5-4-5. Press **Ctrl-S** to save the change.

The program should compile.

5-4-6. In the SDK of the **lab3** project, select **Xilinx Tools > Launch Shell** to open the shell session.

5-4-7. In the shell window, change the directory to **lab3\Debug** using the **cd** command.

5-4-8. Convert the lab3.elf file to lab3elf.bin file by typing the following command.

arm-xilinx-eabi-objcopy -O binary lab3.elf lab3elf.bin

5-4-9. Type **ls –l** in the shell window and note the size of the file. In this case, it is **32780** again.

5-4-10. Determine the entry point (main()) of the program using the following command in the shell window.

arm-xilinx-eabi-objdump -S lab3.elf | grep main

It should be in the **0x00600558**.

5-4-11. Close the shell window.

5-5. Define MULTIBOOT symbol, create Zynq_fsbl application, and change the lab3 BSP reference to zynq_fsbl_bsp. Create the image using the zynq_fsbl.elf, system_wrapper.bit, and lab3.elf files and naming it as lab3.bin in the lab3 directory.

5-5-1. Right-click on the **lab3** entry, select the *C/C++ Build Settings* option.

5-5-2. Select *Symbols* in the left pane under the *ARM gcc compiler* group, click the + button on the right, enter **MULTIBOOT** in open form, click **OK**, and click **OK** again.

The application will re-compile as the MULTIBOOT related code is now included.

5-5-3. Select **File** > **New** > **Application Project**

5-5-4. Enter **zynq_fsbl** as the project name, select the *Create New* option with **zynq_fsbl_bsp**, and click **Next**.

5-5-5. Select *Zynq FSBL* in the **Available Templates** pane and click **Finish**.

5-5-6. Select *lab3* in the Project Explorer pane, right-click, and select *Change Referenced BSP*.

5-5-7. In the displayed form, select *zynq_fsbl_bsp* and click **OK**.

xup@xilinx.com

The lab3 will be re-compiled using the zynq_fsbl_bsp.

5-5-8. Select the **lab3** application in the *Project Explorer* view.

5-5-9. In the SDK, select **Xilinx Tools > Create Zynq Boot Image**.

5-5-10. Select **Create new BIF file** option, click the *Browse* button of the BIF file path field and browse to *c:\xup\adv_embedded\labs\lab3* directory, set filename as *lab3*, and click **Save**.

5-5-11. Make the window bigger (taller), if necessary, so that you can see the **Output** folder field,

5-5-12. Make sure that the three files, **zynq_fsbl.elf, system_wrapper.bit**, and **lab3.elf** file entries are added with the correct path. Change if necessary.

5-5-13. Change the output filename to **lab3.bin** making sure that the output directory is lab3.

5-5-14. Click the **Create Image** button.

The lab3.bin will be created in the lab3 directory.

Create the SD Card Image and Test — Step 6

6-1. Using the Windows Explorer, create the SD_image directory under the lab5 directory. You will first need to create the bin files from lab1 and lab3.

6-1-1. Using the Windows Explorer, create directory called **SD_image** under the **lab5** directory.

6-1-2. In Windows Explorer, copy the **system_wrapper.bit** of the lab1 project into the *SD_image* directory and rename it *lab1.bit*, and do similar for lab3

C:/xup/adv_embedded/labs/lab1/lab1.runs/impl_1/system_wrapper.bit -> SD_image /lab1.bit
C:/xup/adv_embedded/labs/lab3/lab3.runs/impl_1/system_wrapper.bit -> SD_image /lab3.bit

> The XSDK *bootgen* command will be used to convert the bit files into the required binary format. bootgen requires a .bif file which has been provided in the sources/lab5 directory. The .bif file specifies the target .bit files.

6-1-1. Open an XSDK command prompt by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2014.4 > Xilinx SDK 2014.4 Command Prompt**

6-1-2. In the XSDK command prompt window, change the directory to the bitstreams directory.

```
cd c:/xup/adv_embedded/labs/lab5/SD_image
```

6-1-3. Generate the partial bitstream files in the BIN format using the provided ".bif" file located in the *sources* directory. Use the following command:

```
bootgen -image ..\..\..\sources\lab5\bit_files.bif -w -
process_bitstream bin
```

6-1-4. Rename the files *lab1.bit.bin* and *lab3.bit.bin* to *lab1.bin* and *lab3.bin*

6-1-5. The size of the file needs to match the size specified in the **lab5_sd.c** file. The size can be determined by checking the file's properties. If the sizes do not match, then make the necessary change to the source code and save it (The values are defined as `LAB1_BITFILE_LEN` and `LAB3_BITFILE_LEN`).

```
C:\xup\adv_embedded\labs\SD_image>dir
 Volume in drive C is OSDisk
 Volume Serial Number is 4C0A-4F7D

 Directory of C:\xup\adv_embedded\labs\SD_image

01/12/2014  12:16    <DIR>          .
01/12/2014  12:16    <DIR>          ..
01/12/2014  10:11           153,420 BOOT.bin
01/12/2014  12:02         4,045,568 lab1.bin
06/11/2014  14:11         4,045,674 lab1.bit
01/12/2014  12:02         4,045,568 lab3.bin
06/11/2014  14:11         4,045,674 lab3.bit
```

Figure 10 Checking the size of the generate bin file (sizes are for Zedboard)

Note that the lab1.bin and lab3.bin files should be the same size.

6-1-6. Exit the SDK of lab3.

6-2. You will create the BOOT.bin file using the first stage bootloader and lab5_sd.elf files. Note that since there are no PL resources used, no bit file will be used.

6-2-1. In the SDK, select **Xilinx Tools > Create Zynq Boot Image**.

6-2-2. For the Output BIF file path, click on the Browse button and browse to **c:\xup\adv_embedded\labs\lab5\SD_image** directory and click **Save**.

6-2-3. Click on the **Add** button and browse to **C:\xup\adv_embedded\labs\lab5\lab5.sdk\zynq_fsbl\Debug**, select **zynq_fsbl.elf**, click **Open**, and click **OK**.

6-2-4. Click on the **Add** button of the *List of partitions in the boot image* field and add the software application, **lab5_sd.elf**, from the **c:\xup\adv_embedded\labs\lab5\lab5.sdk\lab5_sd\Debug** directory and click **OK**

6-2-5. Click the **Create Image** button.

The BOOT.bin file will be created in the **lab5\SD_image** directory.

1

6-3. Either copy the labxelf.bin files (two) from the sources directory or from the individual directories (if you did the optional part in the previous step and place them in the SD_image directory. Copy all the bin files to the SD card. Configure the board to boot from SD card. Power ON the board. Test the design functionality

6-3-1. In Windows explorer, copy the **lab1elf.bin** and **lab3elf.bin** files either from the **c:\xup\adv_embedded\sources\lab5\ [zybo | zedboard]\SDCard** directory or from the individual directories (if you did the optional parts in the previous step) and place them in the **SD_image** directory.

C:\xup\adv_embedded\labs\lab1\lab1.sdk\lab1\Debug\lab1elf.bin -> SD_image
C:\xup\adv_embedded\labs\lab3\lab3.sdk\lab3\Debug\lab3elf.bin -> SD_image

6-3-2. Insert a blank SD/MicroSD card (FAT32 formatted) in an SD Card reader, and using the Windows Explorer, copy the two bin files, the two elfbin files, and BOOT.bin from the **SD_image** folder in to the SD card.

6-3-3. Place the SD/MicroSD card in the board, and set the mode pins to boot the board from the SD card (Zedboard: JP7-JP11 as GND-SIG, GND-SIG, SIG-3V3, SIG-3V3, GND-SIG([right-to-left). Zybo set JP5 to SD). Connect your PC to the UART port with the provided micro-USB cable.

6-3-4. Power ON the board.

6-3-5. Start the terminal emulator program and follow the menu. Press the PS_RST/PS_SRST button (Red Button) if you don't see the menu.

6-3-6. When finished testing one application, either power cycle the board and verify the second application's functionality, or press the PS_RST/PS_SRST button (Red Button) on the board to display the menu again.

6-3-7. When done, power OFF the board.

Create the QSPI application and image Step 7

7-1. Create the lab5_qspi application using the provided lab5_qspi.c file.

7-1-1. Select **File** > **New** > **Application Project.**

7-1-2. Enter **lab5_qspi** as the project name, select the *Use existing* option for the Board Support Package, and using the drop-down button select *hello_world_bsp*, and click **Next**.

7-1-3. Select *Empty Application* in the **Available Templates** pane and click **Finish**.

7-1-4. Select **lab5_qspi>src** in the project view, right-click, and select **Import.**

7-1-5. Expand General folder and double-click on **File system,** and browse to the **c:\xup\adv_embedded\sources\lab5** directory.

7-1-6. Select **lab5_qspi.c** and click **Finish.**

The program should compile successfully and generate the lab5_qspi.elf file.

7-2. Using the Windows Explorer, create the QSPI_image directory under the lab5 directory. Create the lab5.mcs file using the zynq_fsbl.elf and lab5_qspi.elf files from the lab5, lab1.bin (from the lab1 project) and lab3.bin (from the lab3 project).

7-2-1. Using the Windows Explorer, create the **QSPI_image** directory under the **lab5** directory.

7-2-2. In the SDK, select **Xilinx Tools > Create Zynq Boot Image**.

7-2-3. Select **Create new BIF file** option, click the *Browse* button of the BIF file path field and browse to *c:\xup\adv_embedded\labs\lab5\QSPI_image* directory, select *output.bif*, and click **Save**.

7-2-4. Click on the **Add** button of the *Boot image partitions* window.

7-2-5. Click on the *Browse* button of the *File Path* field, browse to *c:\xup\adv_embedded\labs\lab5\lab5.sdk\zynq_fsbl\Debug, select zynq_fsbl.elf*, click **Open**, and then click **OK.**

7-2-6. Click on the **Add** button of the *Boot image partition* field again and add the software application, **lab5_qspi.elf**, from the *c:\xup\adv_embedded\labs\lab5\lab5.sdk\lab5_qspi\Debug* directory.

7-2-7. Click on the **Add** button again of the *Boot image partition* field again and add the **lab1.bin**, either of created boot image of the lab1 project (in *c:\xup\adv_embedded\labs\lab1)* or from the provided *c:\xup\adv_embedded\sources\lab5\zed (or zybo)\QSPI* directory. Click **Open**.

7-2-8. Enter **0x400000** in the *Offset* field and click **OK**.

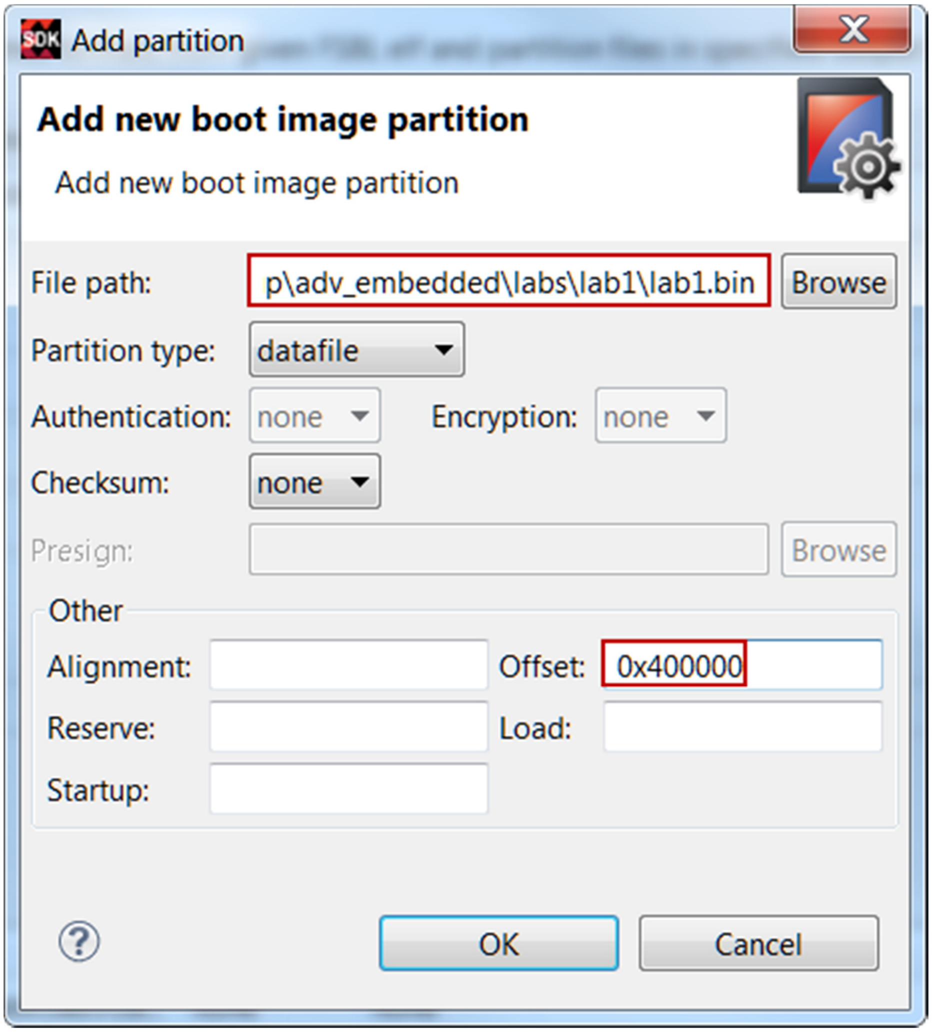


Figure 11 Adding boot image at an offset

7-2-9. Similarly, click on the **Add** button again of the *Boot image partition* field again and add the **lab3.bin**, either of the created boot image of the lab3 project (in *c:\xup\adv_embedded\labs\lab3)* or from the provided *c:\xup\adv_embedded\sources\lab5\zed (or zybo)\QSPI* directory. Click **Open**.

7-2-10. Enter **0x800000** in the *Offset* field and click **OK**.

7-2-11. Make the window bigger (taller), if necessary, so that you can see the **Output path** field,

7-2-12. Change the output filename to lab5.mcs and the location to *c:\xup\adv_embedded\labs\lab5\QSPI_image (if necessary).*

7-2-13. Click the **Create Image** button.

The lab5.mcs file will be created in the **lab5\QSPI_image** directory.

7-3. Make sure that the board is in the JTAG boot mode. Power ON the board. Program the QSPI using the Flash Writer utility.

7-3-1. Make sure that the board is in the JTAG boot mode. Power ON the board.

7-3-2. Select **Xilinx Tools > Program Flash**.

7-3-3. In the *Program Flash Memory* form, click the **Browse** button, and browse to the **c:\xup\adv_embedded\labs\lab5\QSPI_image** directory, select **lab5.mcs** file, and click **Open**.

7-3-4. In the *Offset* field enter **0** as the offset and click the **Program** button.

The QSPI flash will be programmed. It may take up to 4 minutes.

7-3-5. Power off the board.

Test the QSPI Multi-Applications Step 8

8-1. Set the board in the QSPI boot mode. Power ON the board. Connect to the serial port. Press the PS-SRST button and test the functionality.

8-1-1. Set the board in the QSPI mode.

8-1-2. Power ON the board.

8-1-3. Start the terminal emulator session and press PS-SRST button to see the menu.

8-1-4. Follow the menu and test the functionality of each lab.

Press 1 to load and execute lab1 or 2 to load and execute lab3. After each lab is executed, the lab5 gets loaded displaying the menu. Note that lab1 execution terminates when all slide switches are ON (i.e. 0xF) and lab3 execution terminates after it counts from 0 to 15.

8-1-5. Once satisfied, power OFF the board.

8-1-6. Close SDK and Vivado programs by selecting **File > Exit** in each program.

8-1-7. Turn OFF the power on the board.

Conclusion

This lab led you through creating the boot images which were capable of booting standalone applications from either the SD card or the QSPI flash memory. You then created the design capable of booting multiple applications and configurations which you developed in the previous labs.