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

Demonstrate your Step 6 to get credit for this lab.

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 a 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

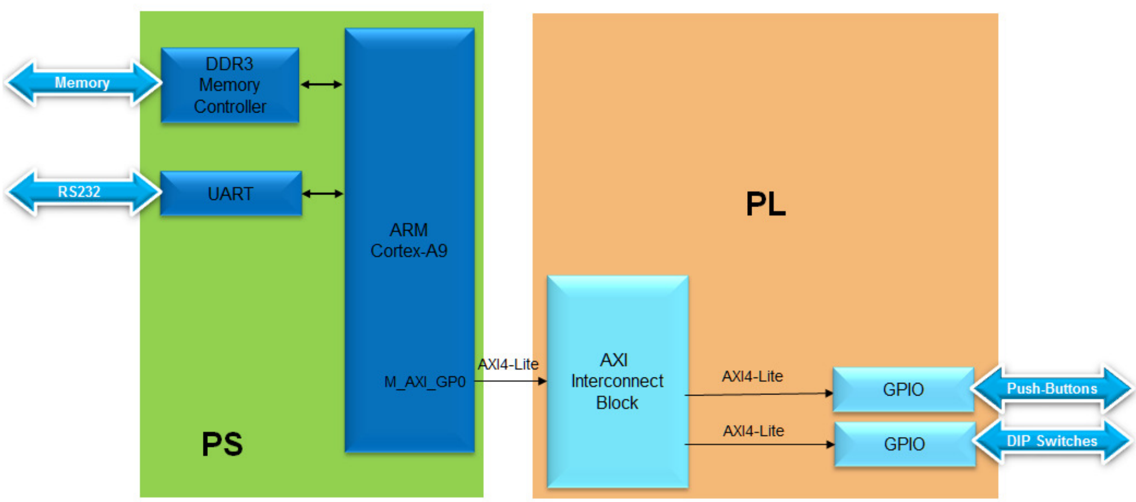
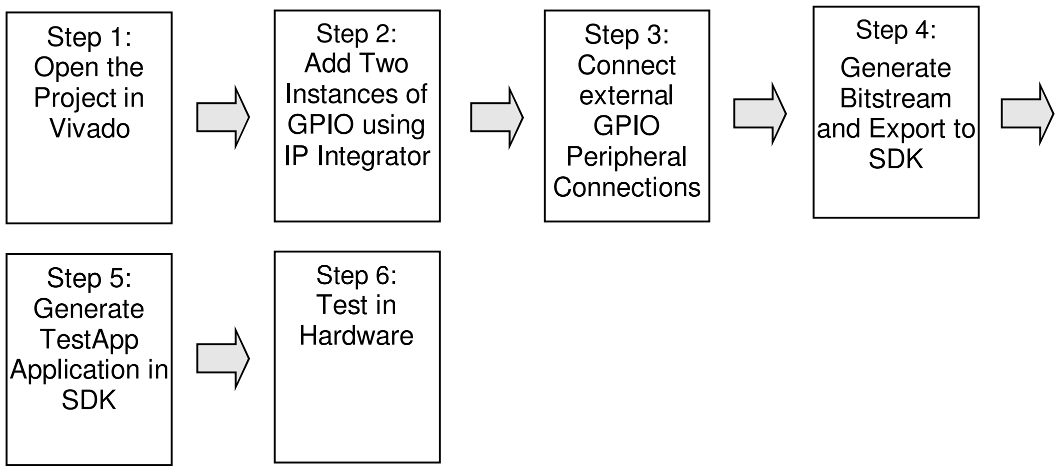


Figure 1. Extend the System from the Previous Lab

General Flow for this Lab



In the instructions below;
{***sources***} refers to: C:\xup\embedded\2015_2_zynq_sources
{***labs***} refers to : C:\xup\embedded\2015_2_zynq_labs
{***labsolutions***} for the ZedBoard refers to: C:\xup\embedded\2015_2_zedboard_labsolution
or for the Zybo refers to: C:\xup\embedded\2015_2_zybo_labsolution

1 Open the Project — Step 1

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

1-1-1. 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.

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 {labs} and click **OK**.

This will create the lab2 directory and save the project and associated directory with lab2 name.

2 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 a 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 (system_i)** 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.

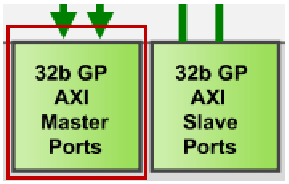


Figure 2. AXI Port Configuration

2-1-4. Expand *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.

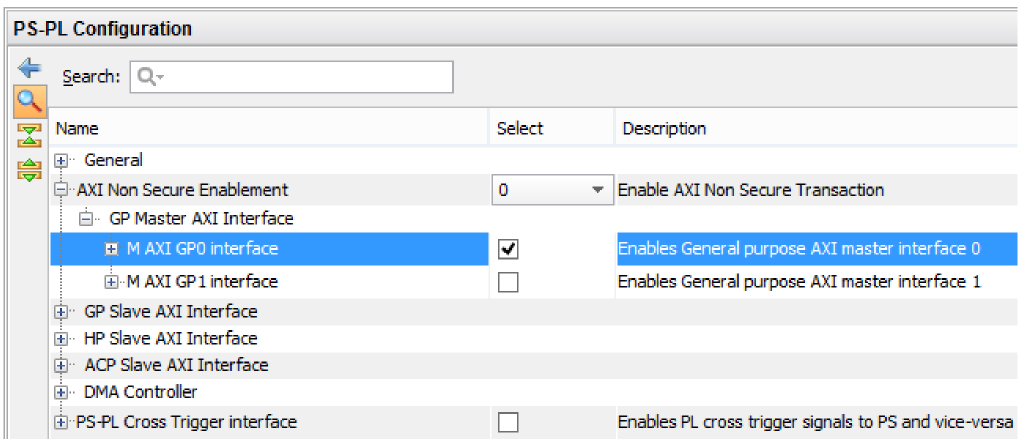


Figure 3. Configuration of 32b Master GP Block

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.

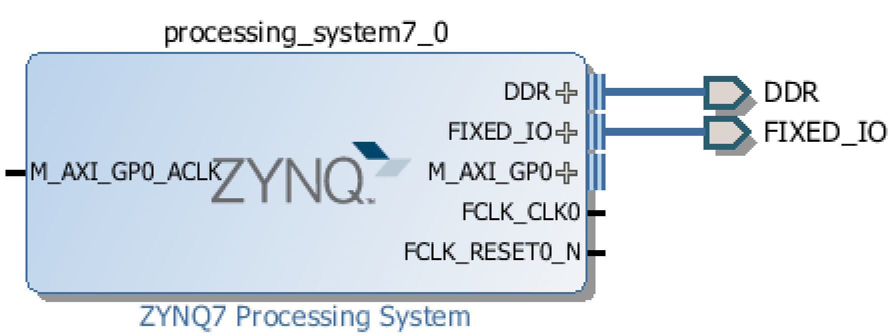


Figure 4. Zynq system with AXI and clock interfaces

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

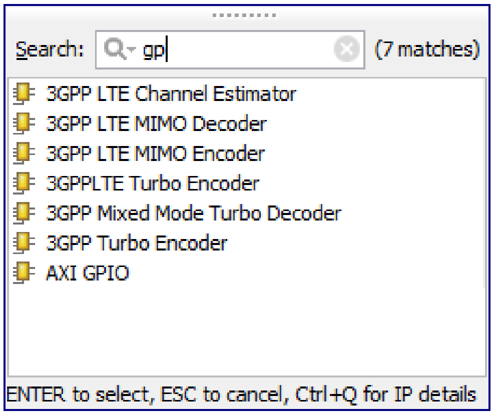


Figure 5. Add GPIO IP

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.

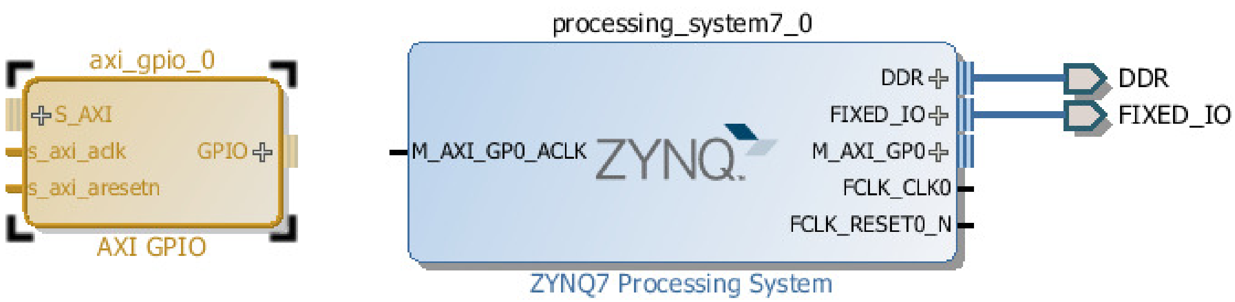
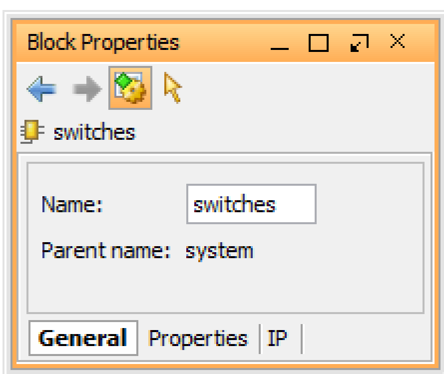


Figure 6. Zynq system with AXI GPIO added

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



Right click on AXI GPIO to open block properties

Figure 7. Change AXI GPIO default name

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

GPIO -> sws 4bits

2-1-12. From the **Board Interface** drop down, select **sws 8bits** for ZedBoard or **sws 4bits** for Zybo for GPIO.

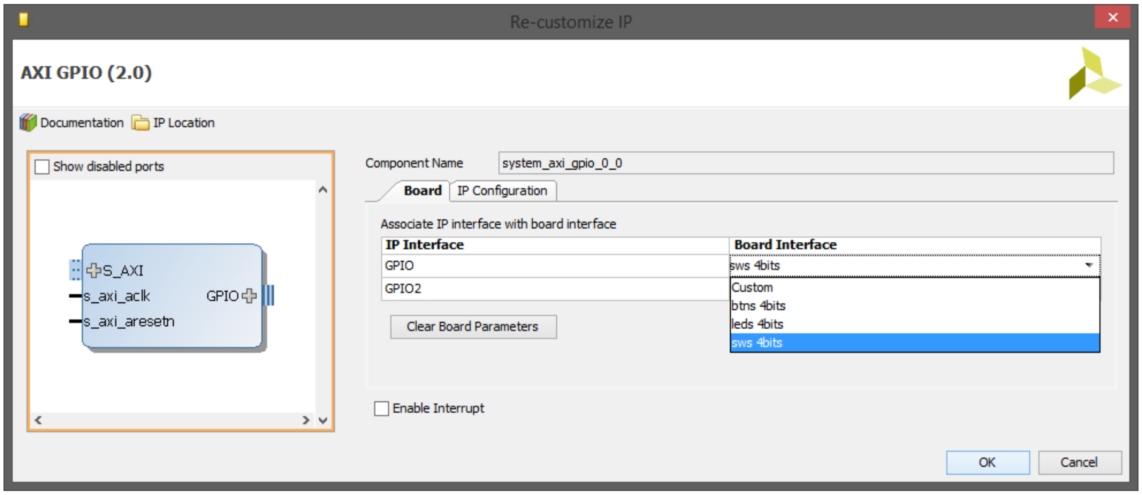


Figure 8. Configuring GPIO instance

2-1-13. Click the IP configuration tab, and notice the width has already been set to match the switches on the ZedBoard (8) or Zybo (4)

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

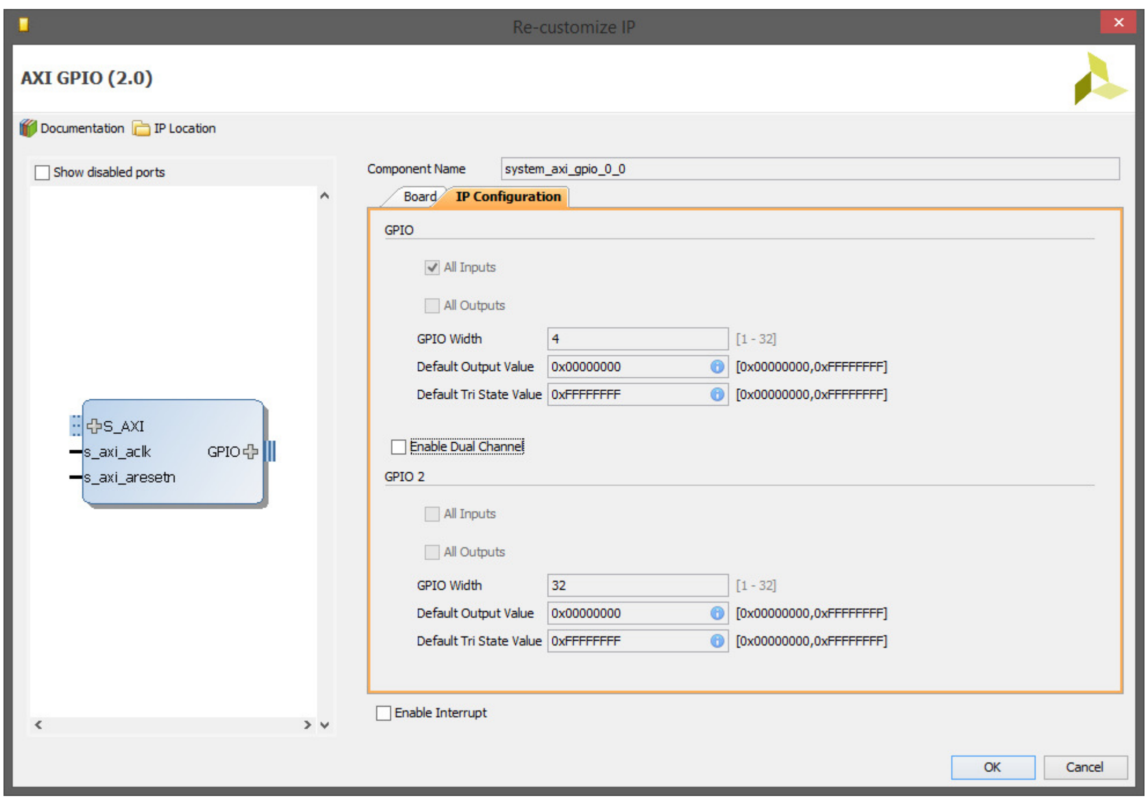


Figure 9. Configuring GPIO instance

2-1-14. Click **OK** to save and close the customization window

2-1-15. Notice that *Designer assistance* is available. Click on **Run Connection Automation**, and select **/switches/S_AXI**

Above the schematics

2-1-16. Click **OK** when prompted to automatically connect the master and slave interfaces

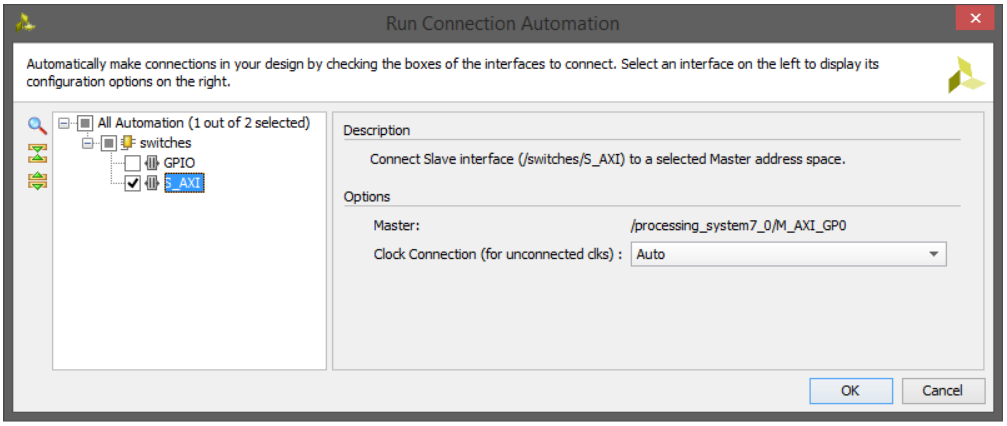


Figure 10. Run connection automation

2-1-17. 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.)

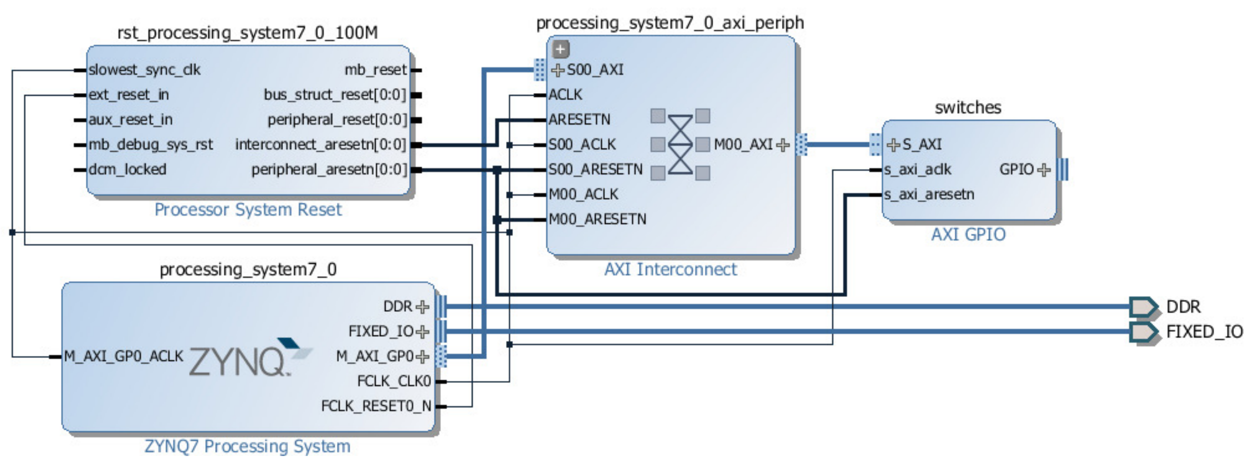


Figure 11. Design with switches automatically connected

2-1-18. Add another instance of the *GPIO* peripheral (**Add IP**). Name it as **buttons**

Add IP -> AXI GPIO. Right click to open block properties. Change name to buttons

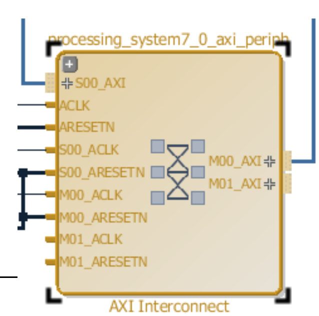
2-1-19. Double click on the IP block, select the *btns* GPIO interface (*btns_5bits* for the Zedboard, *btns_4bits* for the Zybo) and click **OK**.

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

Do not execute "Run Connection Automation.

2-1-20. Double click on the AXI Interconnect and change the *Number of Master* Interfaces to **2** and click **OK**

AXI Interconnect is one of the circuit blocks



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

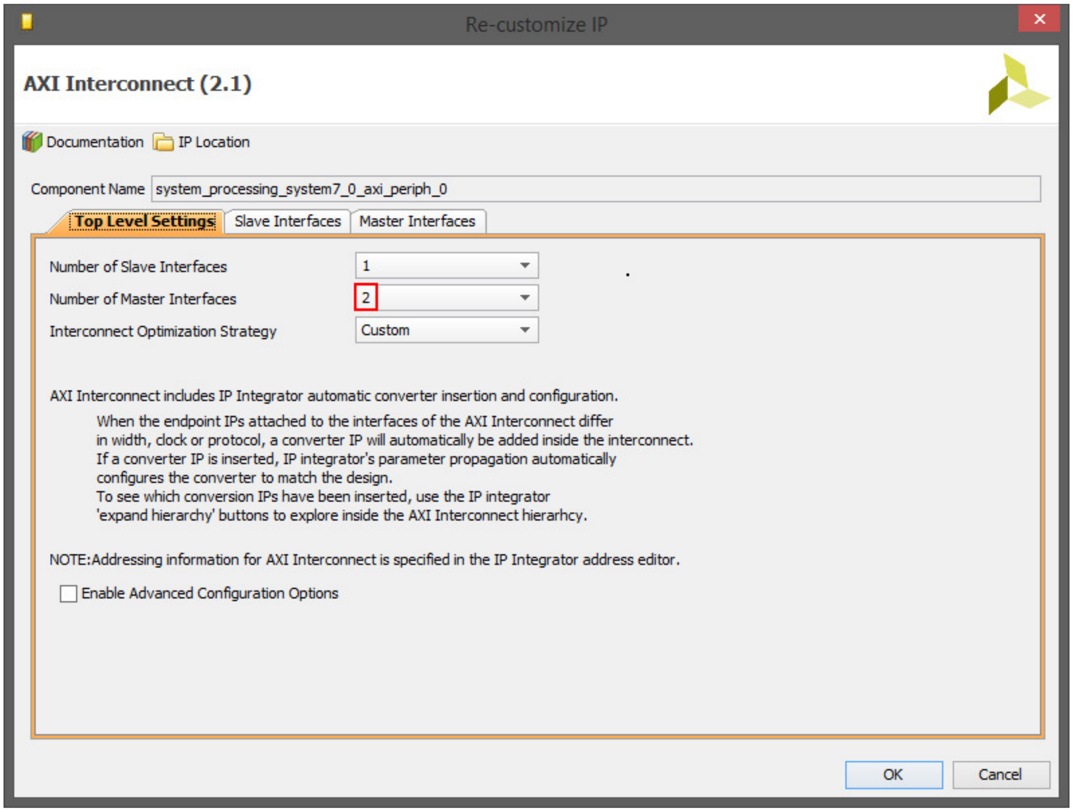


Figure 12. Add master port to AXI Interconnect

Click, hold and drag

2-1-21. Click on the *s_axi* port of the buttons 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.

2-1-22. 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:

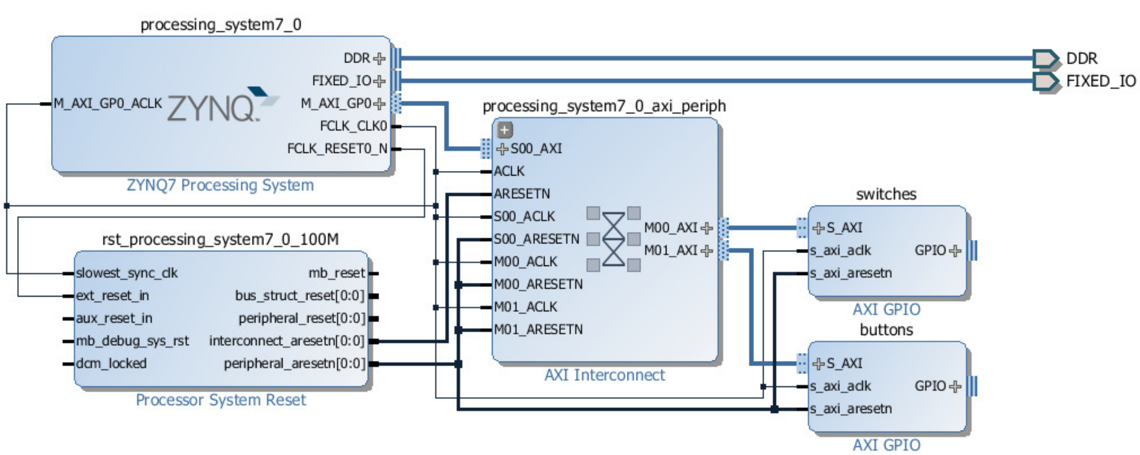


Figure 13. System Assembly View after Adding the Peripherals

2-1-23. Click on the *Address Editor* tab, and expand **processing_system7_0 > Data > Unmapped Slaves** if necessary

2-1-24. Notice that *switches* has been automatically assigned an address, but *buttons* has not (since it was manually connected). Right click on *Buttons* 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).

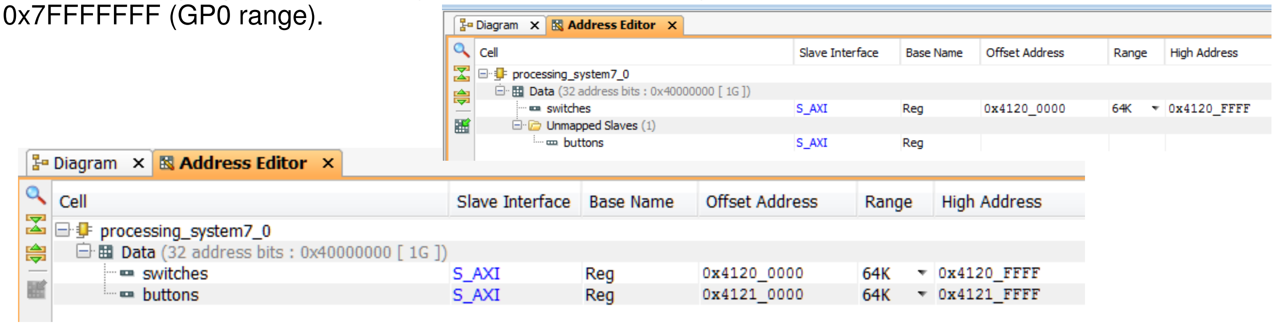


Figure 14. Peripherals Memory Map

3 Make GPIO Peripheral Connections External Step 3

3-1. The push button and dip switch instances will be connected to corresponding pins on the board. This can be done manually, or using Designer Assistance. Normally, one would consult the board's user manual to find this information.

Click Open Block Design in IP Integrator to see Diagram View

3-1-1. In the Diagram view, notice that *Designer Assistance* is available. We will manually create the ports and connect.

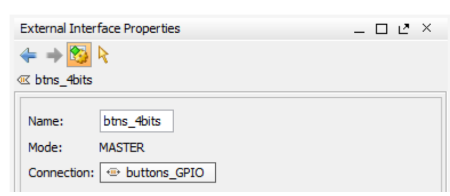
3-1-2. Right-Click on the *GPIO* port of the *switches* instance and select **Make External** to create the external port. This will create the external port named *gpio* and connect it to the peripheral. Because Vivado is "board aware", the pin constraints will be automatically applied to the port.

3-1-3. 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.

3-1-4. For the **buttons** GPIO, click on the *Run Connection Automation* link.

3-1-5. In the opened GUI, select *btns_5bits* (for ZedBoard) or *btns_4bits* (for Zybo) under the options section.



3-1-6. Click **OK**.

3-1-7. Select the created external port and change its name as **buttons**

3-1-8. Run Design Validation (**Tools -> Validate Design**) and verify there are no errors.

The design should now look similar to the diagram below

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

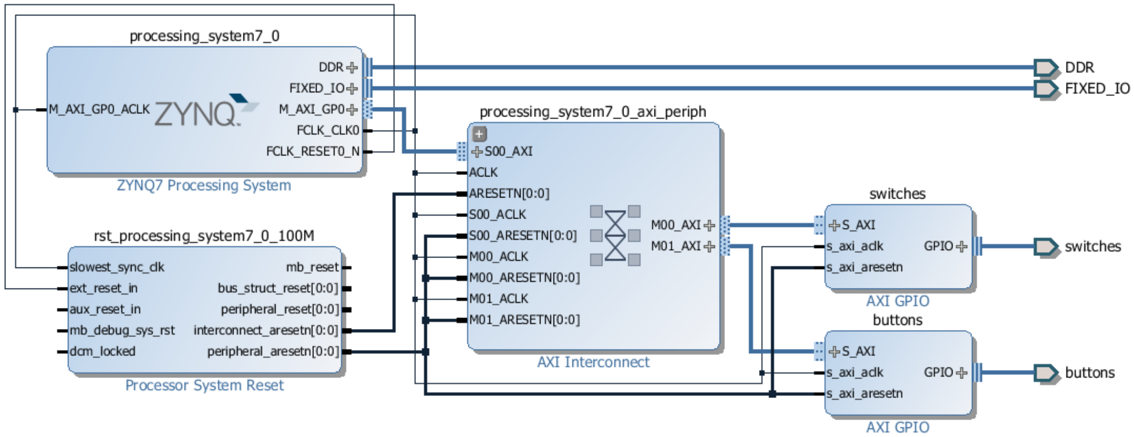
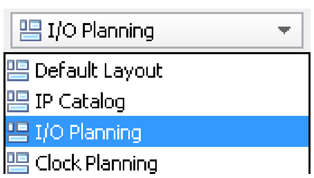


Figure 15. Completed design

3-2. Synthesize the design, open the I/O Planning layout, and check the constraints using the I/O planning tool.

3-2-1. In the Flow Navigator, click **Run Synthesis**. (Click **Save** if prompted) and when synthesis completes, click Cancel to cancel "Run Impplementation. Select **Open Synthesized Design**,.

3-2-2. Select **I/O Planning** from the *Layout* dropdown menu



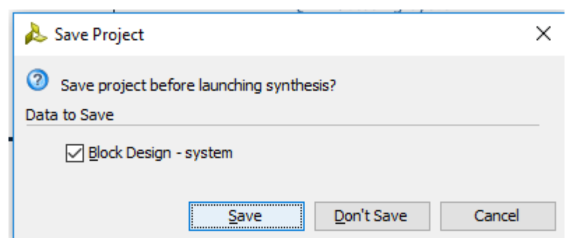
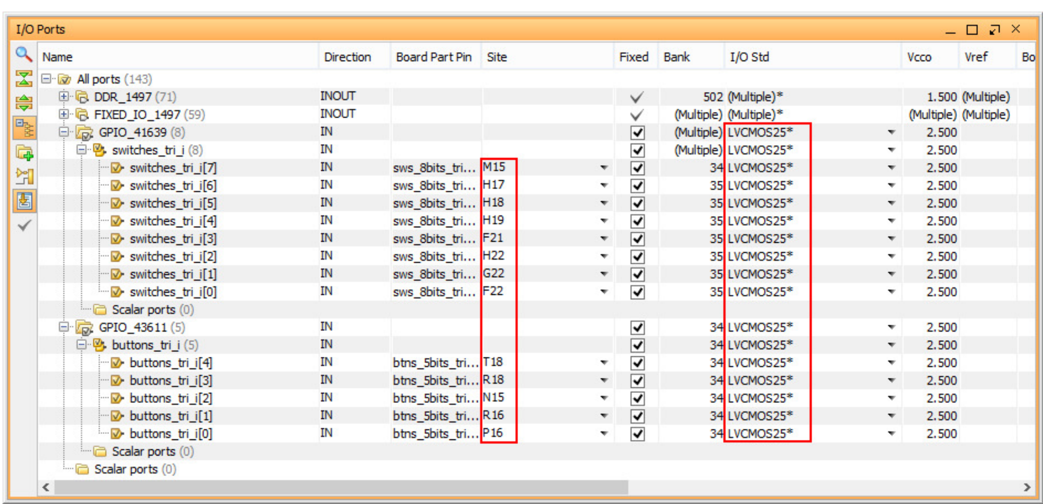


Figure 16. Switch to the IO planning view

3-2-3. 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) 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.


I/O Ports

Name	Direction	Board Part Pin	Site	Fixed	Bank	I/O Std	Vcco	Vref	Bo
All ports (143)									
DDR_1497 (71)	INOUT			✓	502	(Multiple)*	1.500	(Multiple)	
FIXED_IO_1497 (59)	INOUT			✓	(Multiple)	(Multiple)*	(Multiple)	(Multiple)	
GPIO_41639 (8)	IN			✓	(Multiple)	LVCMOS25*	2.500		
switches_tri_i (8)	IN			✓	(Multiple)	LVCMOS25*	2.500		
switches_tri_i[7]	IN	sws_8bits_tri...	M15	✓	34	LVCMOS25*	2.500		
switches_tri_i[6]	IN	sws_8bits_tri...	H17	✓	35	LVCMOS25*	2.500		
switches_tri_i[5]	IN	sws_8bits_tri...	H18	✓	35	LVCMOS25*	2.500		
switches_tri_i[4]	IN	sws_8bits_tri...	H19	✓	35	LVCMOS25*	2.500		
switches_tri_i[3]	IN	sws_8bits_tri...	F21	✓	35	LVCMOS25*	2.500		
switches_tri_i[2]	IN	sws_8bits_tri...	H22	✓	35	LVCMOS25*	2.500		
switches_tri_i[1]	IN	sws_8bits_tri...	G22	✓	35	LVCMOS25*	2.500		
switches_tri_i[0]	IN	sws_8bits_tri...	F22	✓	35	LVCMOS25*	2.500		
Scalar ports (0)									
GPIO_43611 (5)	IN			✓	34	LVCMOS25*	2.500		
buttons_tri_i (5)	IN			✓	34	LVCMOS25*	2.500		
buttons_tri_i[4]	IN	btns_5bits_tri...	T18	✓	34	LVCMOS25*	2.500		
buttons_tri_i[3]	IN	btns_5bits_tri...	R18	✓	34	LVCMOS25*	2.500		
buttons_tri_i[2]	IN	btns_5bits_tri...	N15	✓	34	LVCMOS25*	2.500		
buttons_tri_i[1]	IN	btns_5bits_tri...	R16	✓	34	LVCMOS25*	2.500		
buttons_tri_i[0]	IN	btns_5bits_tri...	P16	✓	34	LVCMOS25*	2.500		
Scalar ports (0)									
Scalar ports (0)									

Figure 17. The IP port pin constraints for the ZedBoard

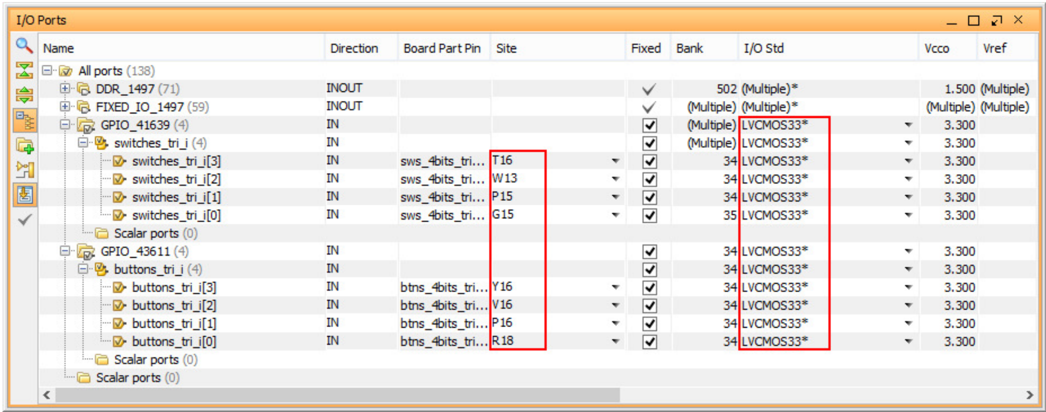


Figure 18. The IP port pin constraints for the Zybo

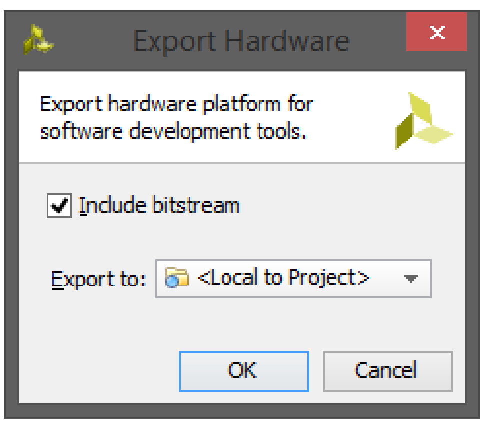
4Generate Bitstream and Export to SDK Step 4

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

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

4-1-2. Click **Cancel** to cancel "Open Implemented Design"

4-1-3. 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.



if in Debug mode, right click on Debug to close it

Figure 19. Export the design

4-1-4. Click **Yes** to overwrite the hardware module.

4-1-5. Start SDK by clicking **File > Launch SDK** and click **OK**

5Generate TestApp Application in SDK Step 5

5-1. Close the projects from the previous lab. 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* and *system_wrapper_hw_platform_0*

5-1-3. From the *File* menu select **File** > **New** > **Board Support Package**

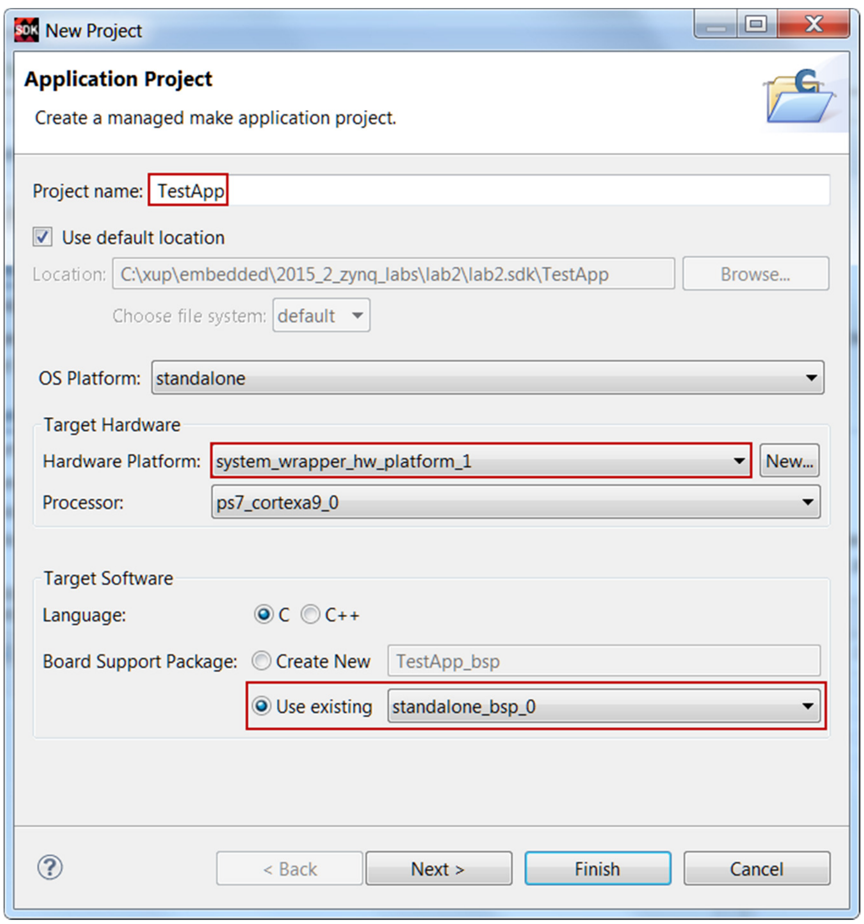
5-1-4. Click **Finish** with the *standalone* OS selected and default project name as *standalone_bsp_0.*

5-1-5. Click **OK** to generate the board support package named *standalone_bsp_0. Click **OK** on Board Support Package Settings window.*

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**

Another way to remove previous SDK files is to close this SDK, open this project folder and delete the



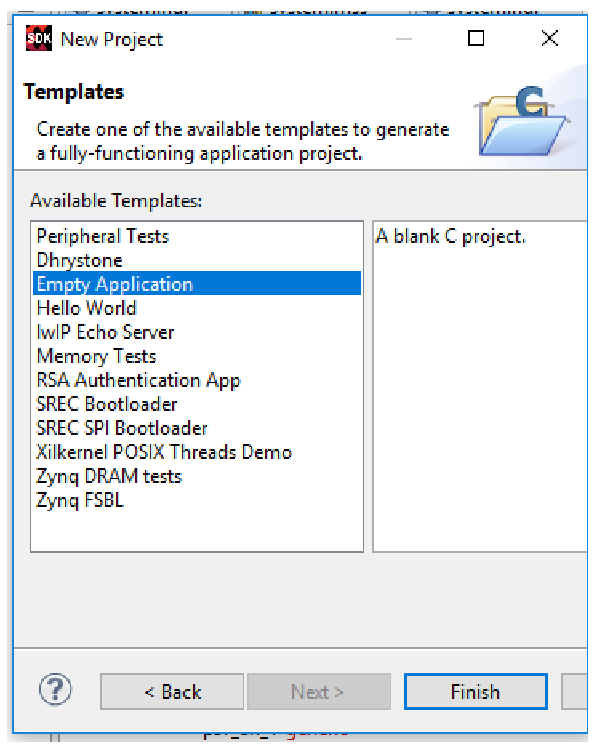


Figure 20. Application Project settings

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 **lab2\lab2.sdk\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 the {sources}**lab2** folder

C:\xup\embedded
\2015_2_zynq_sources\lab2

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

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

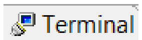
```
#include "xparameters.h"
#include "xgpio.h"


//====================================================

int main (void)
{

    XGpio dip, push;
    int psb_check, dip_check;

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

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

    XGpio_Initialize(&push, XPAR_BUTTONS_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);

      sleep(1);
    }

}
```

Figure 21. Snippet of source code

6Test in Hardware Step 6

6-1. Connect the board with a micro-usb cable(s) and power it ON. Establish the serial communication using SDK's Terminal tab.

6-1-1. Make sure that micro-USB cable(s) is(are) connected between the board and the PC. Turn ON the power

6-1-2. Select the  tab. If it is not visible then select **Window > Show view > Terminal**

6-1-3. 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_wrapper.bit file. Run the TestApp application and verify the functionality

6-2-1. Select **Xilinx Tools** > **Program FPGA**

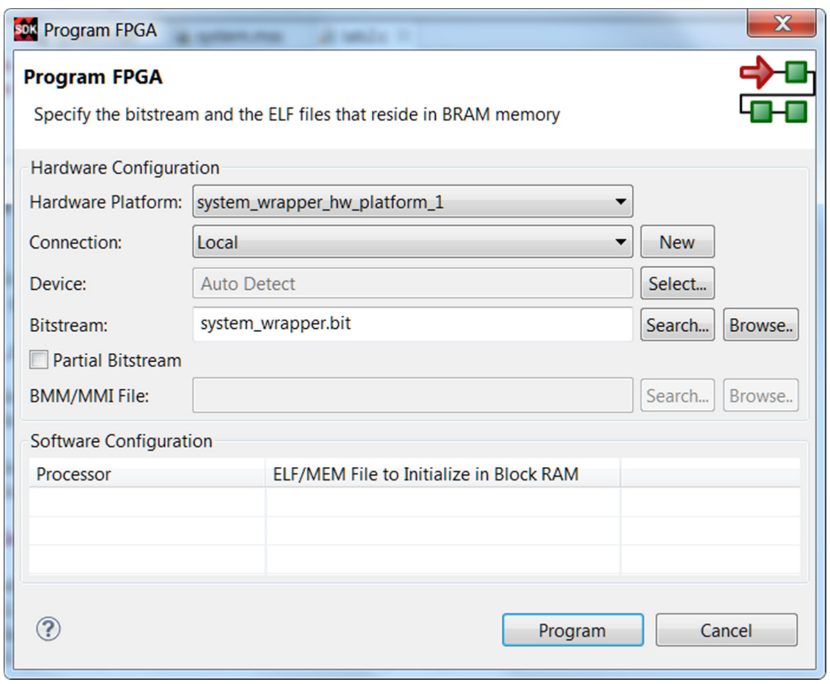


Figure 22. Program FPGA

6-2-2. 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

6-2-3. 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

execute ps7_init, and execute TestApp.elf are automatic.

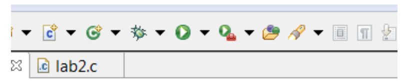
6-2-4. You should see the something similar to the following output on Terminal console

Select Terminal 1 to see the output

```
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

Click on the green go item on top of the SDK to start executing TestApp.elf again.



6-2-5. Select *Console* tab and click on the *Terminate* button ( ) to stop the program

6-2-6. Close SDK and Vivado programs by selecting **File > Exit** in each program

6-2-7. Power OFF the board

Conclusion

Demonstrate your Step 6 to get credit for this lab.

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.