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

Steps

Create a Vivado Project

Launch Vivado and create an empty project, called lab5, using the Verilog language.

1. Open Vivado and click **Create New Project** and click **Next**.
2. Click the Browse button of the *Project Location* field of the **New Project** form, browse to { **labs}** , and click **Select**.
3. Enter **lab6** in the *Project Name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.
4. Select the **RTL Project** option in the *Project Type* form, and click **Next**.
5. Select **Verilog** as the *Target Language* in the *Add Sources* form, and click **Next**.
6. Click **Next** two times.
7. Select your own board like **pynq-z2** and click **Next**.
8. Click **Finish** to create an empty Vivado project.

Creating the Hardware System Using IP Integrator

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

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

2. Name the block **system** and click **OK**.
3. Click the + button.
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.
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**.
6. Double click on the Zynq block to open the *Customization* window.

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

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

1. Click on the *MIO Configuration* panel to open its configuration form.
2. Expand the *IO Peripherals* on the right.
3. Uncheck *ENET 0*, *USB 0*, and *GPIO > GPIO MIO,* leaving *UART 0* and *SD 0* selected.
4. Click **OK**.

   The configuration form will close and the block diagram will be updated.
5. Using wiring tool, connect FCLK_CLK0 to M_AXI_GP0_ACLK.
6. Select the *Diagram* tab, and click on the ☑ (Validate Design) button to make sure that there are no errors.

Export the Design to the SDK and create the software projects

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

1. In Vivado, select the *Sources tab*, expand the *Design Sources,* right-click the *system.bd* and select **Create HDL Wrapper** and click **OK.**
2. Click on **Generate Bitstream** and click **Generate**. Click **Save** to save the project, and **Yes** if prompted to run the processes. Click **OK** to launch the runs.
3. When the bitstream generation process has completed successfully, click **Cancel**.

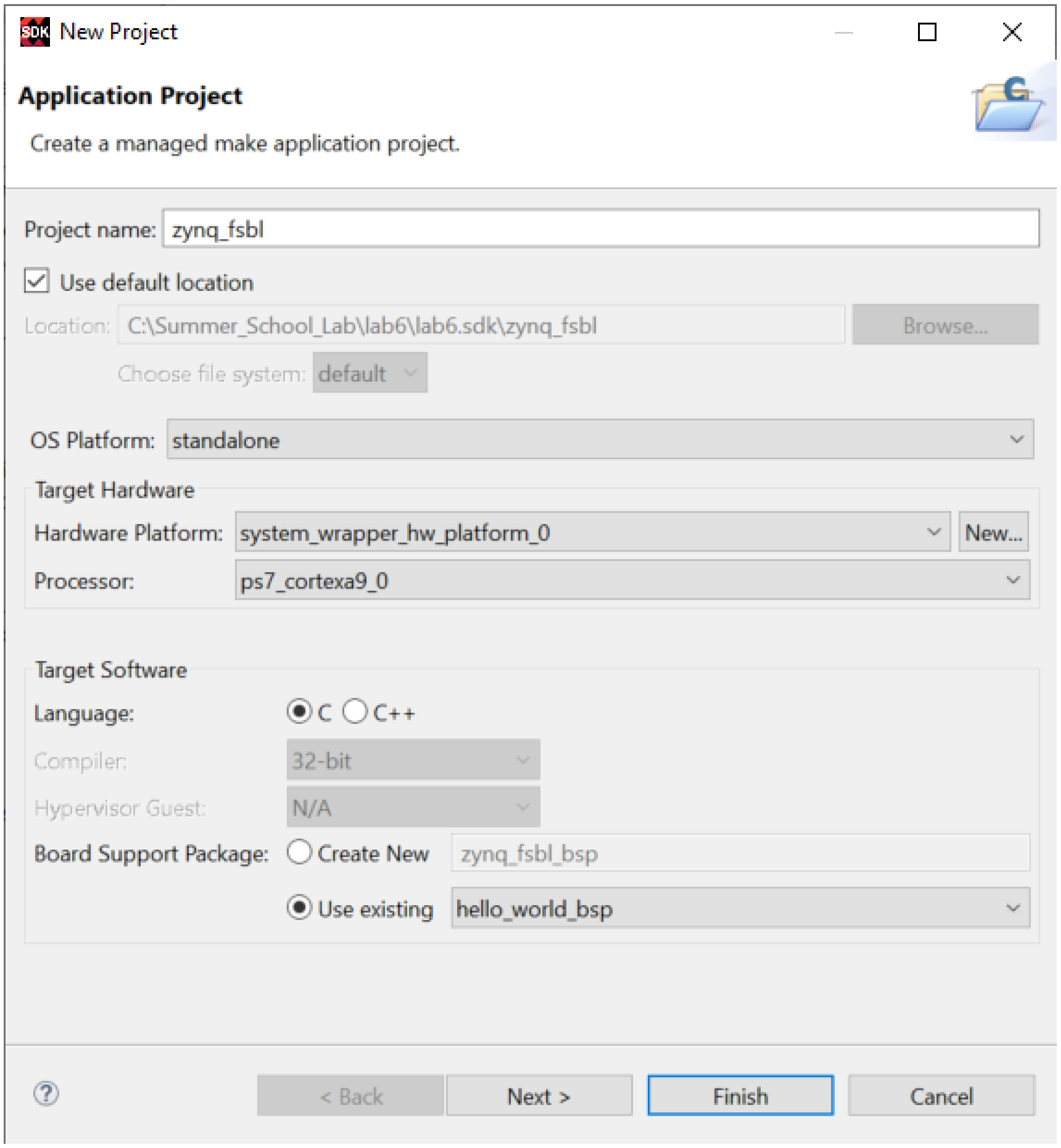
Export the design to the SDK and create the Hello World application.

1. Export the hardware configuration by clicking **File > Export > Export Hardware…**
2. Click the box to *Include Bitstream*, then click **OK**
3. Launch SDK by clicking **File > Launch SDK** and click **OK**
4. In SDK, select **File** > **New** > **Application Project.**
5. Enter **hello_world** in the project name field, and leave all other settings as default.
6. Click **Next** and make sure that the *Hello World* application template is selected, and click **Finish** to generate the application.
7. Right click on hello_world_bsp and click **Board Support Package Settings**
8. Tick to include *xilffs* click **OK** (This is required for the next step to create the FSBL).

Create a first stage bootloader (FSBL).

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

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


New Project

Application Project

Create a managed make application project.

Project name: zynq_fsbl

Use default location [✓]

Location: C:\Summer_School_Lab\lab6\lab6.sdk\zynq_fsbl

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: system_wrapper_hw_platform_0

Processor: ps7_cortexa9_0

Target Software

Language: [✓ C] C++

Compiler: 32-bit

Hypervisor Guest: N/A

Board Support Package: Create New zynq_fsbl_bsp [✓ Use existing] hello_world_bsp

Creating the FSBL Application

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

Using the Windows Explorer, create a directory called *image* under the lab6 directory. You will create the BOOT.bin file using the FSBL, system_wrapper.bit and hello_world.elf files.

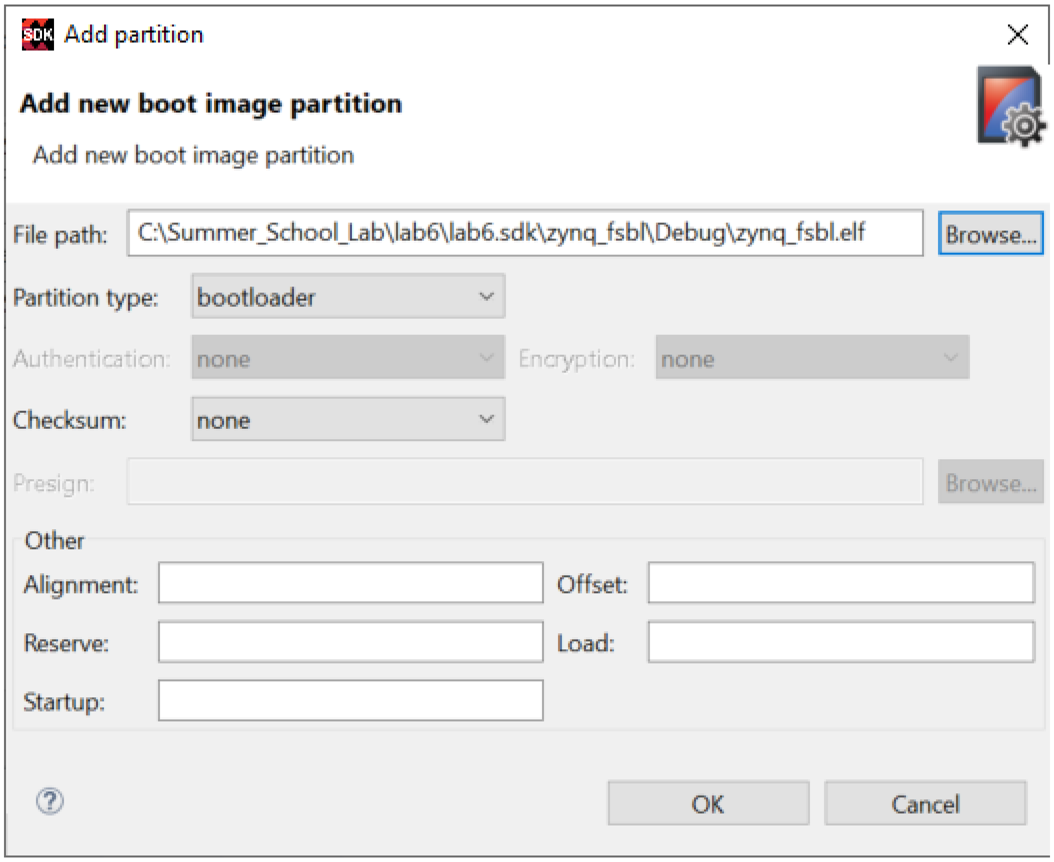
1. Using the Windows Explorer, create a directory under the *lab6* directory and call it **image**.

2. In the SDK, select **Xilinx > Create Boot Image**.

   Click on the Browse button of the **Output BIF file path** field, browse to **{labs}\lab6\image** and click **Save** (leaving the default name of output.bif)

3. Click on the **Add** button of the *Boot image partitions,* click the Browse button in the Add Partition form, browse to **{labs}\lab6\lab6.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.**



Add partition

Add new boot image partition

Add new boot image partition

File path: C:\Summer_School_Lab\lab6\lab6.sdk\zynq_fsbl\Debug\zynq_fsbl.elf

Partition type: bootloader

Authentication: none

Encryption: none

Checksum: none

Presign:

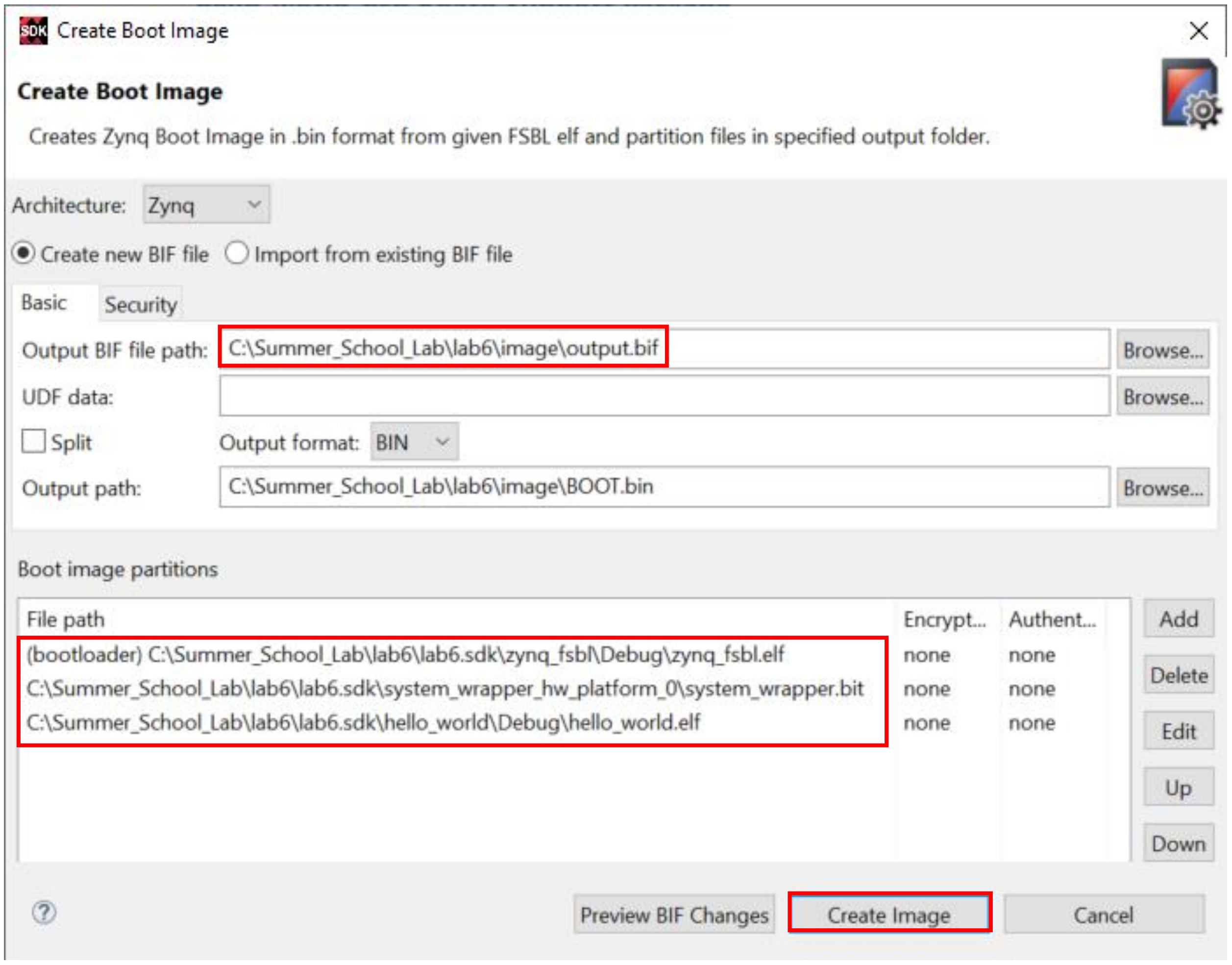
Adding FSBL partition

4. Click on the **Add** button of the *Boot image partitions* and add the bitstream, **system_wrapper.bit** ,
   from **{labs}\lab6\lab6.sdk\system_wrapper_hw_platform_0** and click **OK**.

5. Click on the **Add** button of the *Boot image partitions* and add the software application, **hello_world.elf** ,
   from **{labs}\lab6\lab6.sdk\hello_world\Debug** and click **OK**.

6. Click the **Create Image** button.

   The BOOT.bin and the output.bif files will be created in the **{labs}\lab6\image** directory. We will use the BOOT.bin for the SD card boot up.


Create Boot Image

Create Boot Image

Creates Zynq Boot Image in .bin format from given FSBL elf and partition files in specified output folder.

Architecture: Zynq

(● Create new BIF file) ○ Import from existing BIF file

Basic | Security

Output BIF file path: C:\Summer_School_Lab\lab6\image\output.bif

UDF data:

☐ Split   Output format: BIN

Output path: C:\Summer_School_Lab\lab6\image\BOOT.bin

Boot image partitions

File path	Encrypt...	Authent...
(bootloader) C:\Summer_School_Lab\lab6\lab6.sdk\zynq_fsbl\Debug\zynq_fsbl.elf	none	none
C:\Summer_School_Lab\lab6\lab6.sdk\system_wrapper_hw_platform_0\system_wrapper.bit	none	none
C:\Summer_School_Lab\lab6\lab6.sdk\hello_world\Debug\hello_world.elf	none	none

Creating BOOT.bin image file

7. 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.

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

1. Set the board in the SD card boot mode.
2. Insert the SD/MicroSD card into the board.
3. Power ON the board.
4. Connect your PC to the UART port with the provided micro-USB cable, and start Terminal or SDK Terminal in SDK or a Terminal emulator program setting it to the current COM port and 115200 baudrate.
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 button on the board.
6. Once satisfied power OFF the board and remove the SD card.

Set the board in the QSPI boot mode. Power ON the board, Program the QSPI using the Flash Writer utility, and test by pressing PS-RST button.

1. Set the board in the QSPI mode.

2. Power ON the board.

3. 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 an 115200 baud rate.

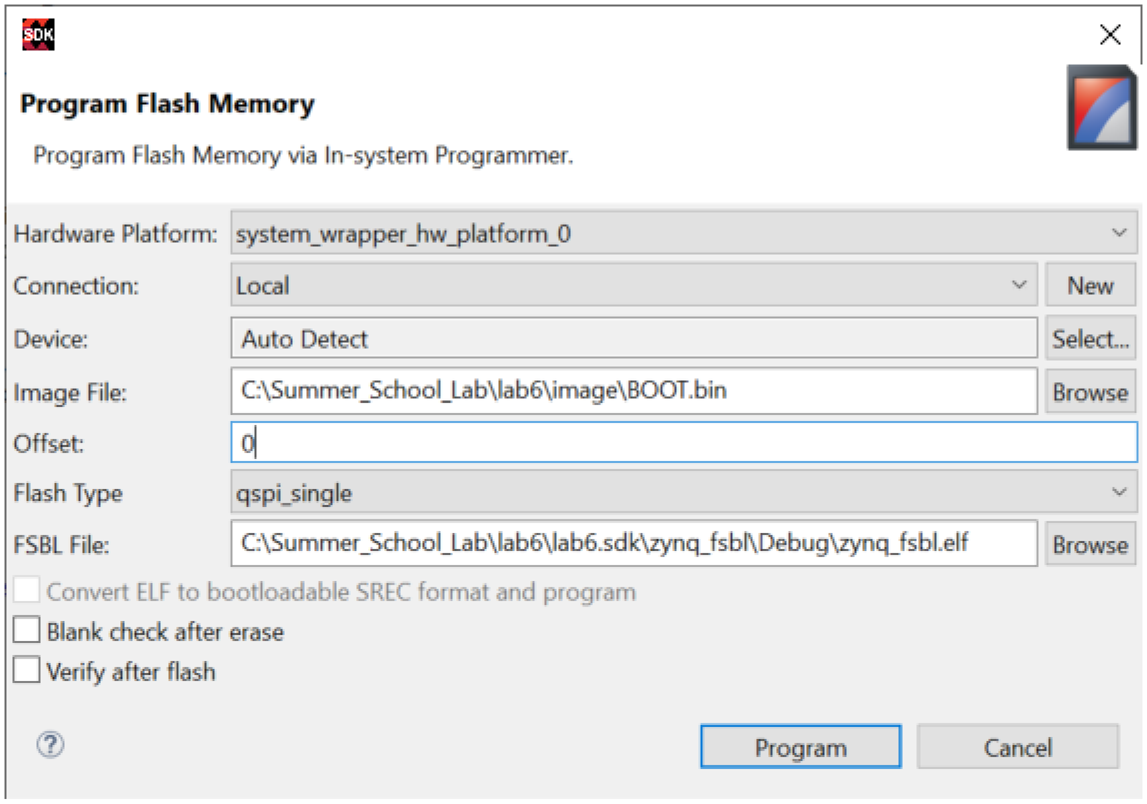
4. Select **Xilinx > Program Flash**.

5. In the *Program Flash Memory* form, click the **Browse** button of the Image File field, browse to the **{labs}\lab6\image** directory, select **BOOT.bin** file, and click **Open**.

6. Click on the FSBL File Browse button, browse to **{labs}\lab6\lab6.sdk\zynq_fsbl\Debug** directory (this is where the FSBL was created), select *zynq_fsbl.elf* and click **Open**.

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

   The QSPI flash will be programmed.



Program Flash Memory

Program Flash Memory via In-system Programmer.

Hardware Platform:	system_wrapper_hw_platform_0
Connection:	Local
Device:	Auto Detect
Image File:	C:\Summer_School_Lab\lab6\image\BOOT.bin
Offset:	0
Flash Type	qspi_single
FSBL File:	C:\Summer_School_Lab\lab6\lab6.sdk\zynq_fsbl\Debug\zynq_fsbl.elf

- [ ] Convert ELF to bootloadable SREC format and program
- [ ] Blank check after erase
- [ ] Verify after flash

Program Flash Memory form

8. Disconnect and reconnect the Terminal window.

9. Press the PS-RST to see the **Hello World** message in the terminal emulator window.

10. Once satisfied, power OFF the board.

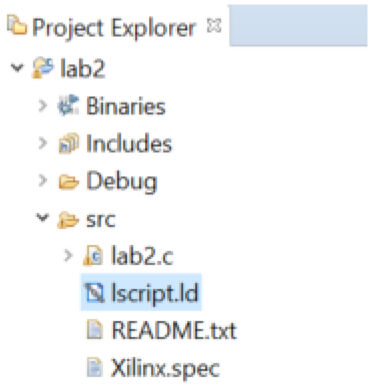
Prepare for the Multi-Applications Boot Using SD Card

The lab6 and lab4 executable files are required in the .bin format before copying to the SD card. The area in memory allocated for each application need to be modified so that they do not overlap each other, or with the main application.

Open the hello_world project, change the ps7_ddr_0 to 0x00200000 and the Size to 0x1FE00000 in the lscript.ld (linker script) file. Recompile the helloworld.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()).

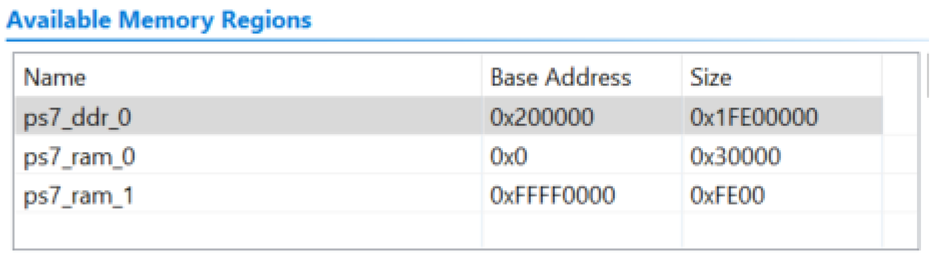
1. Right-click on the **hello_world** project, select the *Generate Linker Script* option, change the *code, data, heap, and stack* sections to use the *ps7_ddr_0*, and click **Generate**. Click **Yes** to overwrite the linker script.

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



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

3. 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**.



Available Memory Regions

Name	Base Address	Size
ps7_ddr_0	0x200000	0x1FE00000
ps7_ram_0	0x0	0x30000
ps7_ram_1	0xFFFF0000	0xFE00

Changing the Base address and the size

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

   The program should compile.

5. In the SDK of the **hello_world** project, select **Xilinx > Launch Shell** to open the shell session.

6. In the shell window, change the directory to **hello_world\Debug** using the **cd** command.

7. Convert the *hello_world.elf* file to *lab6elf.bin* file by typing the following command.

   arm-none-eabi-objcopy -O binary hello_world.elf lab6elf.bin

8. Type **ls –l** in the shell window and note the size of the file. In this case, it is **32776** , which is equivalent to **0x8008** bytes.

9. Determine the entry point "main()" of the program using the following command in the shell window.

   arm-none-eabi-objdump -S hello_world.elf | grep main

   It should be in the **0x002005b8.**

   Make a note of these two numbers (length and entry point) as they will be used in the lab6_sd application.

10. Close the Shell window.

Start another instance of the SDK program. Switch the workspace to lab2's SDK project. Assign all sections to ps7_ddr_0 and generate the linker script. Change the ps7_ddr_0 to 0x00600000 and the Size to 0x1FA00000 in the lscript.ld (linker script) file as you did in the previous step. Recompile the lab2.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()".

1. In the **SDK** program switch the workspace by selecting **File > Switch Workspace > Other...** and browse to the workspace pointing to *{labs}\lab4\lab4.sdk* and click **OK.**

2. Right-click on the **lab2** entry, select the *Generate Linker Script* option, change the *code, data, heap, and stack* sections to use the *ps7_ddr_0*, and generate the linker script.

3. Expand the **lab2 > src** entry in the Project Explorer, and double-click on the **lscript.ld** to open it.

4. In the lscript editor view, change the Base Address of **ps7_ddr_0** from **0x00100000** to **0x00600000** , and the Size from **0x1FF00000** to **0x1FA00000**.

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

   The program should compile.

6. In the SDK of the **lab2** project, select **Xilinx > Launch Shell** to open the shell session.

7. In the shell window, change the directory to **lab2\Debug** using the **cd** command.

8. Convert the lab2.elf file to lab2elf.bin file by typing the following command.

   arm-none-eabi-objcopy -O binary lab2.elf lab2elf.bin

9. Type **ls –l** in the shell window and note the size of the file. In this case, it is **32776** again which is equivalent to **0x8008**.

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

    arm-none-eabi-objdump -S lab2.elf | grep main

    It should be in the **0x006005b8**.

    Make a note of these two numbers (length and entry point) as they will be used in the lab6_sd application.

11. Close the shell window.

12. Exit the SDK of lab2.

Create the lab6_sd application using the provided lab6_sd.c and devcfg.c, devcgf.h, load_elf.s files.

1. Select **File** > **New** > **Application Project.**
2. Enter **lab6_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.**
3. Select *Empty Application* in the **Available Templates** pane and click **Finish**.
4. Select **lab6_sd > src** in the project view, right-click, and select **Import.**
5. Expand the General folder and double-click on **File system,** and browse to the **{sources}\lab6** directory.
6. Select **devcfg.c, devcfg.h, load_elf.s** , and **lab6_sd.c,** and click **Finish.**
7. Change, if necessary, LAB6_ELFBINFILE_LEN, LAB6_ELFBINFILE_LEN, LAB4_ELF_EXEC_ADDR, LAB4_ELF_EXEC_ADDRvalues and save the file.

Create the SD Card Image and Test

Using the Windows Explorer, create the SD_image directory under the lab6 directory. You will first need to create the bin files from lab2 and lab6.

1. Using the Windows Explorer, create directory called **SD_image** under the **lab6** directory.

2. In Windows Explorer, copy the **system_wrapper.bit** of the lab2 project into the *SD_image* directory and rename it *lab2.bit*, and do similar for lab6

   {labs}/lab2/lab2.runs/impl_x/system_wrapper.bit -> SD_image /lab2.bit

   {labs}/lab6/lab6.runs/impl_x/system_wrapper.bit -> SD_image /lab6.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/lab6 directory. The .bif file specifies the target .bit files.

3. Open a command prompt by selecting **Xilinx > Launch Shell.**

4. In the command prompt window, change the directory to the bitstreams directory.

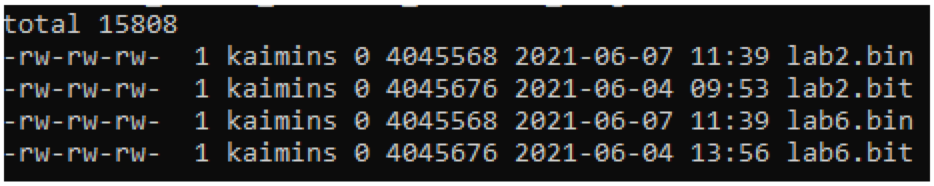
cd {labs}/lab6/SD_image

5. 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\lab6\bit_files.bif -w -process_bitstream bin

6. Rename the files *lab2.bit.bin* and *lab6.bit.bin* to *lab2.bin* and *lab6.bin*

7. The size of the file needs to match the size specified in the **lab6_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 LAB2_BITFILE_LEN and LAB6_BITFILE_LEN).



```
total 15808
-rw-rw-rw-  1 kaimins 0 4045568 2021-06-07 11:39 lab2.bin
-rw-rw-rw-  1 kaimins 0 4045676 2021-06-04 09:53 lab2.bit
-rw-rw-rw-  1 kaimins 0 4045568 2021-06-07 11:39 lab6.bin
-rw-rw-rw-  1 kaimins 0 4045676 2021-06-04 13:56 lab6.bit
```

Checking the size of the generate bin file

Note that the lab2.bin and lab4.bin files should be the same size.

You will create the BOOT.bin file using the first stage bootloader, system_wrapper.bit and lab6_sd.elf files.

1. In the SDK, select **Xilinx > Create Boot Image**.

2. For the Output BIF file path, click on the Browse button and browse to **{labs}\lab6\SD_image** directory and click **Save**.

3. Click on the **Add** button and browse to **{labs}\lab6\lab6.sdk\zynq_fsbl\debug** , select **zynq_fsbl.elf** , click **Open** , and click **OK**.

4. Click on the **Add** button of the *boot image partitions* field and add the bitstream file, **system_wrapper.bit** , from **{labs}\lab6\lab6.sdk\system_wrapper_hw_platform_0** and click **OK**

5. Click on the **Add** button of the *boot image partitions* field and add the software application, **lab6_sd.elf** , from the **{labs}\lab6\lab6.sdk\lab6_sd\Debug** directory and click **OK**

6. Click the **Create Image** button.

The BOOT.bin file will be created in the **lab6\SD_image** directory.

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

1. In Windows explorer, copy the **lab2elf.bin** and **lab6elf.bin** files from the individual directories and place them in the **SD_image** directory.

   {labs}\lab2\lab2.sdk\lab2\Debug\lab2elf.bin -> SD_image

   {labs}\lab6\lab6.sdk\hello_world\Debug\lab6elf.bin -> SD_image

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.

3. Place the SD/MicroSD card in the board, and set the mode pins to boot the board from the SD card. Connect your PC to the UART port with the provided micro-USB cable.

4. Power ON the board.

5. Start the terminal emulator program and follow the menu. Press the PS_RST/PS_SRST button if you don't see the menu.

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 on the board to display the menu again.

7. When done, power OFF the board.

Conclusion

This lab led you through creating the boot images which can boot 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.