Add Bram to System

Objectives

After completing this lab, you will be able to:

- Add block memory to the system
- Develop a linker script
- Partition the executable sections into both the DDR3 and BRAM spaces
- Generate an elf executable file
- Download the bitstream and application and verify on a Zynq board

Steps

Opening the Project

1. Start the Vivado if necessary and open either the lab3 project (lab3.xpr) you created in the previous lab or the lab3 project in the **{labsolutions}** directory using the Open Project link in the Getting Started page.
2. Select **File > Save Project As...** to open the Save Project As dialog box. Enter lab4 as the project name. Make sure that the Create Project Subdirectory option is checked, the project directory path is {labs} and click OK.

This will create the lab4 directory and save the project and associated directory with lab4 name.

Add the BRAM

1. Click **Open Block Design** under IP Integrator in the Flow Navigator pane

2. In the Block Diagram, Right click and select the Add IP option. Search for *BRAM* and add one instance of the *AXI BRAM Controller*

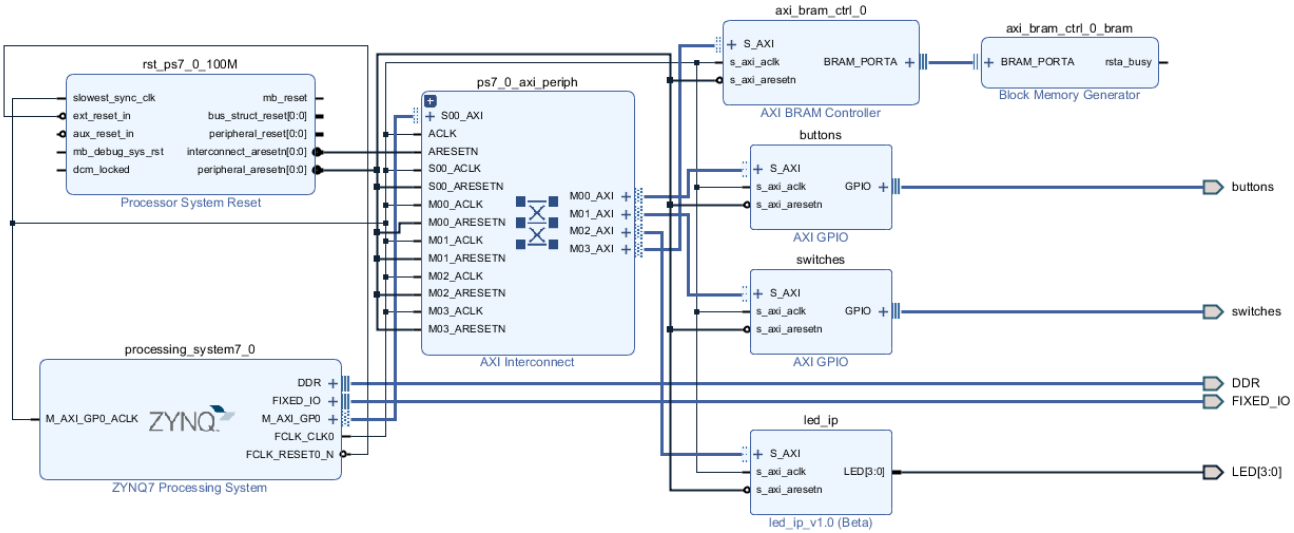
3. Run Connection Automation on **axi_bram_ctrl_0/S_AXI** and click OK when prompted to connect it to the M_AXI_GP0 Master.

4. Double click on the block to customize it and change the number of BRAM interfaces to 1 and click OK. Notice that the AXI Protocol being used is AXI4

instead of AXI4Lite since BRAM can provide higher bandwidth and the controller can support burst transactions.

5. Click on **Run Connection Automation** to add and connect a Block Memory Generator by selecting **axi_bram_ctrl_0/BRAM_PORTA** and click OK (This could be added manually)

6. Validate the design to ensure there are no errors (F6), and click the regenerate button ( ) to redraw the diagram. The design should look similar to the figure below.



Completed Block Diagram

7. In the Address editor, notice the Range of the *axibramctrl_0* is 8K. We will leave it at that.

8. Press **F6** to validate the design one last time.

9. Right click on system.bd and select Generate output products

10. Click on **Generate Bitstream** and click Yes if prompted to save the Block Diagram, and click Yes again if prompted to launch **Synthesis** and **Implementation**. Click Cancel when prompted to Open the Implemented Design

Export to SDK and create Application Project

1. Click **File > Export > Export Hardware**.

2. Click on the checkbox of **Include the bitstream** and then click **Yes** to overwrite.

3. Select **File > Launch SDK** and click OK.

4. To tidy up the workspace and save unnecessary building of a project that is not being used, right click on the *lab3*, *lab3_bsp*, and the *system_wrapper_hw_platform_2* projects from the previous lab, and click Close Project, as these projects will not be used in this lab. They can be reopened later if needed.

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

6. Enter lab4 as the Project Name, and for Board Support Package, choose Create New **lab4_bsp** (should be the only option).

7. Click Next, and select **Empty Application** and click Finish.

8. Expand lab4 in the project view and right-click in the src folder and select Import.

9. Expand General category and double-click on File System.

10. Browse to **{sources}\lab4** folder and click OK.

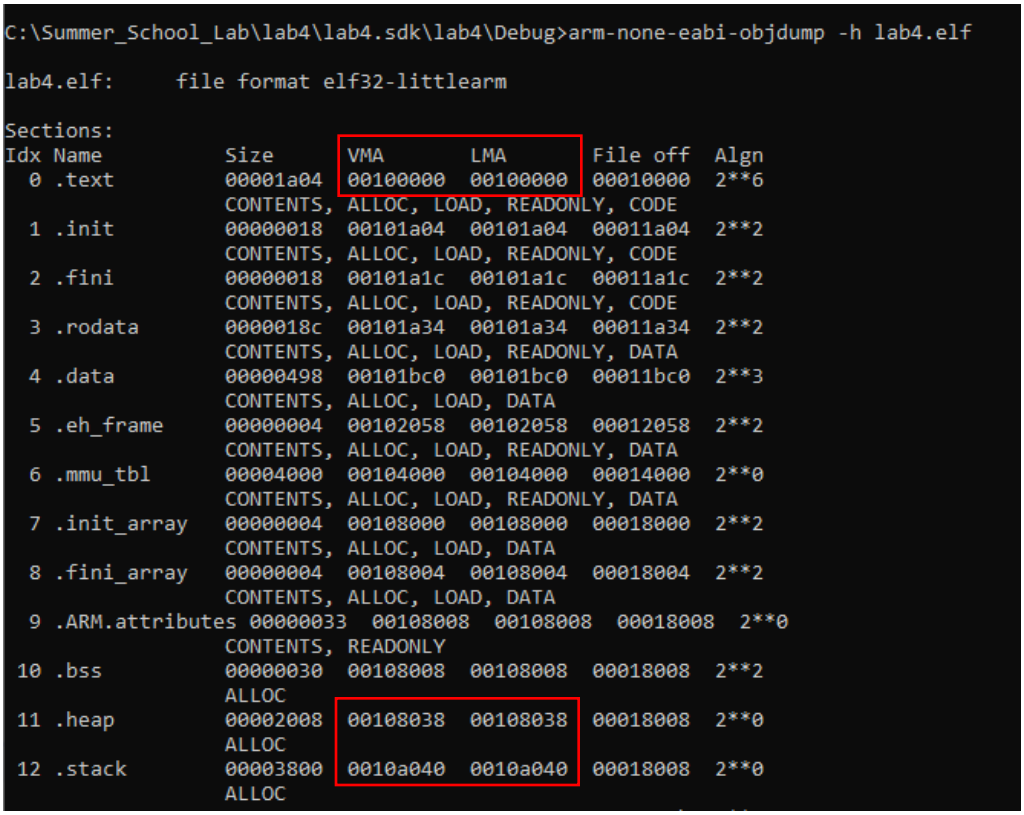
11. Select lab4.c and click Finish to add the file to the project.

Analyze Assembled Object Files

1. Launch the shell from SDK by selecting **Xilinx Tools > Launch Shell**.

2. Change the directory to **lab4\Debug** using the cd command in the shell. You can determine your directory path and the current directory contents by using the pwd and dir commands.

3. Type **arm-none-eabi-objdump –h lab4.elf** at the prompt in the shell window to list various sections of the program, along with the starting address and size of each section You should see results similar to that below:



```
C:\Summer_School_Lab\lab4\lab4.sdk\lab4\Debug>arm-none-eabi-objdump -h lab4.elf

lab4.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00001a04  00100000  00100000  00010000  2**6
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .init         00000018  00101a04  00101a04  00011a04  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .fini         00000018  00101a1c  00101a1c  00011a1c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .rodata       0000018c  00101a34  00101a34  00011a34  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .data         00000498  00101bc0  00101bc0  00011bc0  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  5 .eh_frame     00000004  00102058  00102058  00012058  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .mmu_tbl      00004000  00104000  00104000  00014000  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  7 .init_array   00000004  00108000  00108000  00018000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  8 .fini_array   00000004  00108004  00108004  00018004  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  9 .ARM.attributes 00000033  00108008  00108008  00018008  2**0
                  CONTENTS, READONLY
 10 .bss          00000030  00108008  00108008  00018008  2**2
                  ALLOC
 11 .heap         00002008  00108038  00108038  00018008  2**0
                  ALLOC
 12 .stack        00003800  0010a040  0010a040  00018008  2**0
                  ALLOC
```

Object dump results - .text, .stack, and .heap in the DDR3 space

Verify in Hardware

1. Make sure that micro-USB cable(s) is(are) connected between the board and the PC. Turn ON the power.

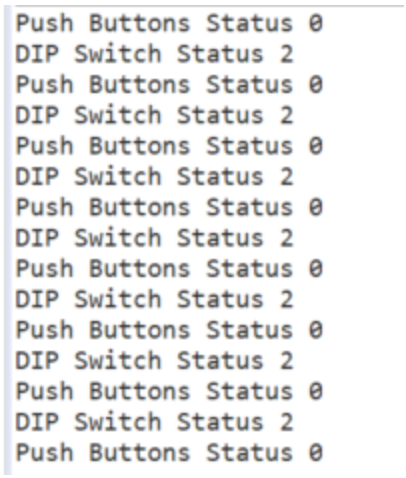
2. Select the tab. If it is not visible then select **Window > Show view > Other.. > Terminal**.

3. Click on the connect button and if required, select appropriate COM port (depends on your computer), and configure it with the parameters as shown. (These settings may have been saved from previous lab).

4. Select **Xilinx Tools > Program FPGA**.

5. Click the Program button to program the FPGA.

6. Select lab4 in Project Explorer, right-click and select **Run As > Launch on Hardware (System Debugger)** to download the application, execute ps7_init, and execute lab4.elf

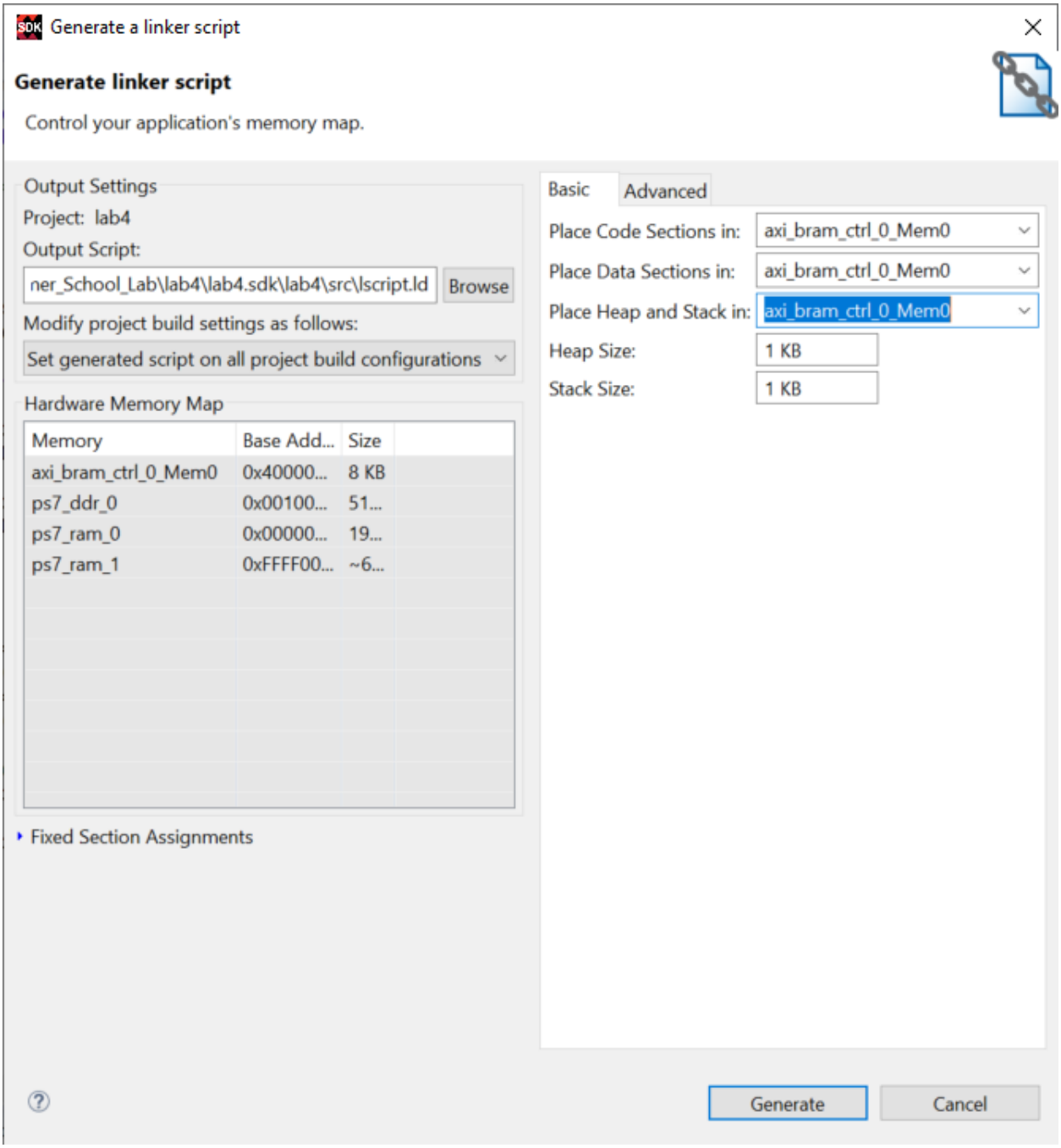


DIP switch and Push button settings displayed in SDK terminal

Note: Setting the DIP switches and push buttons will change the results displayed. DIP switches set to 3 will end the program.

7. Right click on lab4 and click Generate Linker Script… Note that all four major sections, code, data, stack and heap are to be assigned to BRAM controller.

8. In the Basic Tab change the Code and Data sections to **ps7_ddr_0**, leaving the Heap and Stack in section to **axi_bram_ctrl_0_S_AXI_BASEADDR** memory and click **Generate**, and click Yes to overwrite.

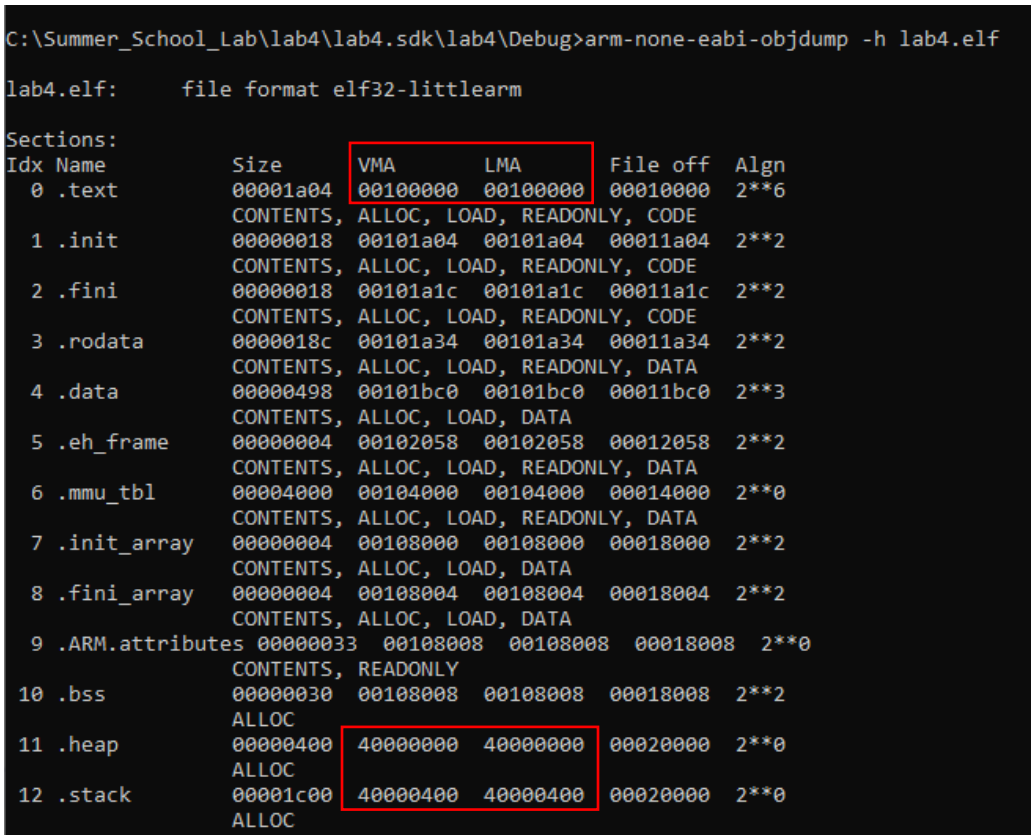


Targeting Stack/Heap sections to BRAM

The program will compile again.

9. Type **arm-none-eabi-objdump –h lab4.elf** at the prompt in the shell window to list various sections of the program, along with the starting address and size of each section

You should see results similar to that below:



```
C:\Summer_School_Lab\lab4\lab4.sdk\lab4\Debug>arm-none-eabi-objdump -h lab4.elf

lab4.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00001a04  00100000  00100000  00010000  2**6
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .init         00000018  00101a04  00101a04  00011a04  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .fini         00000018  00101a1c  00101a1c  00011a1c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .rodata       0000018c  00101a34  00101a34  00011a34  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .data         00000498  00101bc0  00101bc0  00011bc0  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  5 .eh_frame     00000004  00102058  00102058  00012058  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .mmu_tbl      00004000  00104000  00104000  00014000  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  7 .init_array   00000004  00108000  00108000  00018000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  8 .fini_array   00000004  00108004  00108004  00018004  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  9 .ARM.attributes 00000033  00108008  00108008  00018008  2**0
                  CONTENTS, READONLY
 10 .bss          00000030  00108008  00108008  00018008  2**2
                  ALLOC
 11 .heap         00000400  40000000  40000000  00020000  2**0
                  ALLOC
 12 .stack        00001c00  40000400  40000400  00020000  2**0
                  ALLOC
```

The .heap and .stack sections targeted to BRAM whereas the rest of the application is in DDR

Push the buttons and verify that the LEDs light according to the buttons settings. Verify that you see the results of the DIP switch and Push button settings in SDK Terminal.

10. Select lab4 in Project Explorer, right-click and select **Run As > Launch on Hardware (System Debugger)** to download the application, execute ps7_init, and execute lab4.elf

    Click Yes if prompted to stop the execution and run the new application.

    Observe the SDK Terminal window as the program executes. Play with dip switches and observe the LEDs. Notice that the system is relatively slow in displaying the message in the Terminal tab and to change in the switches as the stack and heap are from a non-cached BRAM memory.

11. When finished, click on the Terminate button in the Console tab.

12. Exit SDK and Vivado.

13. Power **OFF** the board.

Conclusion

An additional BRAM was added to the design. You can also use a linker script to target various segments in various memories. When the application is too big to fit in the internal BRAM, you can download the application in external memory and then execute the program.