

Adding IP cores in PL

Introduction

This lab guides you through the process of extending the processing system you created in the previous lab by adding two GPIO (General Purpose Input/Output) IPs

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

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.

This lab comprises 6 primary steps: You will open the project in Vivado, add and configure GPIO peripherals in the system using IP Integrator, connect external ports, generate bitstream and export to SDK, create TestApp application in SDK, and, finally, verify the design in hardware.

Design Description

The purpose of this lab exercise is to extend the hardware design (**Figure 1**) created in Lab 1

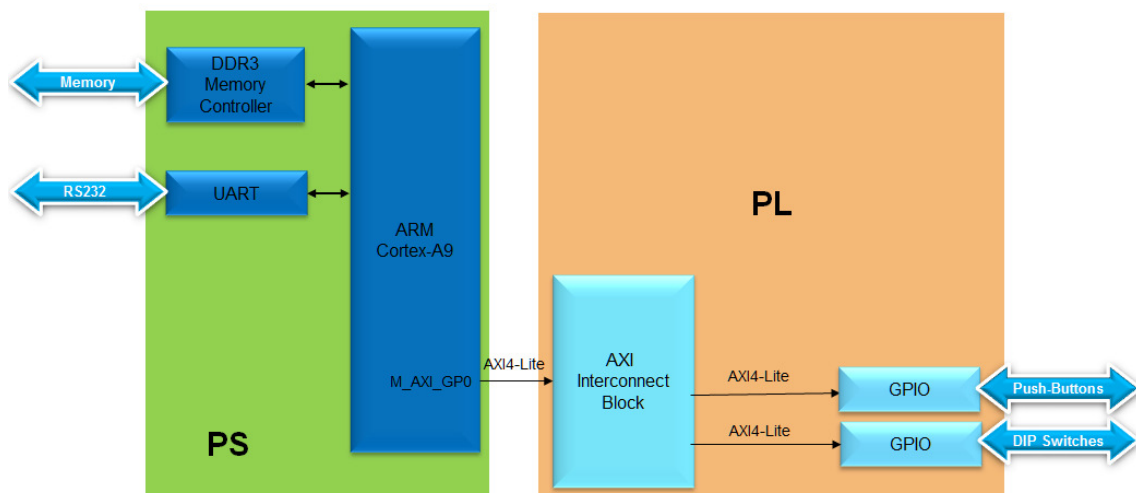
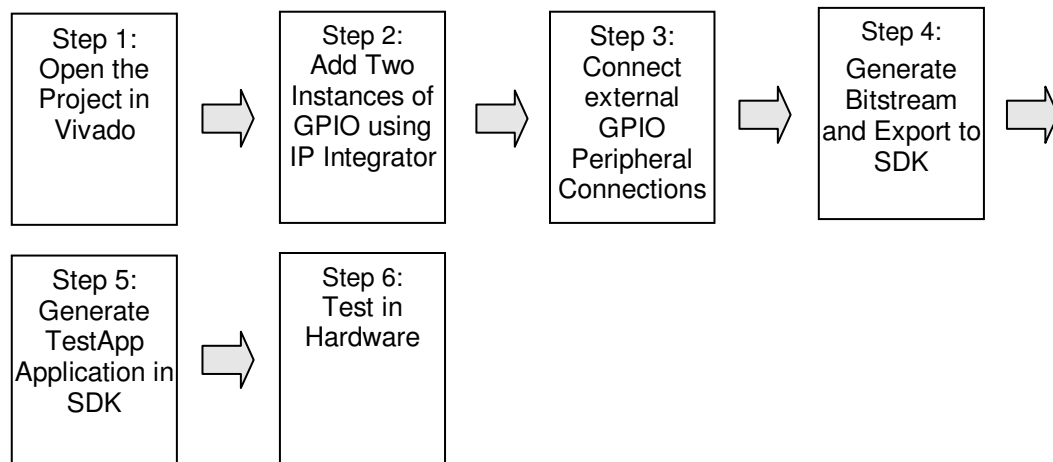


Figure 1. Extend the System from the Previous Lab

General Flow for this Lab



Open the Project

Step 1

1-1. Open the previous project, or the lab1 project from the labsolution directory, and save the project as lab2. Open the Block Design.

- 1-1-1. Start the Vivado if necessary and open either the lab1 project (lab1.xpr) you created in the previous lab or from the labsolution directory using the **Open Project** link in the Getting Started page.
- 1-1-2. 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 c:\xup\embedded\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

Step 2

2-1. Enable AXI_M_GP0 interface, FCLK_RESET0_N, and FCLK_CLK0 ports, Add two instances of an GPIO Peripheral from the IP catalog to the processor system.

- 2-1-1. In the *Sources* panel, expand *system_wrapper*, and double-click on the **system.bd** file to invoke IP Integrator. (The Block Design can also be opened from the Flow Navigator)
- 2-1-2. Double click on the Zynq block in the diagram to open the *Zynq configuration* window.
- 2-1-3. Select **PS-PL Configuration** page menu on the left, or click **32b GP AXI Master Ports** block in the Zynq Block Design view.

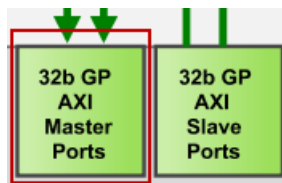


Figure 2. AXI Port Configuration

- 2-1-4. Expand *General Purpose Master AXI Interfaces* if necessary, and click on **Enable M_AXI_GP0 interface** check box under the field to enable the AXI GP0 port.

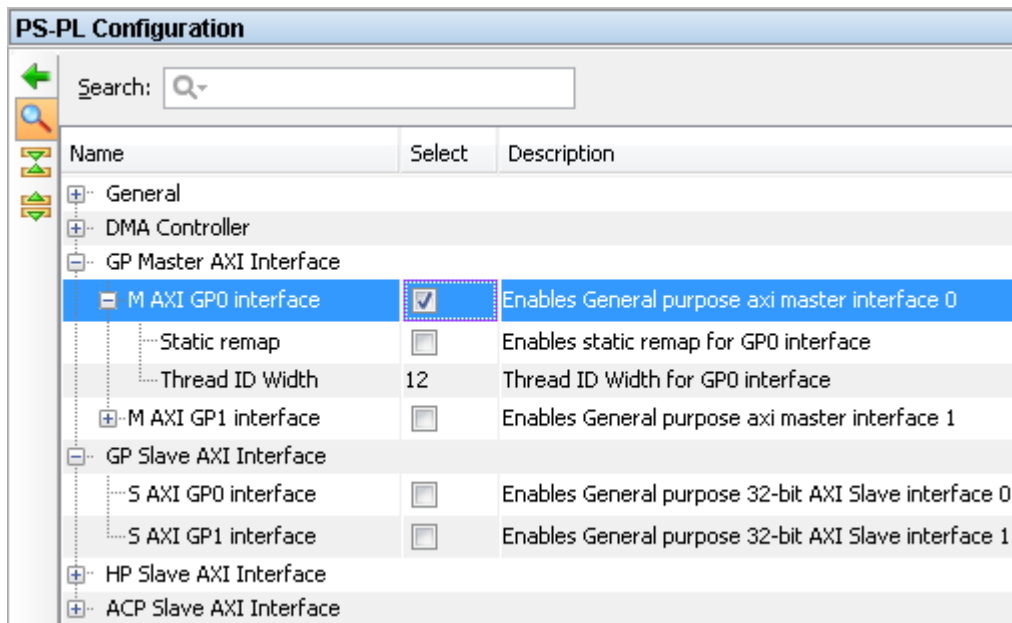


Figure 3. Configuration of 32b Master GP Block

- 2-1-5. Expand **General > Enable Clock Resets** and select the **FCLK_RESET0_N** option.
- 2-1-6. 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**.
- 2-1-7. 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.

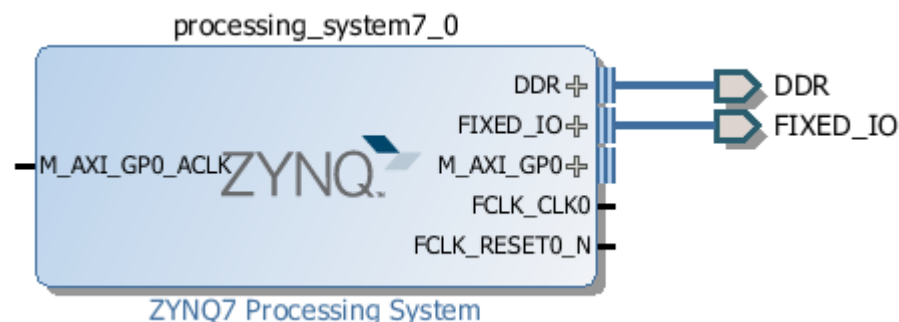



Figure 4. Zynq system with AXI and clock interfaces

2-1-8. Click the Add IP icon  and search for **AXI GPIO** in the catalog

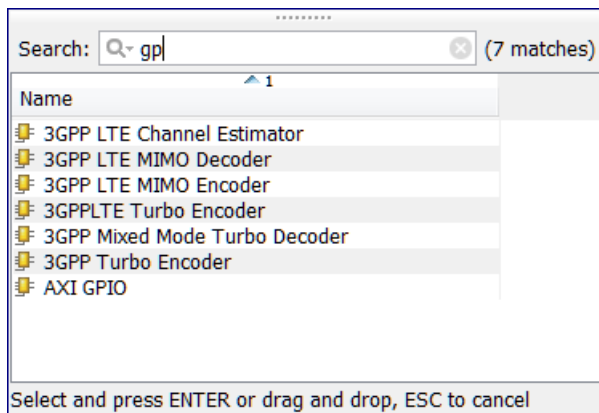


Figure 5. Add GPIO IP

2-1-9. 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.

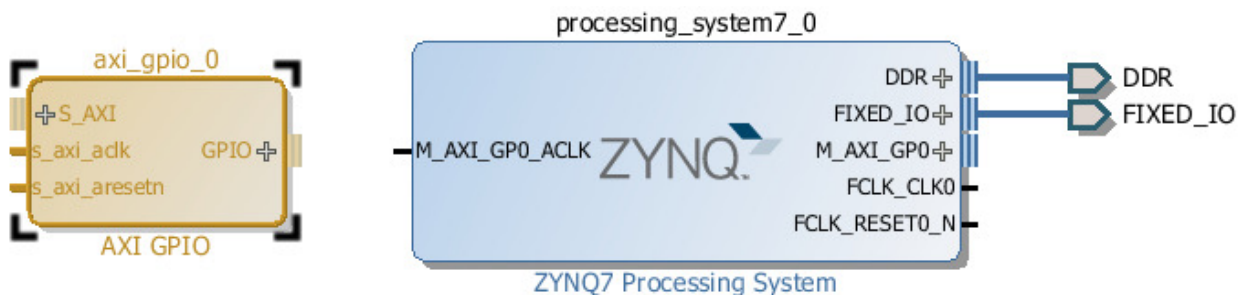


Figure 6. Zynq system with AXI GPIO added

2-1-10. Click on the **AXI GPIO** block to select it, and in the properties tab, change the name to **sw_4bit**

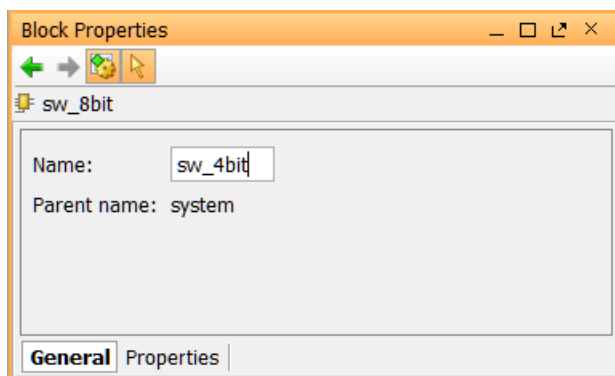


Figure 7. Change AXI GPIO default name

2-1-11. Double click on the **AXI GPIO** block to open the customization window.

If during the project creation, the target board had been selected then Vivado would have knowledge of available resources on the board. You could then have selected peripherals and generated the corresponding constraints based on the selected board. The GUI would have looked similar to the figure shown below.

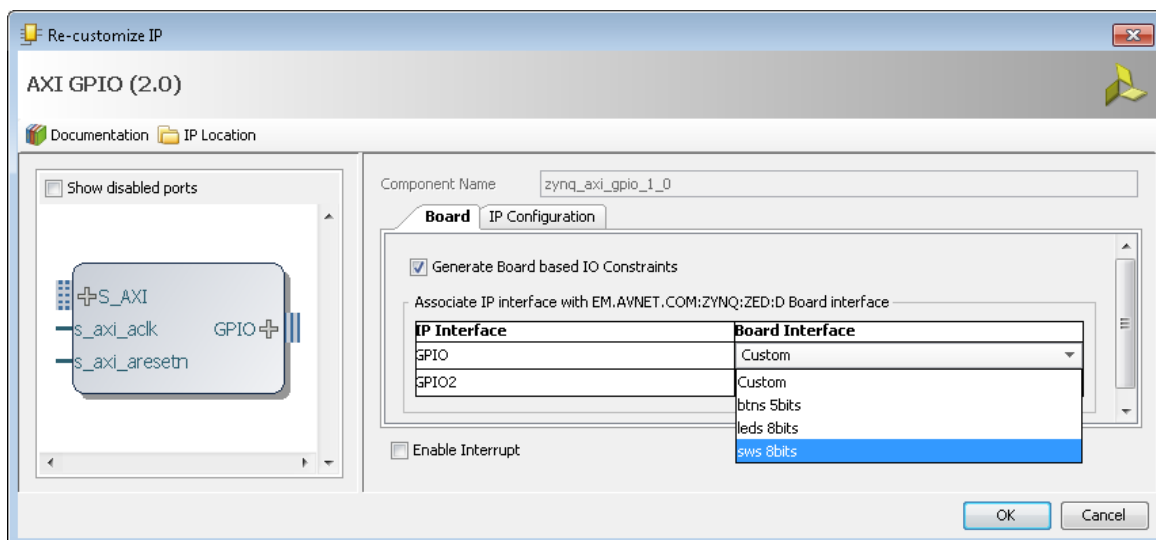


Figure 8. Configuring GPIO instance

Since the ZYBO board was not selected during the project creation, the GPIO has to be configured and the constraints will have to be explicitly added.

- 2-1-12.** Select the All Inputs option as the switches are input only type devices. Also, set the GPIO width to 4 as the ZYBO board has four switches.

Notice that the peripheral can be configured for two channels, but, since we want to use only one channel without interrupt, leave the *GPIO Supports Interrupts* and *Enable Channel 2* unchecked.

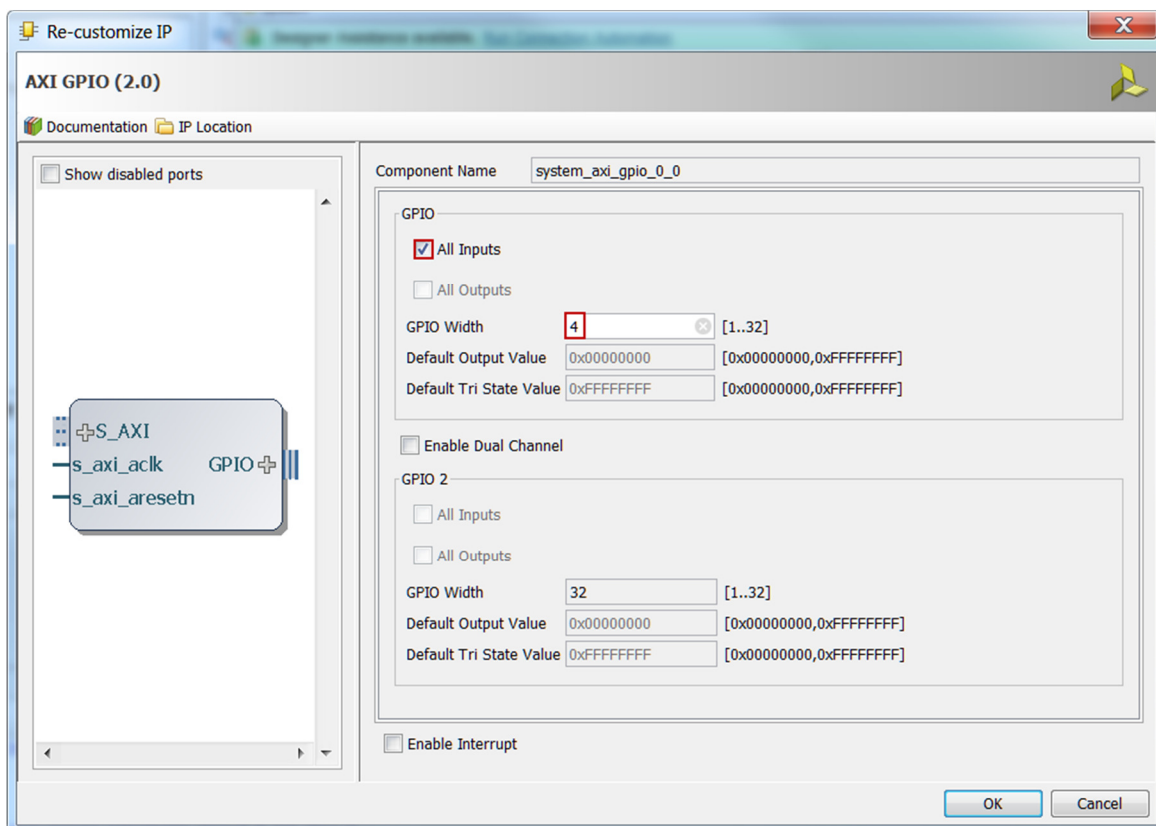


Figure 9. Configuring GPIO instance

2-1-13. Click **OK** to save and close the customization window

2-1-14. Notice that *Design assistance* is available. Click on **Run Connection Automation**, and select **/sw_4bit/S_AXI**

2-1-15. Click **OK** when prompted to automatically connect the master and slave interfaces

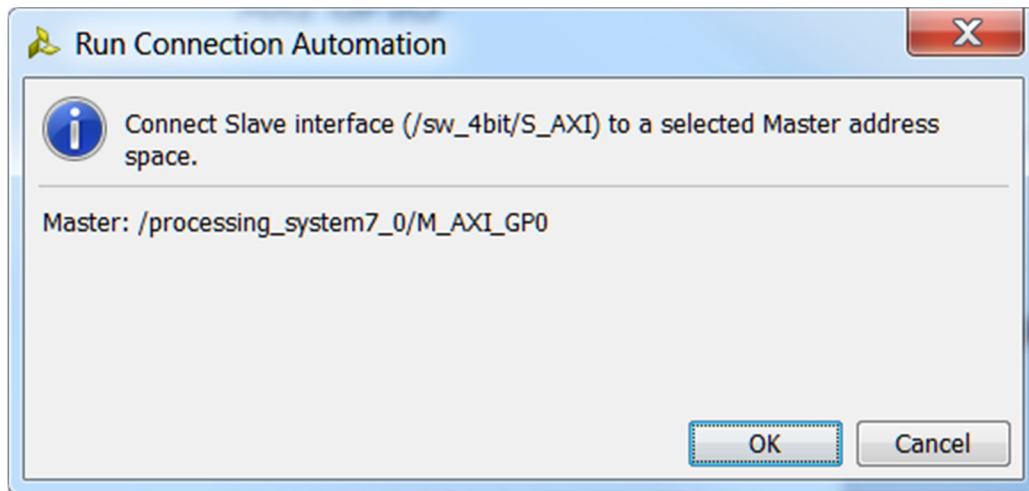


Figure 10. Run connection automation

2-1-16. 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.)

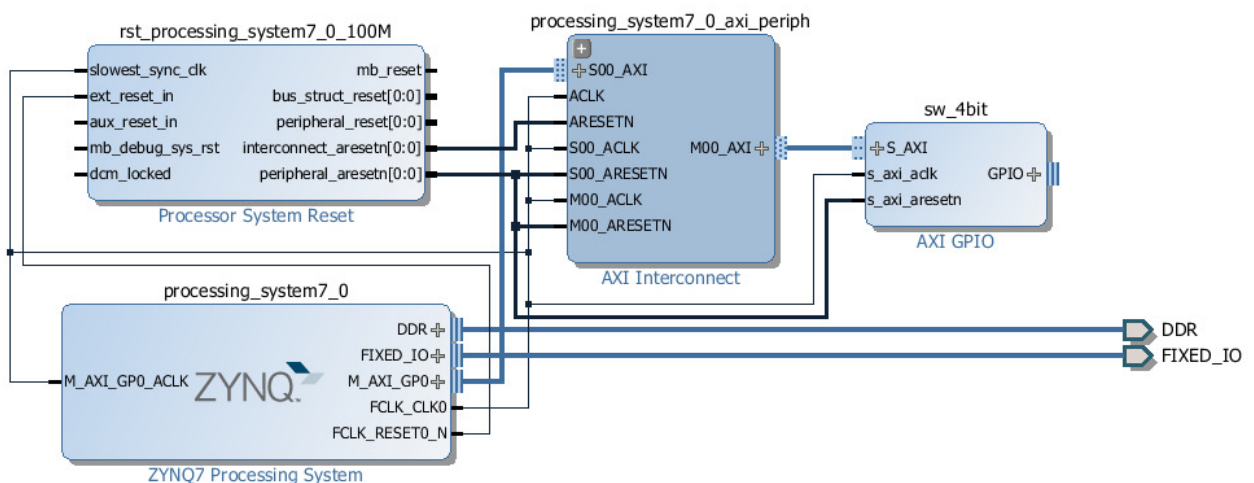


Figure 11. Design with SW_4bit automatically connected

2-1-17. Add another instance of the *GPIO* peripheral (**Add IP**). Name it as **btns_4bit**

2-1-18. Configure it to be all inputs with width of 4.

At this point connection automation could be run, or the block could be connected manually. This time the block will be connected manually.

- 2-1-19.** Double click on the AXI Interconnect and change the *Number of Master Interfaces* to **2** and click **OK**

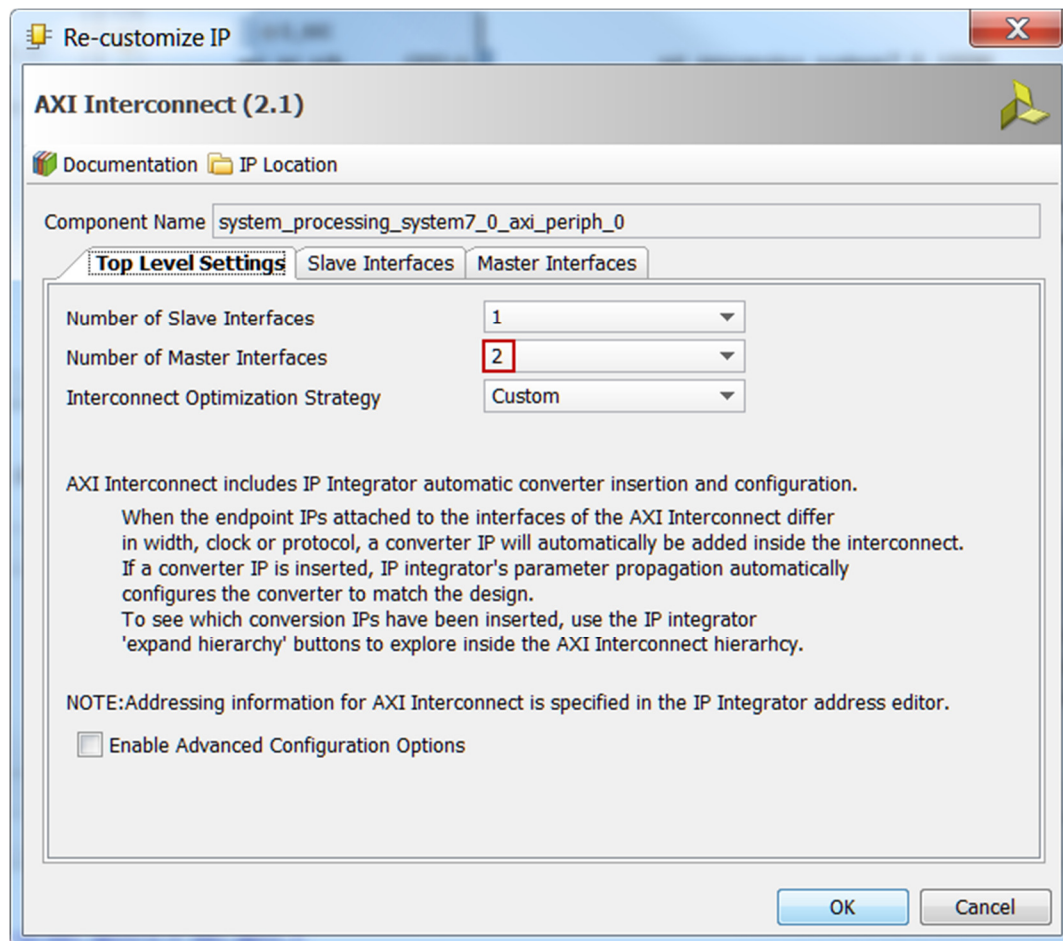


Figure 12. Add master port to AXI Interconnect

- 2-1-20.** Click on the `s_axi` port of the new AXI GPIO block, 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.

- 2-1-21.** In a similar way, connect the following ports:
- `btns_4bit s_axi_aclk` -> Zynq7 Processing System **FCLK_CLK0**
 - `btns_4bit s_axi_aresetn` -> Proc Sys Reset **peripheral_aresetn**
 - AXI Interconnect M01_ACLK** -> Zynq7 Processing System **FCLK_CLK0**
 - AXI Interconnect M01_ARESETN** -> Proc Sys Reset **peripheral_aresetn**

The block diagram should look similar to this:

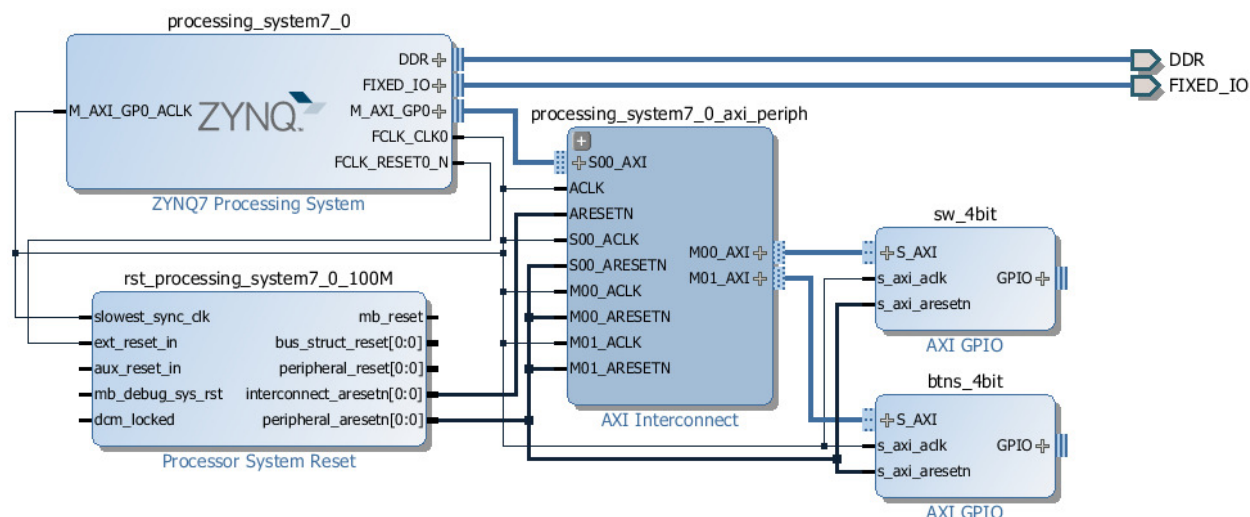



Figure 13. System Assembly View after Adding the Peripherals

2-1-22. Click on the *Address Editor* tab, and expand **processing_system7_0 > Data > Unmapped Slaves** if necessary

2-1-23. Notice that **sw_4bit** has been automatically assigned an address, but **btns_4bit** has not (since it was manually connected). Right click on **btns_4bit** and select **Assign Address** or click on the  button.

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

Diagram		Address Editor				
Cell	Interface Pin	Base Name	Offset Address	Range	High Address	
processing_system7_0						
Data (32 address bits : 4G)						
sw_4bit	S_AXI	Reg	0x41200000	64K	0x4120FFFF	
btns_4bit	S_AXI	Reg	0x41210000	64K	0x4121FFFF	

Figure 14. Peripherals Memory Map

Make GPIO Peripheral Connections External

Step 3

3-1. The push button and dip switch instances will be connected to corresponding pins on the ZYBO board. This can be done manually, or using *Designer Assistance*. The location constraints will have to be explicitly added as the tools are not aware of the ZYBO board. You will add the provided constraints. Normally, one would consult the ZYBO board user manual to find this information.

3-1-1. In the Diagram view, notice that *Designer Assistance* is available. Since the tools are not board aware we will ignore it, and we will manually create the ports and connect.

3-1-2. Right-Click on the gpio port of the **sw_4bit** instance and select **Make External** to create the external port. This will create the external port named **gpio** and connect it to the peripheral.

3-1-3. Select the *gpio* port and change the name to **sw_4bit** in its properties form.

The width of the interface will be automatically determined by the upstream block.

3-1-4. Similarly, create the external port on the *btns_4bit* block and naming it as *btns_4bit*

3-1-5. Run Design Validation (**Tools -> Validate Design**) and verify there are no errors.

The design should now look similar to the diagram below

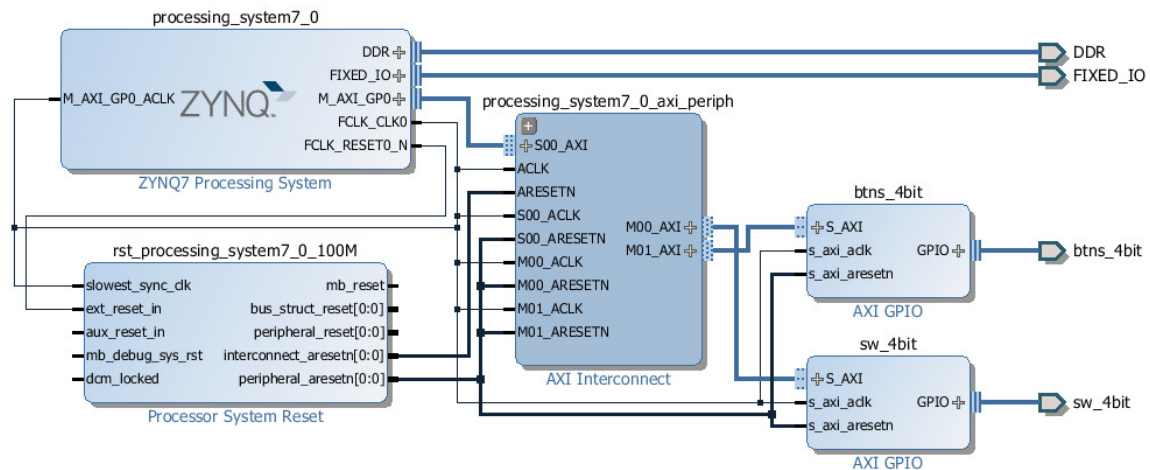


Figure 15. Completed design

3-1-6. In the *sources* view, Right Click on the block diagram file, **system.bd**, and select **Create HDL Wrapper** to update the HDL wrapper file. When prompted, click **OK**.

3-2. Add the provided *lab2.xdc* file that has constraints for *sw_4bit*, synthesize the design, open the I/O Planning layout, and add the *btns_4bit* constraints using the I/O planning tool.

3-2-1. In the Flow Navigator, click **Add Sources**

3-2-2. Select the *Add or Create Constraints* option and click **Next**.

3-2-3. Click on the **Add Files...** button, browse to **c:\xup\embedded\sources\lab2** and select *lab2.xdc*

3-2-4. Click **OK** and then **Finish**

3-2-5. In the Sources window, notice the *lab2.xdc* entry under the constraints folder.

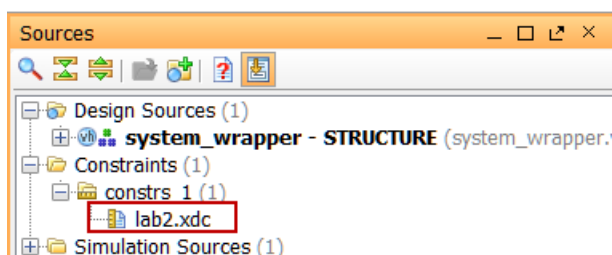


Figure 16. The constraint file added

3-2-6. In the Flow Navigator, click **Run Synthesis**. (Click **Save** when prompted) and when synthesis completes, select **Open Synthesized Design** and click **OK**

3-2-7. In the shortcut Bar, select **I/O Planning** from the *Layout* dropdown menu

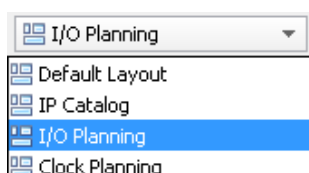


Figure 17. Switch to the IO planning view

3-2-8. In the I/O ports tab, expand *btns_4bit_tri_i*, and notice pins have not been assigned to this peripheral. The *sw_4bit_tri_i* have been assigned pin locations, along with the other Fixed ports in the design. Notice that the I/O Std of *btns_4bit_tri_i* have default to LVCMOS18 whereas *sw_4bit_tri_i* have been assigned LVCMOS33. Since they reside in the same bank, the two standards are conflicting with each other.

Name	Direction	Neg Diff Pair	S...	Fixed	Bank	I/O Std	Vcco	Vref
All ports (138)								
DDR_1497 (71)	In/Out				(Multiple)	(Multiple)*	1.500	(Multiple)
FIXED_IO_1497 (59)	In/Out				(Multiple)	(Multiple)*	(Multiple)	(Multiple)
GPIO_23532 (4)	Input					default (LVCMOS18)	1.800	
btns_4bit_tri_i (4)	Input					default (LVCMOS18)	1.800	
btns_4bit_tri_i[3]	Input					default (LVCMOS18)	1.800	
btns_4bit_tri_i[2]	Input					default (LVCMOS18)	1.800	
btns_4bit_tri_i[1]	Input					default (LVCMOS18)	1.800	
btns_4bit_tri_i[0]	Input					default (LVCMOS18)	1.800	
Scalar ports (0)								
GPIO_48824 (4)	Input				(Multiple)	LVCMOS33*	3.300	
sw_4bit_tri_i (4)	Input					LVCMOS33*	3.300	
sw_4bit_tri_i[3]	Input		T16	✓	34	LVCMOS33*	3.300	
sw_4bit_tri_i[2]	Input		W13	✓	34	LVCMOS33*	3.300	
sw_4bit_tri_i[1]	Input		P15	✓	34	LVCMOS33*	3.300	
sw_4bit_tri_i[0]	Input		G15	✓	35	LVCMOS33*	3.300	
Scalar ports (0)								
Scalar ports (0)								

Figure 18. The IP port pin constraints

3-2-9. In the *I/O Ports* pane, click in the Site column of *btns_4bit_tri_i[3]* row, enter **Y16**.

3-2-10. Click in the *I/O Std* column of the same row.

Notice that LVCMOS18 is displayed in Red under the IO Standard column. It is shown in Red because of mismatch IO standard.

3-2-11. Click on the *LVC MOS18* entry to see a pop-up window. Scroll down and select **LVC MOS33** and then hit **Enter**. The LVC MOS33 is displayed in black.

3-2-12. Similarly, assign *V16*, *P16*, and *R18* to **btns_4bit_tri_i[2]**, **btns_4bit_tri_i[1]**, and **btns_4bit_tri_i[0]**. Assign LVC MOS33 standard too.

At this stage the I/O Ports tab should look like as shown below.

GPIO_23532 (4)	Input				34 LVC MOS33*
btns_4bit_tri_i (4)	Input				34 LVC MOS33*
btns_4bit_tri_i[3]	Input	Y16	<input checked="" type="checkbox"/>		34 LVC MOS33*
btns_4bit_tri_i[2]	Input	V16	<input checked="" type="checkbox"/>		34 LVC MOS33*
btns_4bit_tri_i[1]	Input	P16	<input checked="" type="checkbox"/>		34 LVC MOS33*
btns_4bit_tri_i[0]	Input	R18	<input checked="" type="checkbox"/>		34 LVC MOS33*
Scalar ports (0)					
GPIO_48824 (4)	Input				(Multiple) LVC MOS33*
sw_4bit_tri_i (4)	Input				LVC MOS33*
sw_4bit_tri_i[3]	Input	T16	<input checked="" type="checkbox"/>		34 LVC MOS33*
sw_4bit_tri_i[2]	Input	W13	<input checked="" type="checkbox"/>		34 LVC MOS33*
sw_4bit_tri_i[1]	Input	P15	<input checked="" type="checkbox"/>		34 LVC MOS33*
sw_4bit_tri_i[0]	Input	G15	<input checked="" type="checkbox"/>		35 LVC MOS33*
Scalar ports (0)					

Figure 19. After pin assignments done to the btns_4bit_tri_i ports

3-2-13. Select **File > Save Constraints** and click **OK** when Out of Date Design warning message is displayed.

3-2-14. Click **OK** to save the constraints to *lab2.xdc* file.

Generate Bitstream and Export to SDK

Step 4

4-1. Generate the bistream, and export the hardware along with the generated bitstream to SDK.

4-1-1. Click on the **Generate Block Design > system.bd** in the Flow Navigator, to generate the output products of various IPs added to the project. Click **Generate**.

4-1-2. Click on **Generate Bitstream**, and click **Yes** if prompted to Launch Implementation (Click **Yes** if prompted to save the design)

4-1-3. Select **Open Implemented Design** option when the bitstream generation process is complete and click **OK**. (Click **Yes** if prompted to close the synthesized design.)

You should have the block design and the implemented design open (since we have a portion of the design in the PL section) before you export the hardware to SDK.

4-1-4. Start SDK by clicking **File > Export > Export Hardware for SDK**.

The export to SDK GUI will be displayed.

Note: Since we have hardware in Programmable Logic (PL) and we have generated the bitstream, the check box is selectable.

- 4-1-5. Check the **Launch SDK** box (all three should be checked) and click **OK**.

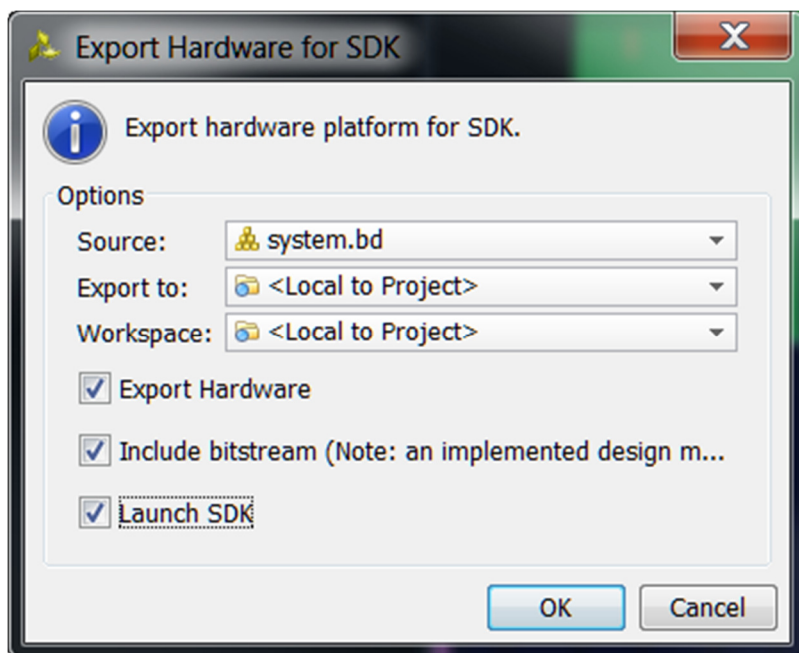


Figure 13. Export the design to SDK

- 4-1-6. Click **Yes** to overwrite the exported module (from lab 1).

Generate TestApp Application in SDK

Step 5

- 5-1. **Generate software platform project with default settings and default software project name (standalone_0).**

- 5-1-1. In SDK, right click on the `mem_test` project from the previous lab and select **Close Project**
- 5-1-2. Do the same for `mem_test_bsp` (The bsp could be reused for this project, but a new one will be created instead. The existing hardware platform project, `hw_platform_0`, has been overwritten and updated by the new export from Vivado, and will be reused for this lab.)
- 5-1-3. From the *File* menu select **File > New > Board Support Package**
- 5-1-4. Click **Finish** with the *standalone* OS selected and default project name as `standalone_bsp_0`.
- 5-1-5. Click **OK** to generate the board support package named `standalone_bsp_0`.
- 5-1-6. From the *File* menu select **File > New > Application Project**
- 5-1-7. Name the project **TestApp**, select *Use existing* board support package, select `standalone_bsp_0` and click **Next**

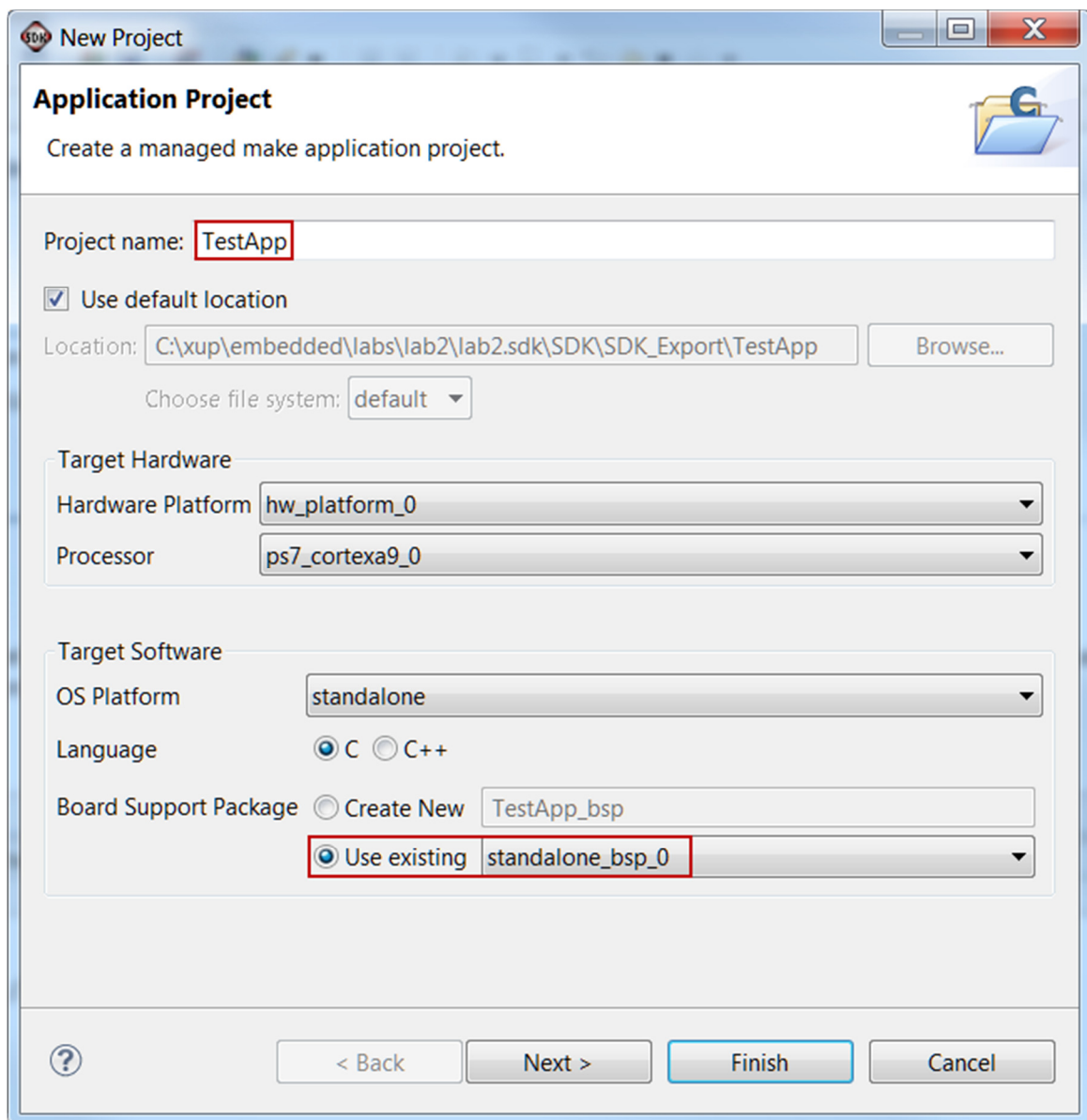


Figure 14. Application Project settings

5-1-8. Select **Empty Application and click **Finish****

This will create a new Application project using the created board support package.

5-1-9. The library generator will run in the background and will create the **xparameters.h file in the **C:\xup\embedded\labs\lab2\lab2.sdk\SDK\SDK_Export\standalone_bsp_0\ps7_cortexa9_0\include** directory**

5-1-10. Expand **TestApp in the project view, and right-click on the **src** folder, and select **Import****

5-1-11. Expand **General category and double-click on **File System****

5-1-12. Browse to **c:\xup\embedded\sources\lab2 folder.**

5-1-13. Select **lab2.c** and click **Finish**.

A snippet of the source code is shown in figure below.

```
#include "xparameters.h"
#include "xgpio.h"
#include "xutil.h"

//=====

int main (void)
{
    XGpio dip, push;
    int i, psb_check, dip_check;

    xil_printf("-- Start of the Program --\r\n");

    XGpio_Initialize(&dip, XPAR_SW_4BIT_DEVICE_ID);
    XGpio_SetDataDirection(&dip, 1, 0xffffffff);

    XGpio_Initialize(&push, XPAR_BTNS_4BIT_DEVICE_ID);
    XGpio_SetDataDirection(&push, 1, 0xffffffff);

    while (1)
    {
        psb_check = XGpio_DiscreteRead(&push, 1);
        xil_printf("Push Buttons Status %x\r\n", psb_check);
        dip_check = XGpio_DiscreteRead(&dip, 1);
        xil_printf("DIP Switch Status %x\r\n", dip_check);

        for (i=0; i<99999999; i++);
    }
}
```

Figure 15 Snippet of source code

Test in Hardware


Step 6

6-1. Make sure that the JP7 is set to select USB power. Connect the board with a micro-usb cable and power it ON. Establish the serial communication using SDK's Terminal tab.

6-1-1. Make sure that the JP7 is set to select USB power.


6-1-2. Make sure that a micro-USB cable is connected to the JTAG PROG connector (next to the power supply connector). Turn ON the power.

6-1-3. Select the  **Terminal** tab. If it is not visible then select **Window > Show view > Terminal**.

- 6-1-4. Click on  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).
- 6-2. Program the FPGA by selecting Xilinx Tools > Program FPGA and assigning system.bit file. Run the TestApp application and verify the functionality.**
- 6-2-1. Select **Xilinx Tools > Program FPGA**
- 6-2-2. Click on the **Search** button of the Bitstream field, select *system.bit*, and click **OK**.
- 6-2-3. Click **Program** to download the hardware bitstream. When FPGA is programmed, the DONE LED (green color) will be lit.
- 6-2-4. Select **TestApp** in *Project Explorer*, right-click and select **Run As > Launch on Hardware (GDB)** to download the application, execute *ps7_init*, and execute *TestApp.elf*.
- 6-2-5. 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
```

Figure 23. SDK Terminal output

- 6-2-6. Select *Console* tab and click on the *Terminate* button () to stop the program.
- 6-2-7. Close SDK and Vivado programs by selecting **File > Exit** in each program.
- 6-2-8. 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. Pin location constraints were made explicitly to connect the peripherals to the push buttons and DIP switches of the ZYBO board as the tools are not aware of the board. A TestApp application project was created and the functionality was verified after downloading the bitstream and executing the program.