Adding IP cores in PL

Objectives

After completing this lab, you will be able to:

- Configure the GP Master port of the PS to connect to IP in the PL
- Add additional IP to a hardware design
- Setup some of the compiler settings

Steps

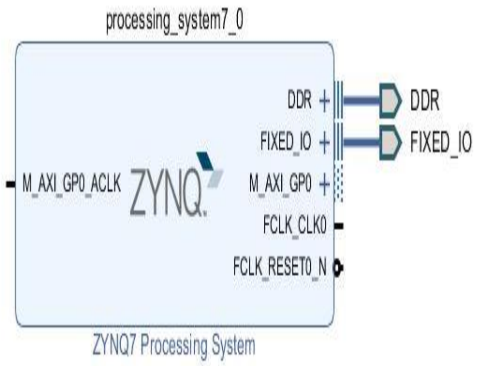
Open the Project

1. Open the previous project, or the lab1 project from the **{labsolutions}** directory, and save the project as lab2. Open the Block Design.
2. Start Vivado, if necessary, and open either the lab1 project (lab1.xpr) you created in the previous lab or from the **{labsolutions}** directory using the Open Project link in the Getting Started page.
3. Select **File > Save Project As...** to open the Save Project As dialog box. Enter lab2 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 lab2 directory and save the project and associated directory with lab2 name.

Add Two Instances of GPIO

1. In the Sources panel, expand system_wrapper, and double-click on the system.bd (system_i) file to invoke IP Integrator.
2. Double click on the Zynq block in the diagram to open the Zynq configuration window.
3. Select **PS-PL Configuration** page menu on the left.
4. Expand **AXI Non Secure Enablement > GP Master AXI Interfaces**, if necessary, and click on Enable **M_AXI_GP0 interface** check box under the field to enable the AXI GP0 port.
5. Expand **General > Enable Clock Resets** and select the FCLK_RESET0_N option.

6. Select the **Clock Configuration** tab on the left. Expand the **PL Fabric Clocks** and select the FCLK_CLK0 option (with requested clock frequency of 100.000000 MHz) and click OK.
7. Notice the additional *M_AXI_GPO* interface, and *M_AXI_GPO_ACLK*, *FCLK_CLK0*, and *FCLK_RESET0_N* ports are now included on the Zynq block. You can click the regenerate button to redraw the diagram to get something like this:

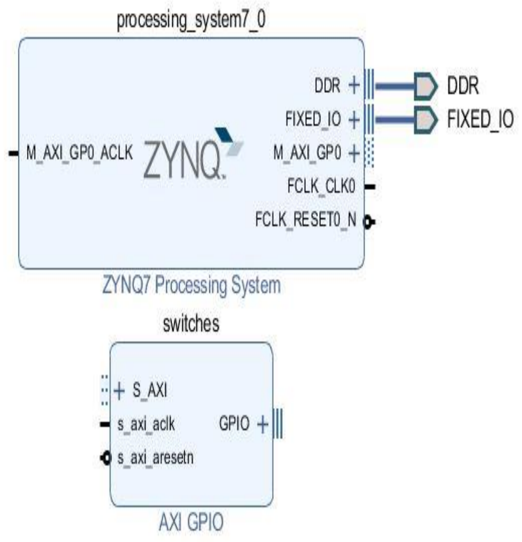


Zynq system with AXI and clock interfaces

8. Next add an IP by **right clicking on the Diagram window> Add IP** and search for AXI GPIO in the catalog

9. Double-click the *AXI GPIO* to add the core to the design. The core will be added to the design and the block diagram will be updated.

10. Click on the AXI GPIO block to select it, and in the properties tab, change the name to **switches**

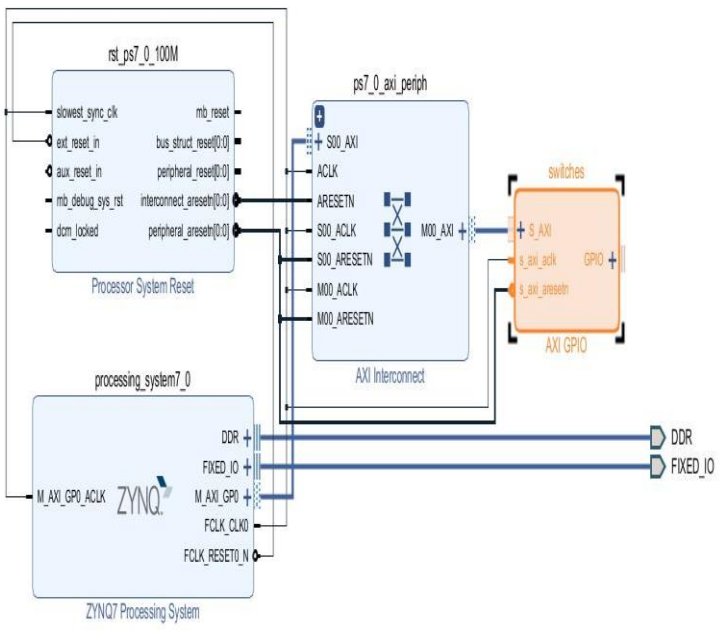


Change AXI GPIO default name

11. Double click on the *AXI GPIO block* to open the customization window.
12. From the Board Interface drop down, select sws 8bits for *ZedBoard*, sws 4bits for *Zybo* or sws 2bits for *PYNQ-Z2* for **GPIO IP Interface**.
13. Next, click the IP configuration tab, and notice the width has already been set to match the switches on the *Zedboard* (8), *Zybo* (4) or *PYNQ-Z2* (2)

Notice that the peripheral can be configured for two channels, but, since we want to use only one channel without interrupt, leave the Enable Dual Channel and Enable Interrupt unchecked.

14. Click OK to save and close the customization window
15. Notice that **Designer assistance** is available. Click on Run Connection Automation, and select **/switches/S_AXI**
16. Click OK when prompted to automatically connect the master and slave interfaces



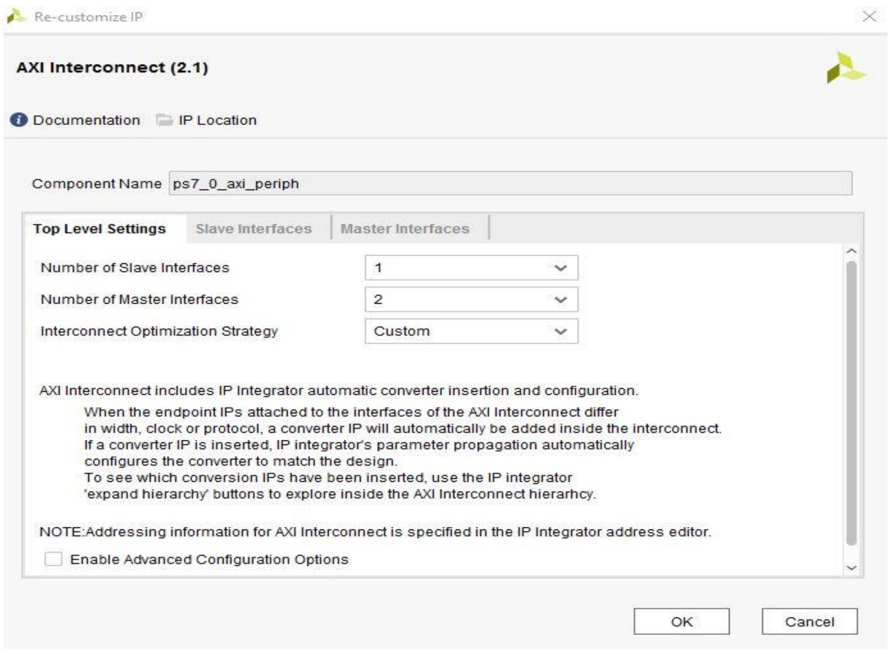
Design with switches automatically connected

Notice two additional blocks, Processor System Reset, and AXI Interconnect have automatically been added to the design. (The blocks can be dragged to be rearranged, or the design can be redrawn.).

18. Add another instance of the GPIO peripheral (Add IP). Name it as **buttons**

19. Double click on the IP block, select the *btns GPIO interface* (btns_5bits for the *Zedboard*, btns_4bits for the *Zybo* and btns 4bits for the *PYNQ-Z2*) and click OK. At this point connection automation could be run, or the block could be connected manually. This time the block will be connected manually.

20. Double click on the *AXI Interconnect* (name : ps7_0_axi_periph) and change the Number of **Master Interfaces** to 2 and click OK


Re-customize IP

AXI Interconnect (2.1)

Documentation   IP Location

Component Name: ps7_0_axi_periph

Top Level Settings | Slave Interfaces | Master Interfaces

Number of Slave Interfaces: 1
Number of Master Interfaces: 2
Interconnect Optimization Strategy: Custom

AXI Interconnect includes IP Integrator automatic converter insertion and configuration.
When the endpoint IPs attached to the interfaces of the AXI Interconnect differ in width, clock or protocol, a converter IP will automatically be added inside the interconnect. If a converter IP is inserted, IP integrator's parameter propagation automatically configures the converter to match the design. To see which conversion IPs have been inserted, use the IP integrator 'expand hierarchy' buttons to explore inside the AXI Interconnect hierarhcy.

NOTE:Addressing information for AXI Interconnect is specified in the IP Integrator address editor.

☐ Enable Advanced Configuration Options

OK   Cancel

Add master port to AXI Interconnect

21. Click on the s_axi port of the buttons AXI GPIO block (name: buttons), and drag the pointer towards the AXI Interconnect block.

    The message 'Found 1 interface' should appear, and a green tick should appear beside the M01_AXI port on the AXI Interconnect indicating this is a valid port to connect to. Drag the pointer to this port and release the mouse button to make the connection.

22. In a similar way, connect the following ports:

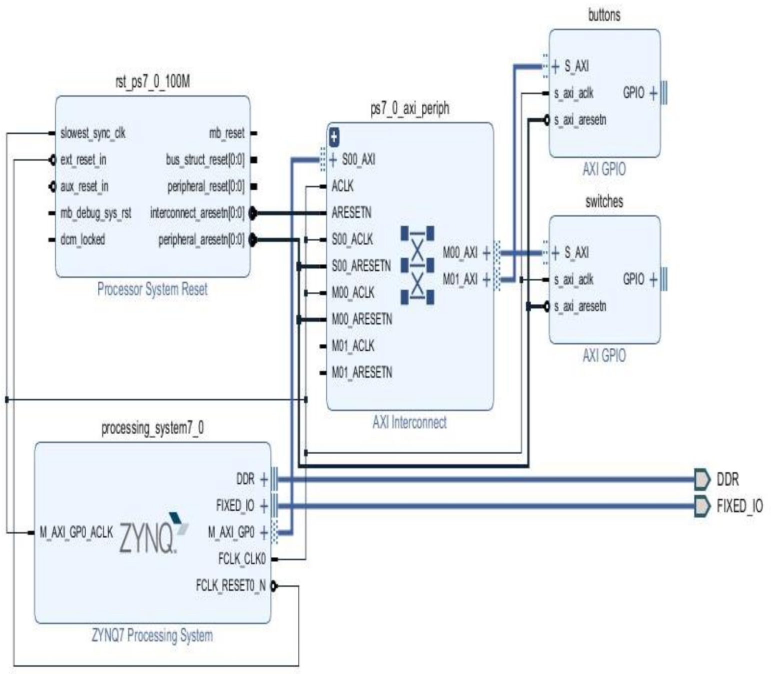
    buttons s_axi_aclk -> Zynq7 Processing System FCLK_CLK0

    buttons s_axi_aresetn -> Processor System Reset peripheral_aresetn

    AXI Interconnect M01_ACLK -> Zynq7 Processing System FCLK_CLK0

    AXI Interconnect M01_ARESETN -> Processor System Reset peripheral_aresetn

    The block diagram should look similar to this:

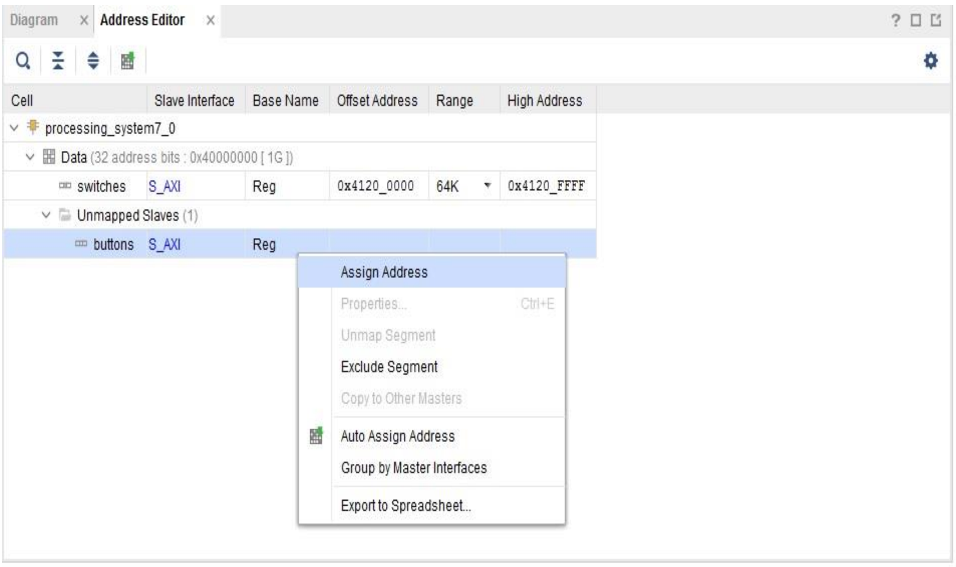


System Assembly View after Adding the Peripherals

23. Click on the **Address Editor** tab, and expand **processing_system7_0 > Data > Unmapped Slaves** if necessary

24. Notice that switches has been automatically assigned an address, but buttons has not (since it was manually connected). Right click on btns_4bit and select Assign Address.

Note that both peripherals are assigned in the address range of *0x40000000* to *0x7FFFFFFF* (GP0 range).



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
switches	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
Unmapped Slaves (1)					
buttons	S_AXI	Reg			

Peripherals Memory Map

Make GPIO Peripheral Connections External

1. In the Diagram view, notice that **Designer Assistance** is available. We will manually create the ports and connect.

2. Right-Click on the *GPIO port* of the switches instance and select **Make External** to create the external port. This will create the external port named **gpio** and connect it to the peripheral. Because Vivado is "board aware", the pin constraints will be automatically applied to the port.

3. Select the gpio port and change the name to **switches** in its properties form. The width of the interface will be automatically determined by the upstream block.

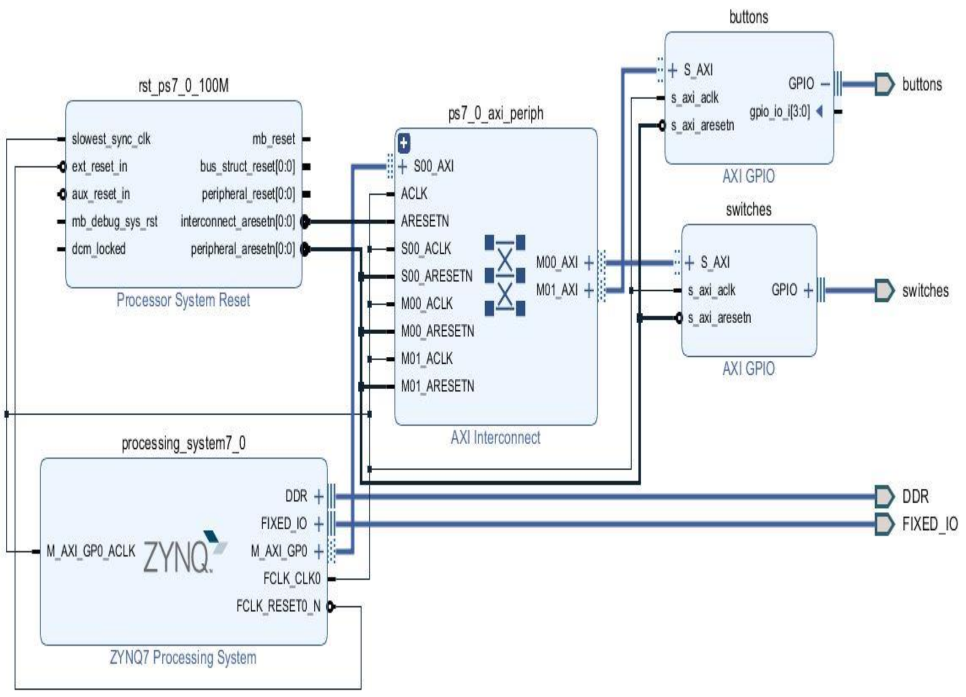
4. For the buttons GPIO, click on the Run Connection Automation link.

5. In the opened GUI, select btns_5bits (for *ZedBoard*) or btns_4bits (for *Zybo* and *PYNQ-Z2*) under the options section.

6. Click OK.

7. Select the created external port and change its name as buttons

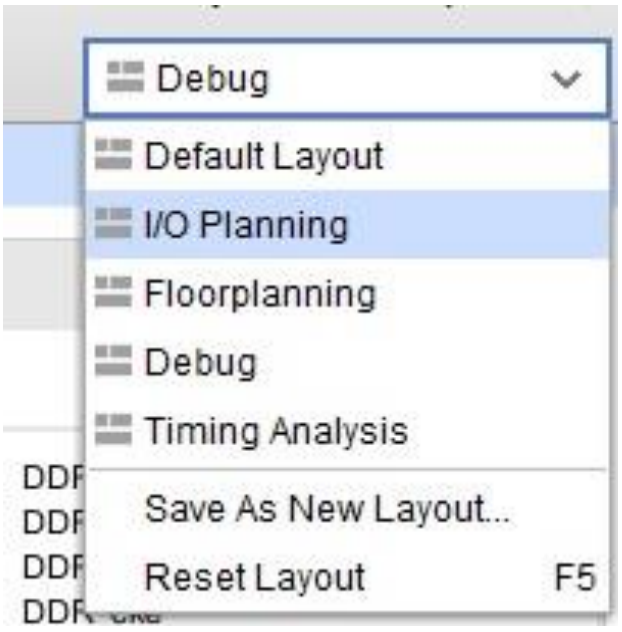
8. Run Design Validation (**Tools -> Validate Design**) and verify there are no errors. The design should now look similar to the diagram below



Completed design

9. In the Flow Navigator, click **Run Synthesis**. (Click Save if prompted) and when synthesis completes, select Open Synthesized Design and click OK

10. In the shortcut Bar, select **I/O Planning** from the Layout dropdown menu



Switch to the IO planning view

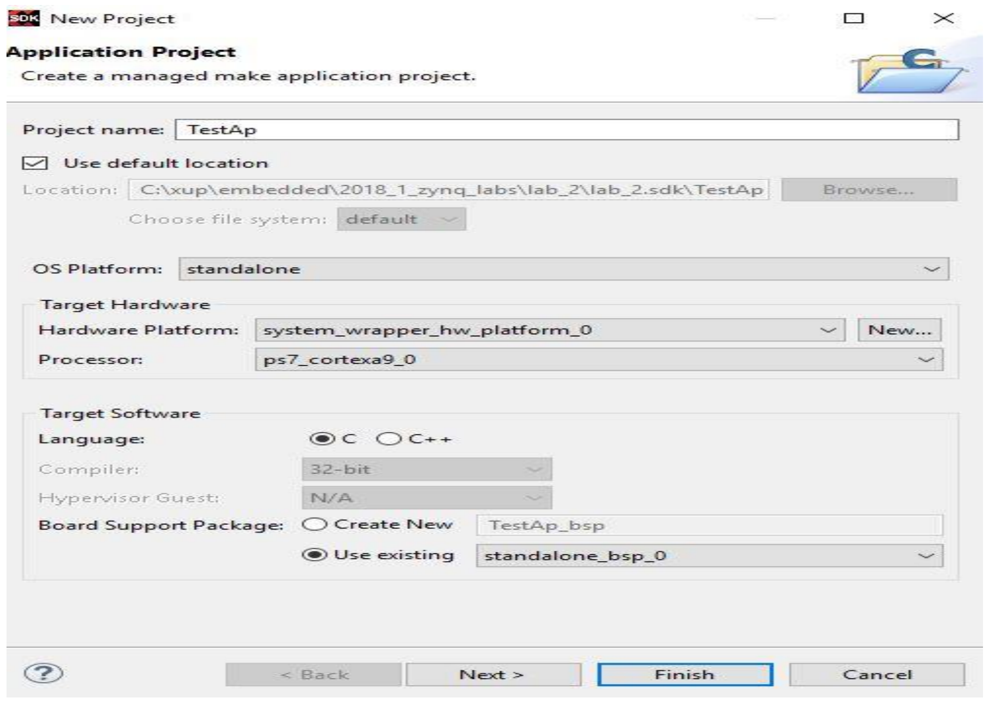
3. In the I/O ports tab, expand the two GPIO icons, and expand *buttons_tri_i*, and *switches_tri_i*, and notice that the ports have been automatically assigned pin locations, along with the other Fixed IO ports in the design, and an I/O Std of *LVCMOS25* (for *Zedboard*) and *LVCMOS33* (for *Zybo* and *PYNQ-Z2*) has been applied. If they were not automatically applied, pin constraints can be included in a constraints file, or entered manually or modified through the I/O Ports tab.

Generate Bitstream and Export to SDK

1. Click on **Generate Bitstream**, and click Yes if prompted to **Launch Implementation** (Click Yes if prompted to save the design)
2. Click Cancel
3. Export the hardware by clicking **File > Export > Export Hardware** and click OK. This time, there is hardware in Programmable Logic (PL) and a bitstream has been generated and should be included in the export to SDK.
4. Click Yes to overwrite the hardware module.
5. Start SDK by clicking **File > Launch SDK** and click OK

Generate TestApp Application in SDK

1. In SDK, right click on the mem_test project from the previous lab and select **Close Project**
2. Do the same for mem_test_bsp and system_wrapper_hw_platform_0
3. From the File menu select **File > New > Board Support Package**
4. Click Finish with the standalone OS selected and default project name as standalone_bsp_0
5. Click OK to generate the board support package named standalone_bsp_0
6. From the File menu select **File > New > Application Project**
7. Name the project **TestApp**, select Use existing board support package, select standalone_bsp_0 and click Next


New Project

Application Project

Create a managed make application project.

Project name: TestAp

Use default location [✓]

Location: C:\xup\embedded\2018_1_zynq_labs\lab_2\lab_2.sdk\TestAp

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 TestAp_bsp [✓ Use existing] standalone_bsp_0

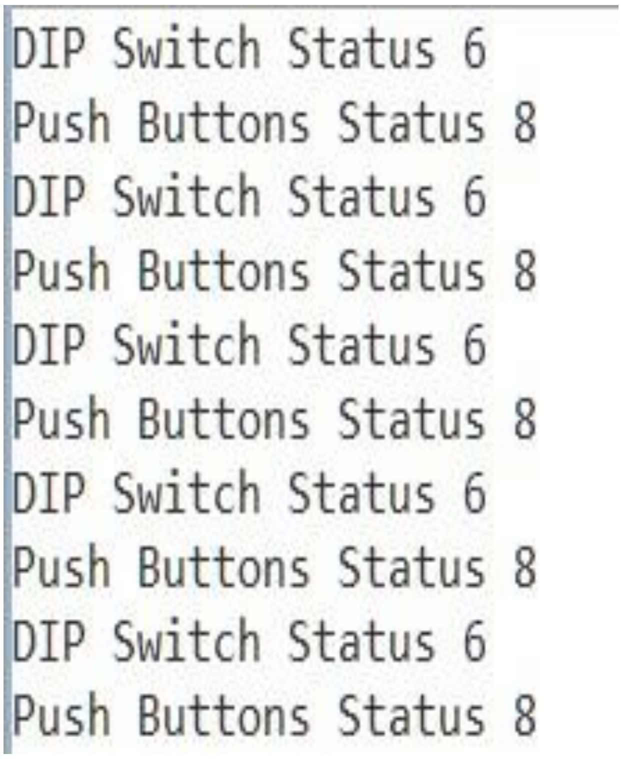
Application Project settings

8. Select Empty Application and click Finish This will create a new Application project using the created board support package.
9. The library generator will run in the background and will create the xparameters.h file in the lab2\lab2.sdk\standalone_bsp_0\ps7_cortexa9_0\include directory

10. Expand TestApp in the project view, and right-click on the src folder, and select Import
11. Expand General category and double-click on File System
12. Browse to the **{sources}\lab2** folder
13. Select **lab2.c** and click Finish

Test in Hardware

1. Make sure that micro-USB cable(s) is(are) connected between the board and the PC. Turn ON the power of the board.
2. Open Terminal from **Window > Show View > Other..**
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 in lab1. (These settings may have been saved from previous lab, lab1)
4. Select **Xilinx Tools > Program FPGA**
5. Click Program to download the hardware bitstream. When FPGA is being programmed, the DONE LED (green color) will be off, and will turn on again when the FPGA is programmed
6. Select TestApp in Project Explorer, right-click and select **Run As > Launch on Hardware** (System Debugger) to download the application, execute *ps7_init*, and execute *TestApp.elf*
7. You should see the something similar to the following output on Terminal console



```
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
```

SDK Terminal output

8. Select Console tab and click on the Terminate button ( ) to stop the program

9. Close SDK and Vivado programs by selecting **File > Exit** in each program

10. Power OFF the board

Conclusion

GPIO peripherals were added from the IP catalog and connected to the Processing System through the 32b Master GP0 interface. The peripherals were configured and external FPGA connections were established. A TestApp application project was created and the functionality was verified after downloading the bitstream and executing the program.