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

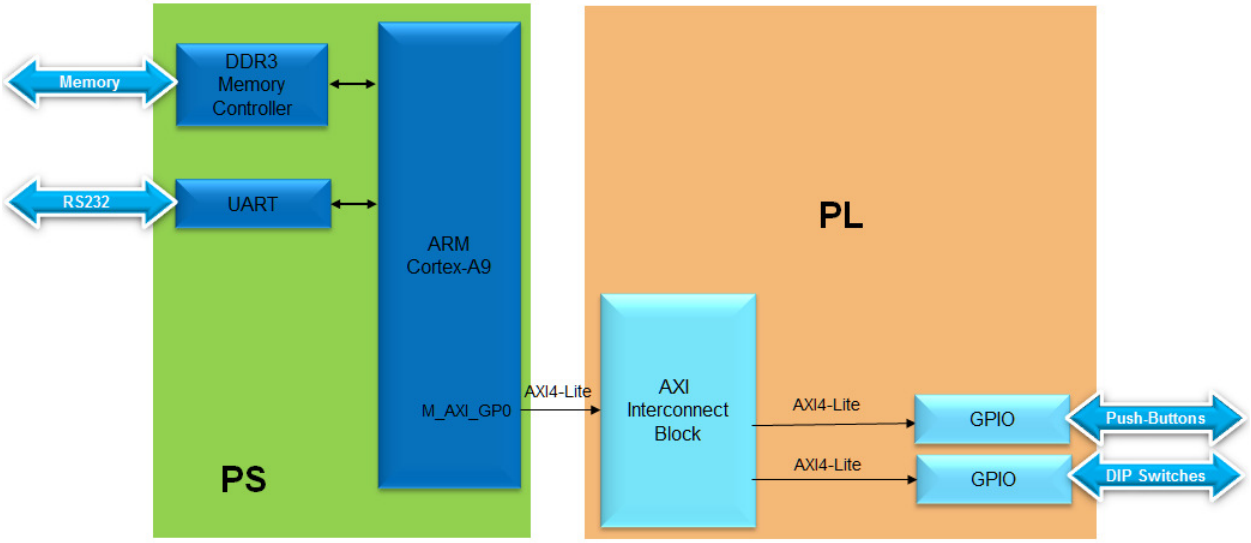
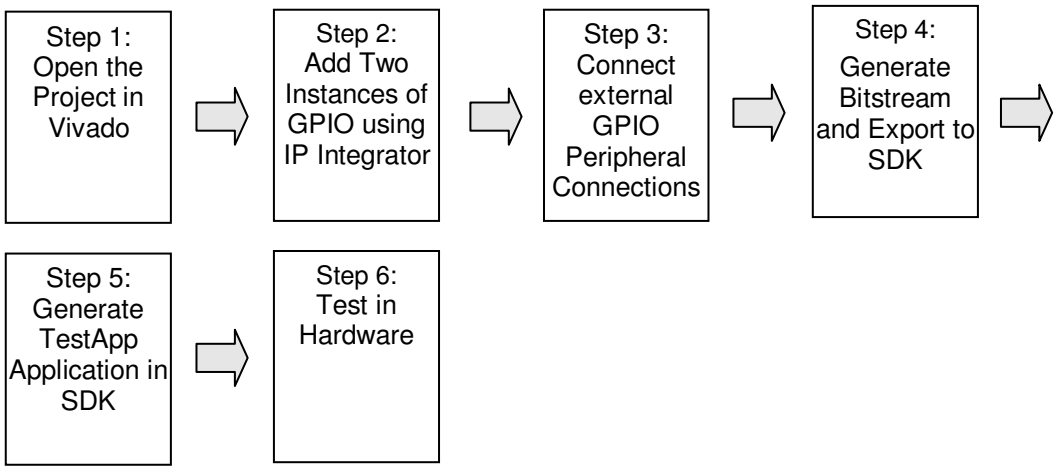


Figure 1 Extend the System from the Previous Lab

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.

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

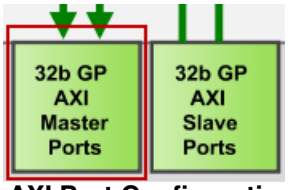


Figure 2 AXI Port Configuration

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.

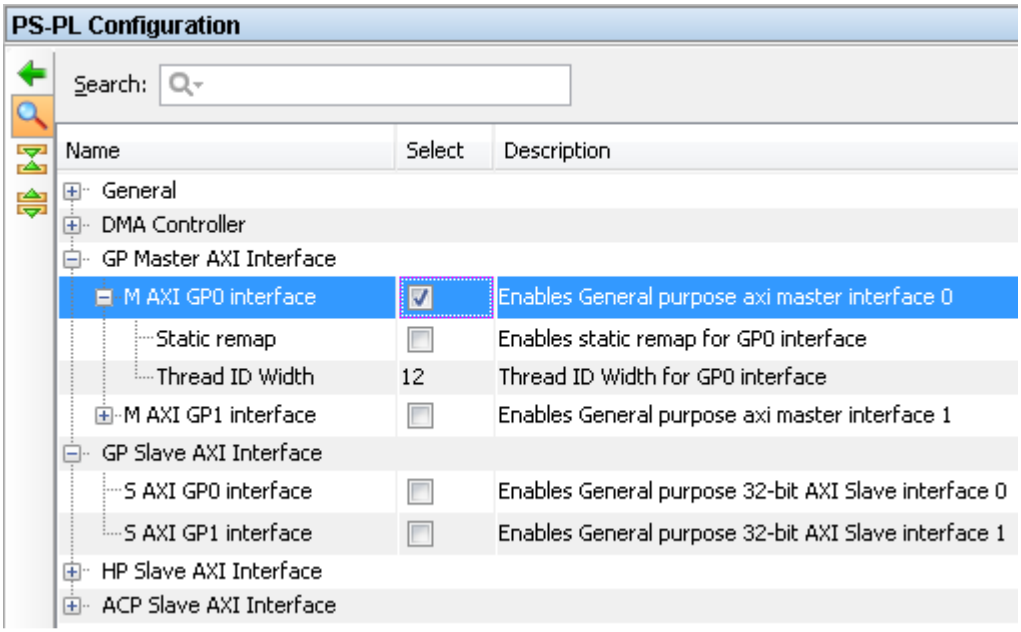


Figure 3 Configuration of 32b Master GP Block

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.

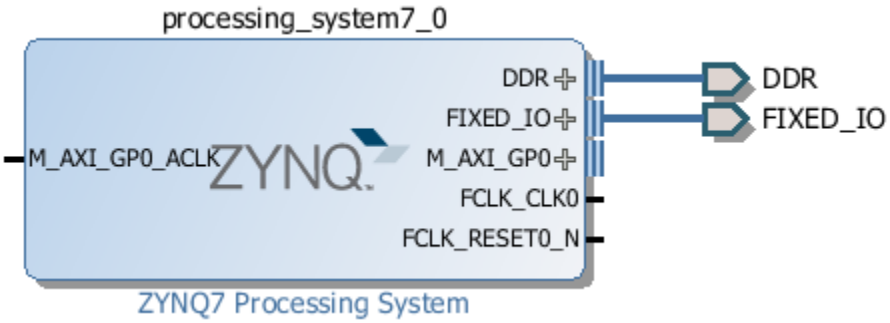


Figure 4 Zynq system with AXI and clock interfaces

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

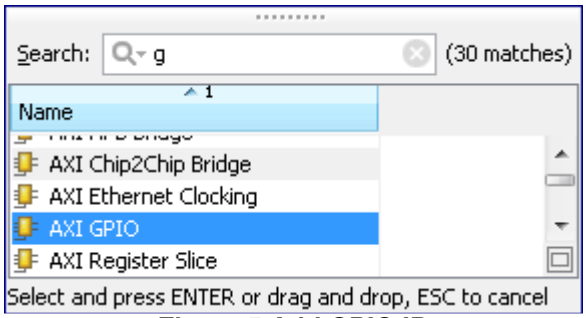


Figure 5 Add GPIO IP

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.

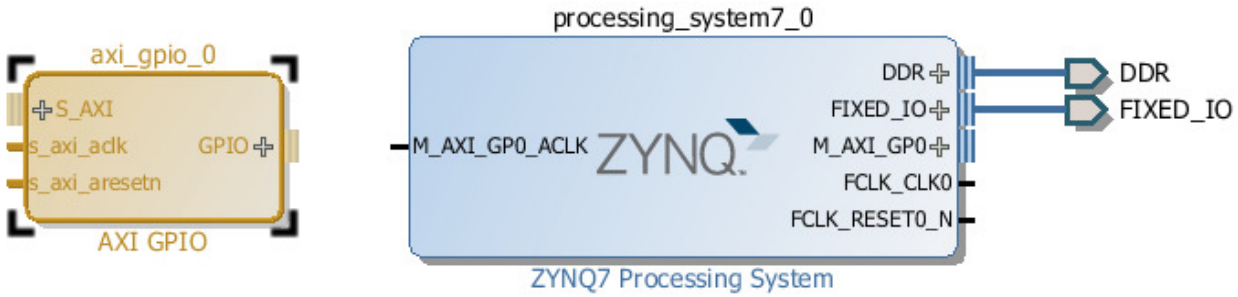


Figure 6 Zynq system with AXI GPIO added

2-1-10. Click on the **AXI GPIO** block to select it, and in the properties tab, change the name to **sw_8bit**

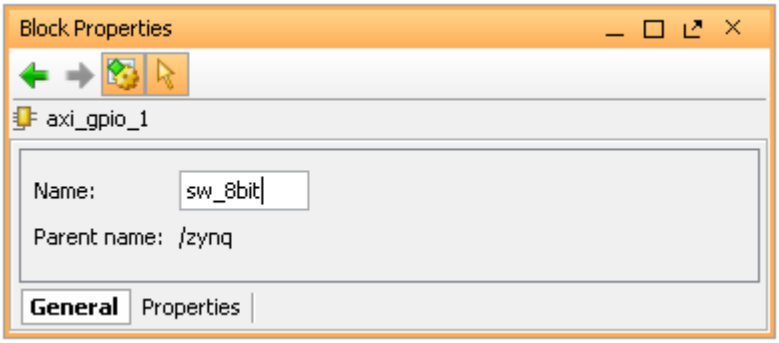


Figure 7 Change AXI GPIO default name

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

As the Zedboard was selected during the project creation in lab1, and a board support package is available for the Zedboard, Vivado has knowledge of available resources on the board.

2-1-12. Click on **Generate Board Based IO Constraints**, and under *Board Interface*, for *GPIO*, click on **Custom** to view the dropdown menu options, and select **sws_8bits**

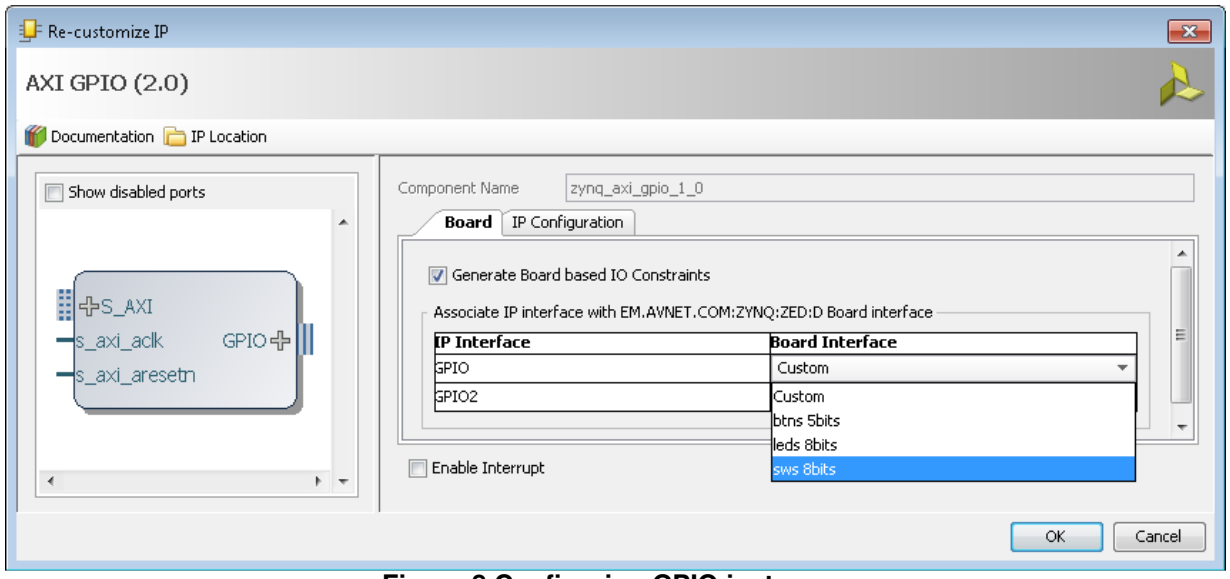


Figure 8 Configuring GPIO instance

2-1-13. Click the *IP Configuration* tab. Notice the GPIO Width is set to 8.

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*.

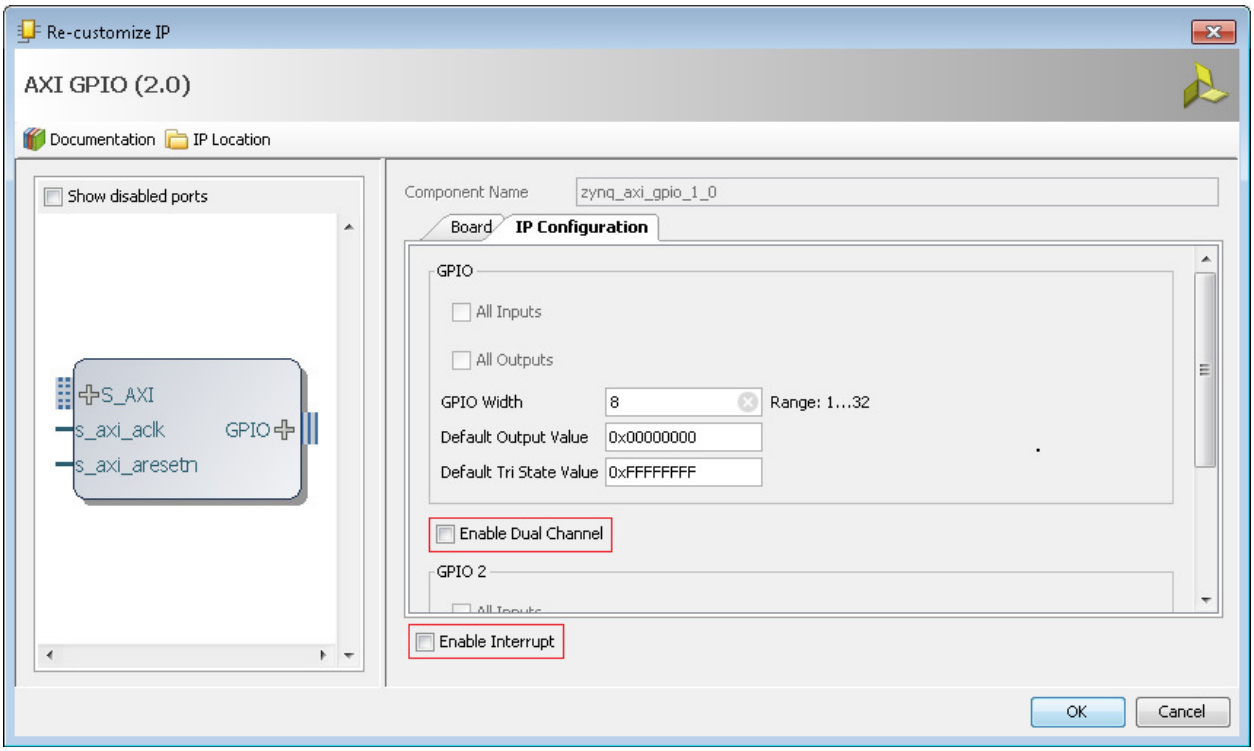


Figure 9 Configuring GPIO instance

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

2-1-15. Notice that *Design assistance* is available. Click on **Run Connection Automation**, and select **/sw_8bit/S_AXI**

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

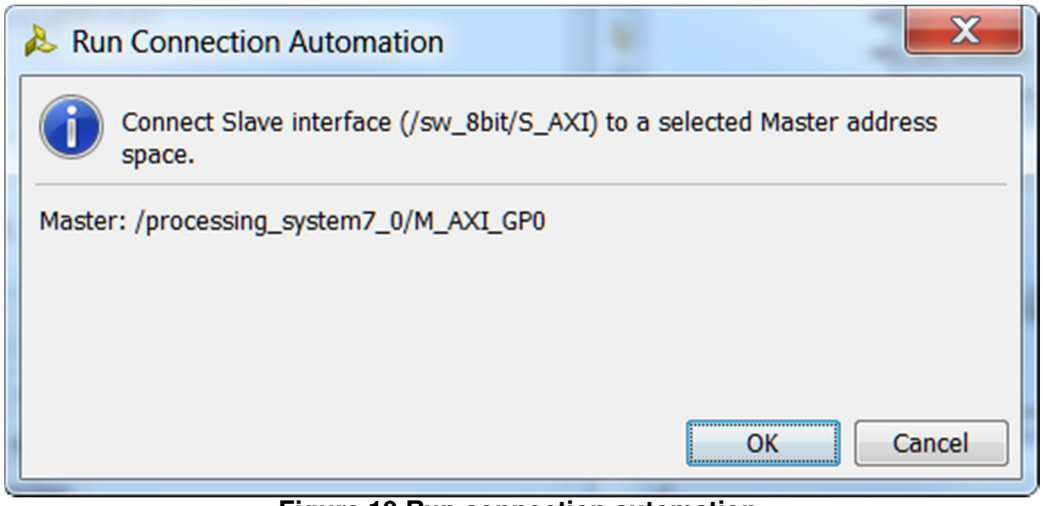


Figure 10 Run connection automation

2-1-17. Notice two additional blocks, *Proc Sys 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.)

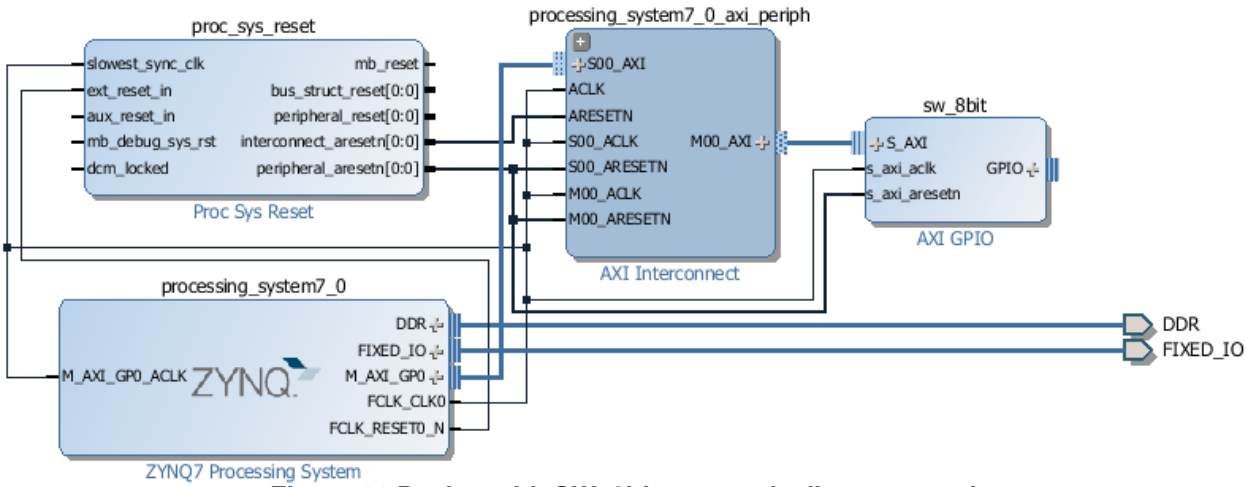


Figure 11 Design with SW_8bit automatically connected

2-1-18. Add another instance of the *GPIO* peripheral (**Add IP**), and using the board flow, configure it to connect to the **btns_5bit**

2-1-19. Change the name of the block to **btns_5bit** (Click on the block to select it, and change the name in the properties view)

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

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

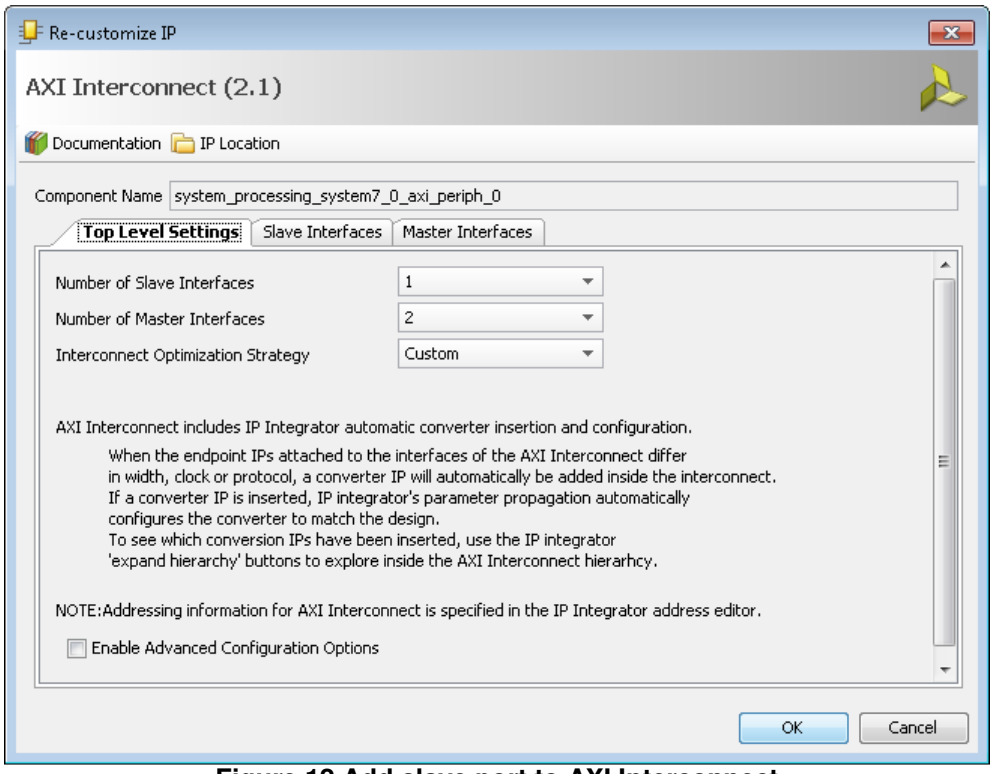


Figure 12 Add slave port to AXI Interconnect

2-1-21. 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.

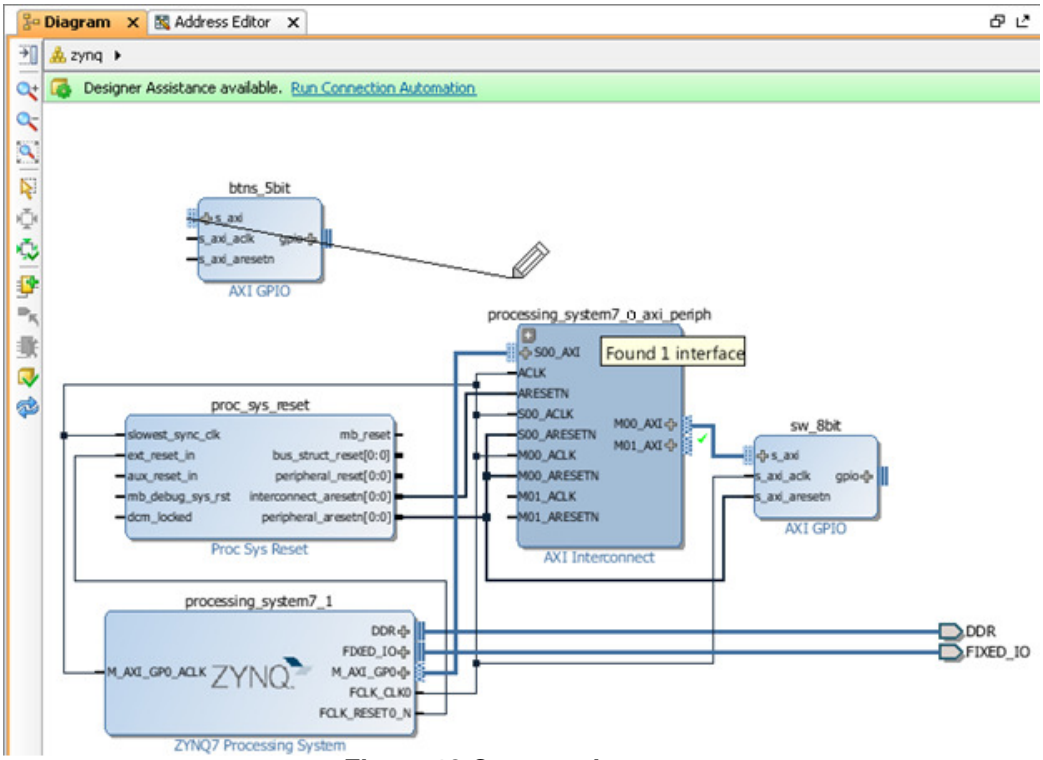


Figure 13 Connect the ports

2-1-22. In a similar way, connect the following ports:
btns_5bit **s_axi_aclk** -> *Zynq7 Processing System* **FCLK_CLK0**
btns_5bit **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:

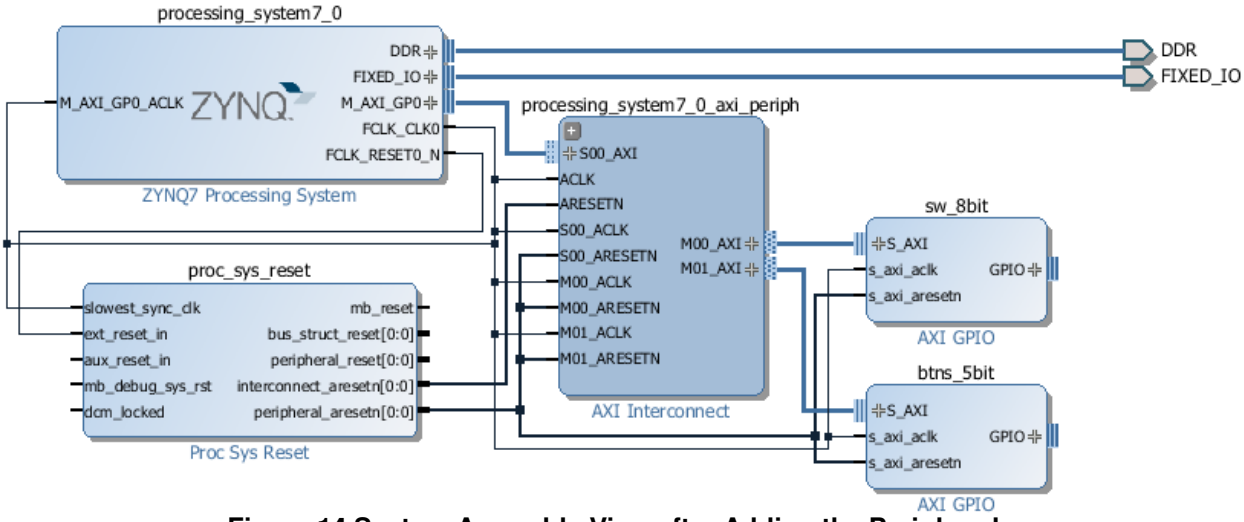


Figure 14 System Assembly View after Adding the Peripherals

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

2-1-24. Notice that sw_8bit has been automatically assigned an address, but btns_5bit has not. Right click on *btns_5bit* and select **Assign Address**

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

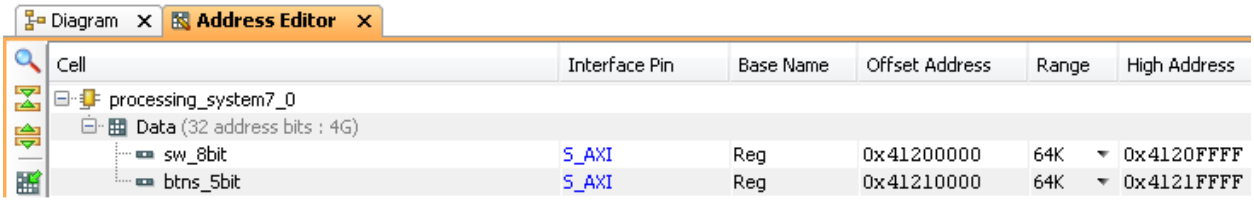


Figure 15 Peripherals Memory Map

Make GPIO Peripheral Connections External Step 3

3-1. The push button and dip switch instances will be connected to corresponding pins on the ZedBoard. This can be done manually, or using Designer Assistance. The location constraints are automatically applied by the tools as the information for the ZedBoard is already known. Normally, one would consult the ZedBoard user manual to find this information.

3-1-1. In the Diagram view, notice that *Designer Assistance* is available. This will be ignored for now, and a port will be manually created and connected for the sw_8bit instance. Designer Assistance will be used to connect the *btns_5bit* peripheral.

3-1-2. Right-Click on the gpio port of the *sw_8bit* 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_8bit** in its properties form.

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

3-1-4. Connection automation will be used to create a port for the btns_5bit block. Add the port for the *btns_5bit* component automatically, by clicking on *Run Connection Automation*, and selecting */btns_5bit/GIO*

3-1-5. In the *Select Board Interface* drop down menu, select **btns_5bits**, and click **OK** to create and connect the external port.

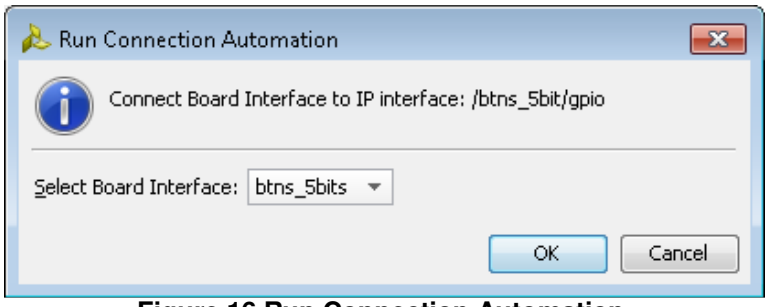


Figure 16 Run Connection Automation

3-1-6. Run Design Validation (**Tools -> Validate Design**) and verify there are no errors.

The design should now look similar to the diagram below

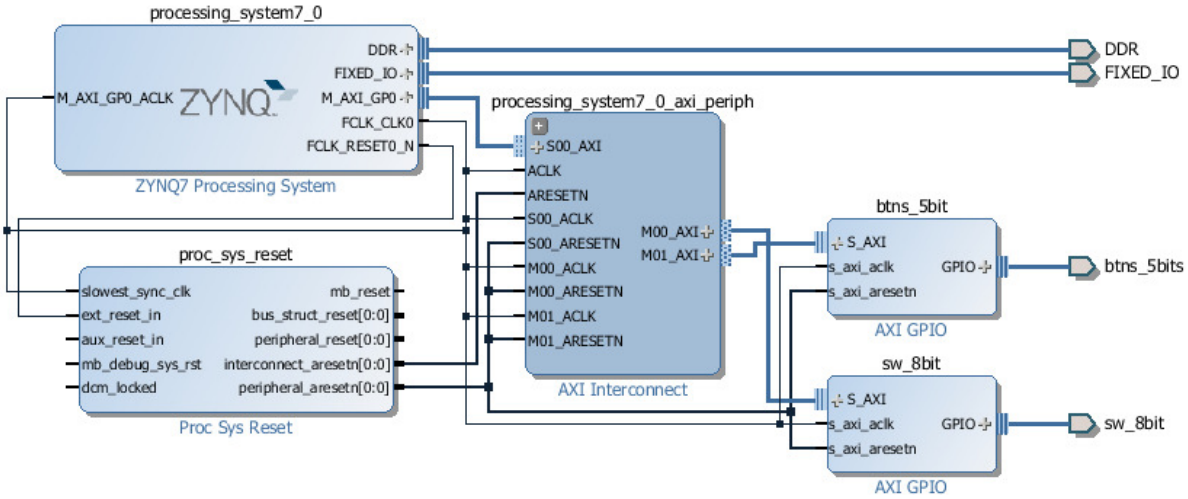


Figure 17 Completed design

3-1-7. 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**

3-1-8. In the Flow Navigator, click **Run Synthesis**. (Click **Save** when prompted) and when synthesis completes, select **Open Synthesized Design** and click **OK**

3-1-9. In the shortcut Bar, select **I/O Planning** from the *Layout* dropdown menu

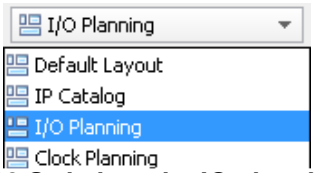


Figure 18 Switch to the IO planning view

3-1-10. In the I/O ports tab, expand *BTNs_5bit_tri_i*, and notice pins have already been assigned to this peripheral. The pin information was included in the board support package, and automatically assigning when the IP was automatically connected to the port. The *sw_8bit_tri_i* have also been automatically assigned pin locations, along with the other Fixed ports in the design.

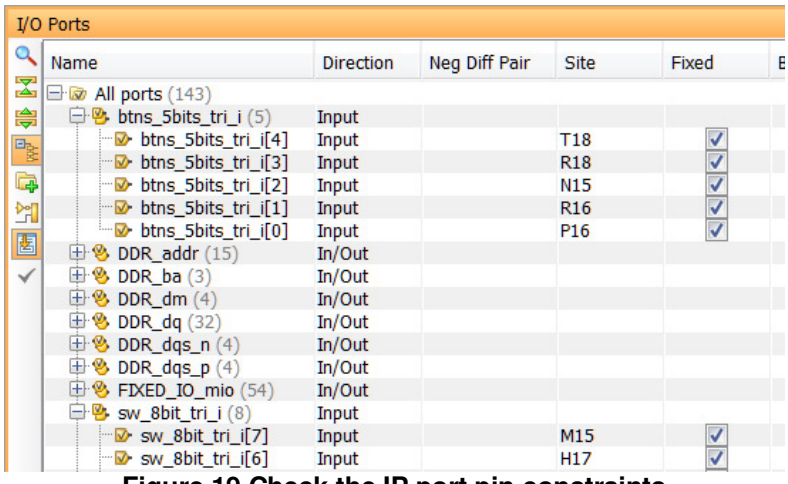


Figure 19 Check the IP port pin constraints

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 **Generate Bitstream**, and click **Yes** if prompted to Launch Implementation (Click **Yes** if prompted to save the design)

4-1-2. 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.

4-1-3. 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.

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

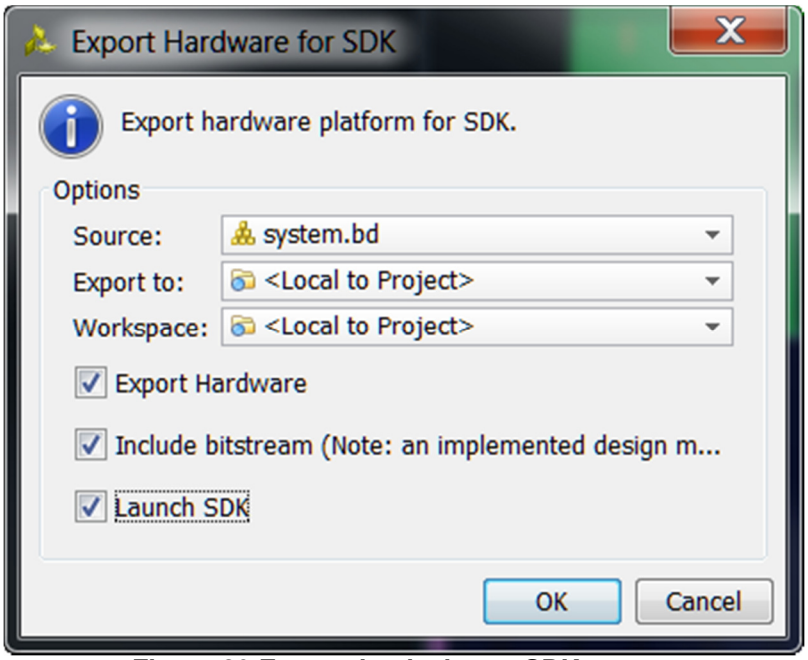


Figure 20 Export the design to SDK

4-1-5. 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.

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. Change the name to **standalone_bsp** and click **Finish** with the *standalone* OS selected.

5-1-5. Click **OK** to generate the board support package named standalone_bsp.

5-1-6. From the *File* menu select **File** > **New** > **Application Project**

5-1-7. Name the project **TestApp** and in the *Board Support Package* section, select *Create New* and type the name **standalone_bsp** and click **Next**

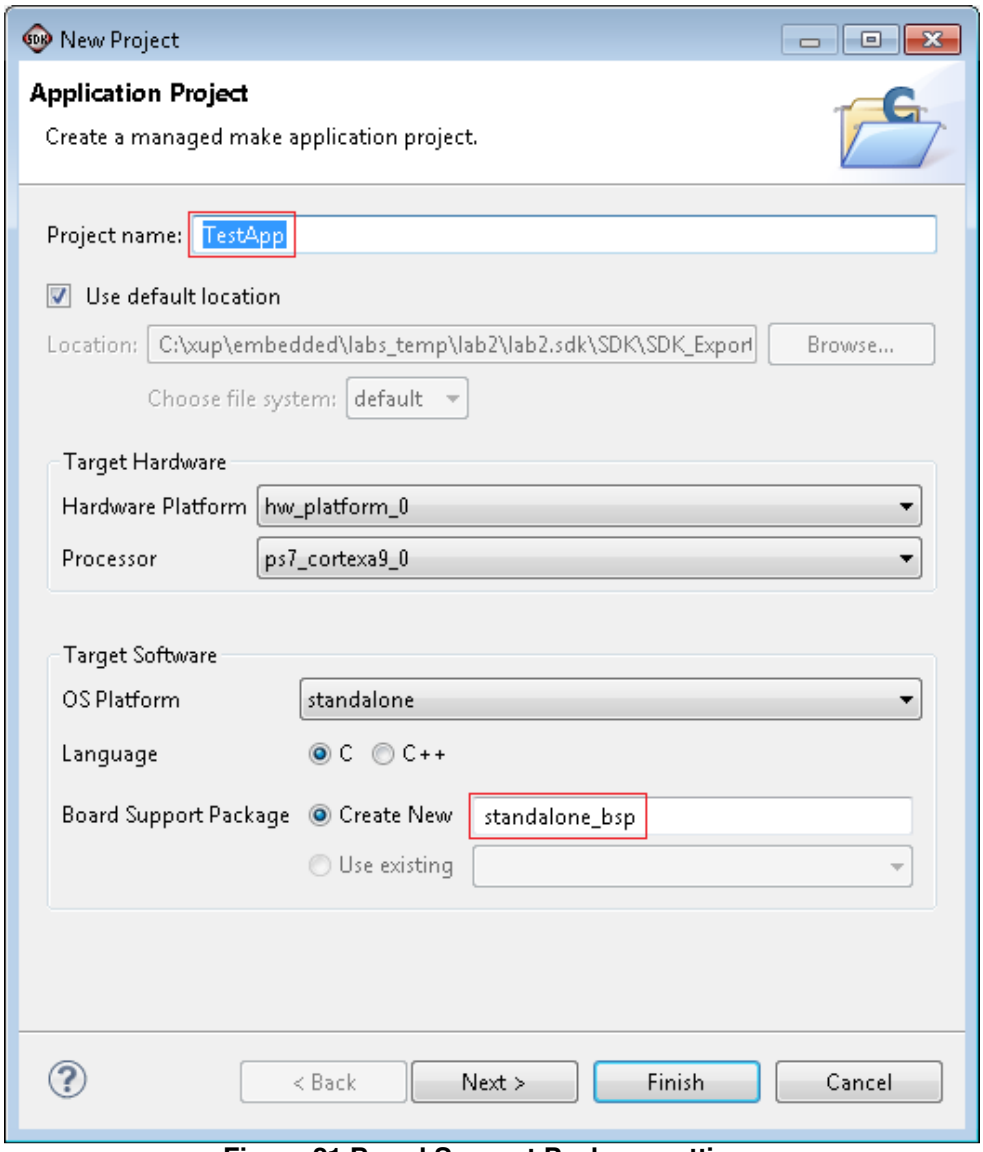


Figure 21 Board Support Package settings

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

This will create a new Application project, and a new Board Support Package Project

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\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_8BIT_DEVICE_ID);
    XGpio_SetDataDirection(&dip, 1, 0xffffffff);

    XGpio_Initialize(&push, XPAR_BTNS_5BIT_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<9999999; i++);
    }

}
```

Figure 22 Snippet of source code

5-1-14. Right click on *standalone_bsp* and select **Board support package settings**.

5-1-15. Select drivers and click in the *Driver* column for *btns_5bit* (where is currently shows generic) and select **gpio**

5-1-16. Do the same for *sw_8bit*

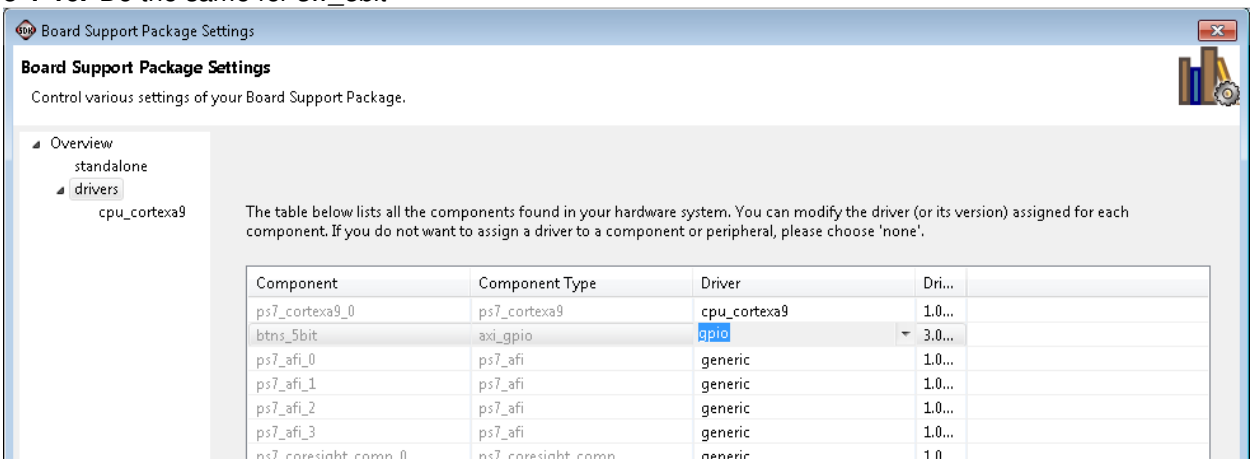


Figure 23 Select the GPIO driver

Test in Hardware Step 6

6-1. Connect and power up the board. Establish the serial communication using SDK's Terminal tab.

6-1-1. Connect and power up the ZedBoard.

6-1-2. Select the Terminal 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. Run the TestApp application and verify the functionality.

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

6-2-2. Click **Program** to download the hardware bitstream. When FPGA is programmed, the DONE LED (blue color) will be lit.

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.

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

```
Push Buttons Status 10
DIP Switch Status 28
Push Buttons Status 10
DIP Switch Status 28
Push Buttons Status 10
DIP Switch Status 28
Push Buttons Status 10
DIP Switch Status 28
```

Figure 24 SDKTerminal Output

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

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, since we used the pre-defined port names, were automatically applied to connect the peripherals to the push buttons and DIP switches of the ZedBoard. A TestApp application project was created and the functionality was verified after downloading the bitstream and executing the program.