
Embedded System Design on PYNQ - Hardcore Processors

Due Date: By beginning of next lab section

Objectives

- To become familiar with the IP integrator
- To implement a hardcore ARM Cortex-A9 processor on an FPGA
- To become familiar with the Vivado SDK (Software Development Kit)
- To create an embedded system application

Part I – Hardware and Software Processor System Design

In this part, you will create a simple hardcore ARM Cortex-A9 based embedded system on the PYNQ board. The embedded system design is broken up into three parts: Hardware design of the ARM Cortex-A9 hardcore processor, application software design using SDK, and finally hardware implementation of the software running on the hardcore processor. The application you will design in this part is a UART application that prints **"HELLO WORLD"** to a terminal emulator like ***Tera Term***. Use the figure below as a flow guide for this lab. The diagram for the completed design is shown in Figure 2.

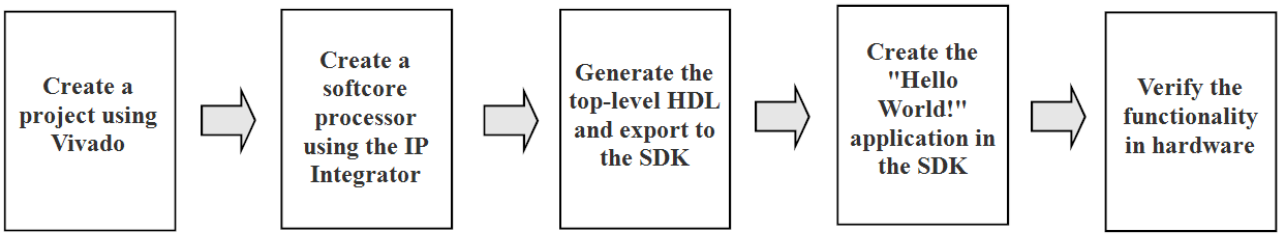

Figure 1: General flow for this lab

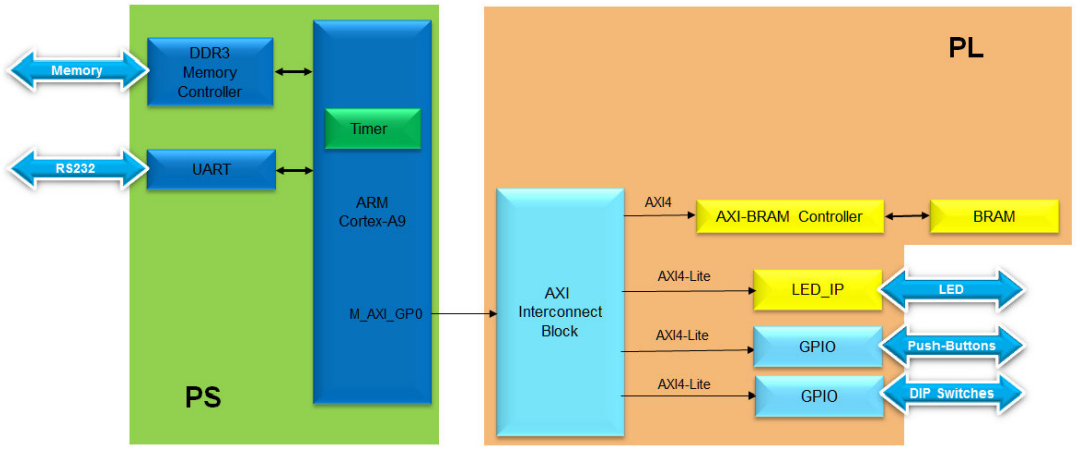

Figure 2: Completed Design

1. Create a new **RTL** project using Vivado. Name the project ***UART_app*** and select the device ***xc7z020clg400-1*** or the ***PYNQ*** board. Remember to select the ***"Create project subdirectory"*** option.

2. In previous labs, this is where we would create a new Verilog module. Instead, you will use the IP Integrator to create a new Block Design named ***"system"*** as shown in Figure 3.

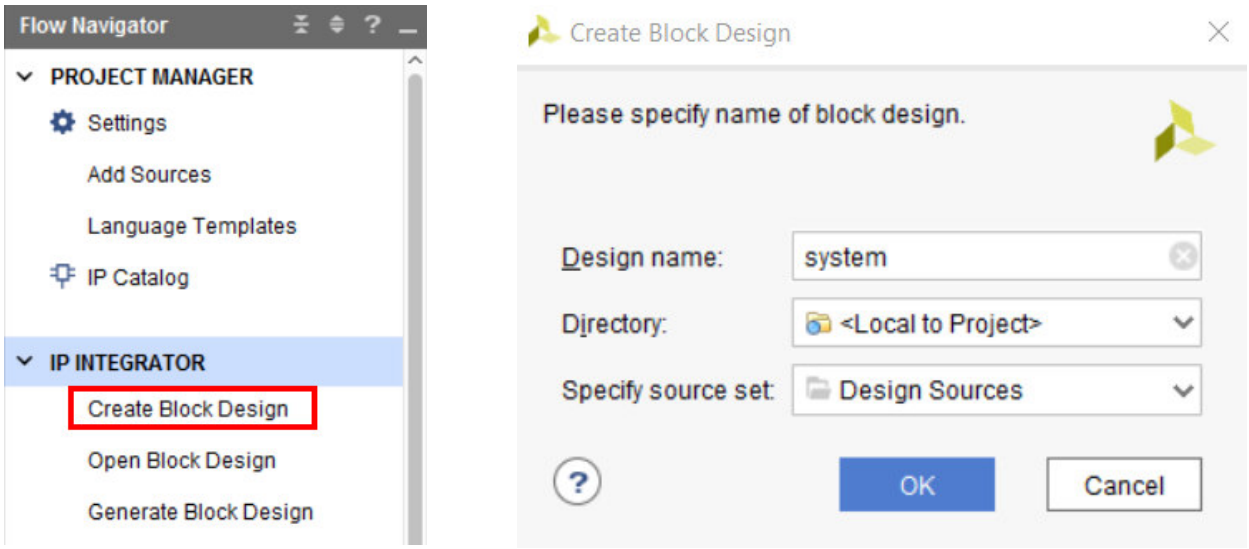

Figure 3: Creating a new block design

3. Next, click on the ***ADD IP*** icon by selecting one of the icons shown in Figure 4. Or you can press **CTRL+I** to add an IP into the empty diagram workspace.

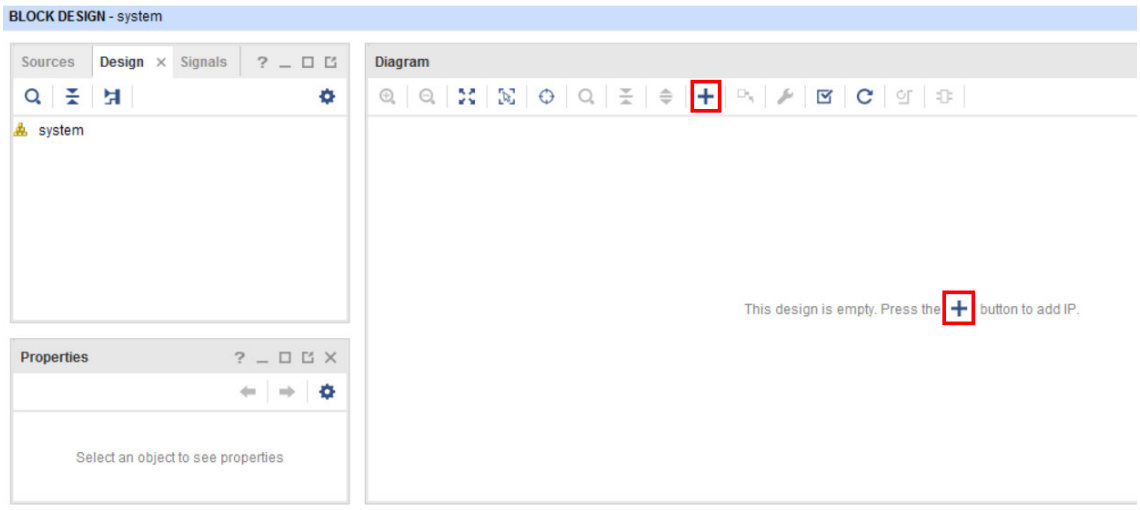

Figure 4: Adding an IP to the diagram workspace

4. Once the IP browser is open, enter ***"z"*** into the search box and select ***"ZYNQ7 Processing System"*** and press enter to add the ZYNQ hardcore processor IP to the block design. See Figure 5 below.

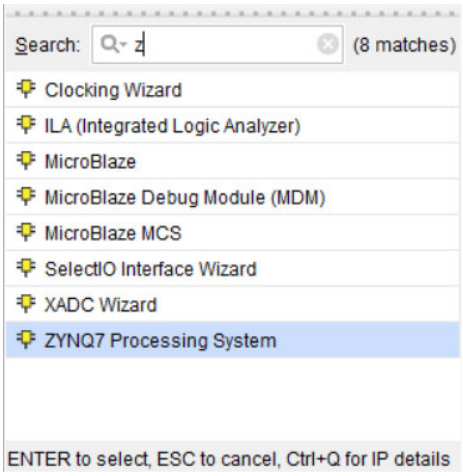

Figure 5: Adding the ZYNQ7 Processing System

5. Notice the green highlighted bar at the top of the block design window. This is where you will see messages for design assistance. Click on ***Run Block Automation*** and make sure the settings match those in Figure 6. The default settings should already have the desired settings. Click ***OK***.

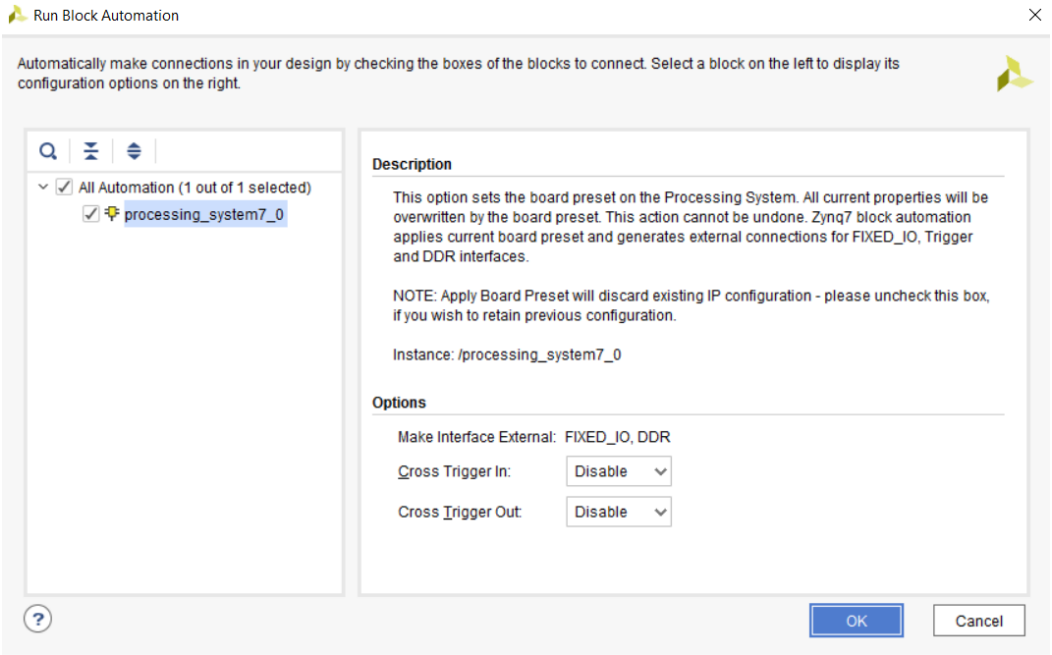

Figure 6: Run Block Automation settings

6. You should now see the newly added ZYNQ IP with the **DDR** and **FIXED_IO** ports added. See the figure below.

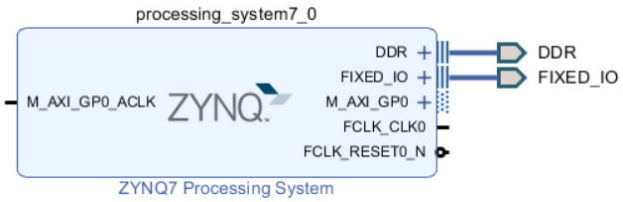

Figure 7: ZYNQ IP Core

7. Double click on the ZYNQ IP Core to open the ZYNQ block design. This is where you can configure your ZYNQ processor to enable or disable certain peripherals and features. Anything in green can be enabled or disabled. For this part, we only want to select the **UART 0** I/O peripheral. You can do this by clicking on the **MIO Configuration** tab in the **Page Navigator.** Make sure that only the **UART 0** is selected. Make sure **MIO 14** & **15** are set in the I/O column. Do **NOT** click OK yet.

8. Now, select the **PS-PL Configuration** tab in the **Page Navigator**. Deselect **M AXI GP0** as shown in the figure below.

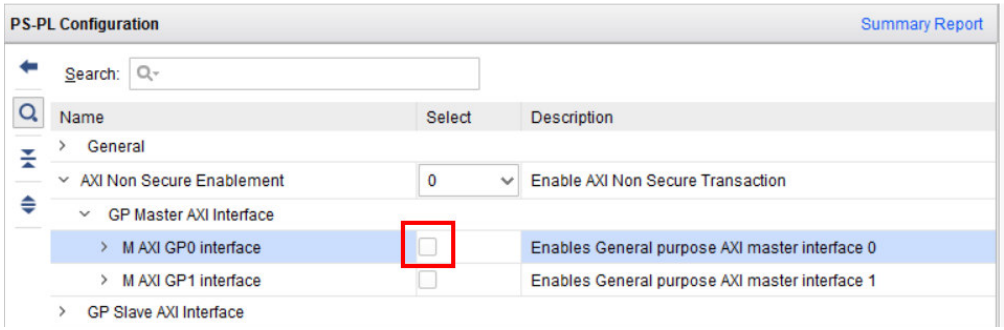

Figure 8: PS-PL Configuration

9. Expand **General → Enable Clock Presets** and deselect **FCLK_RESET0_N** as shown below.

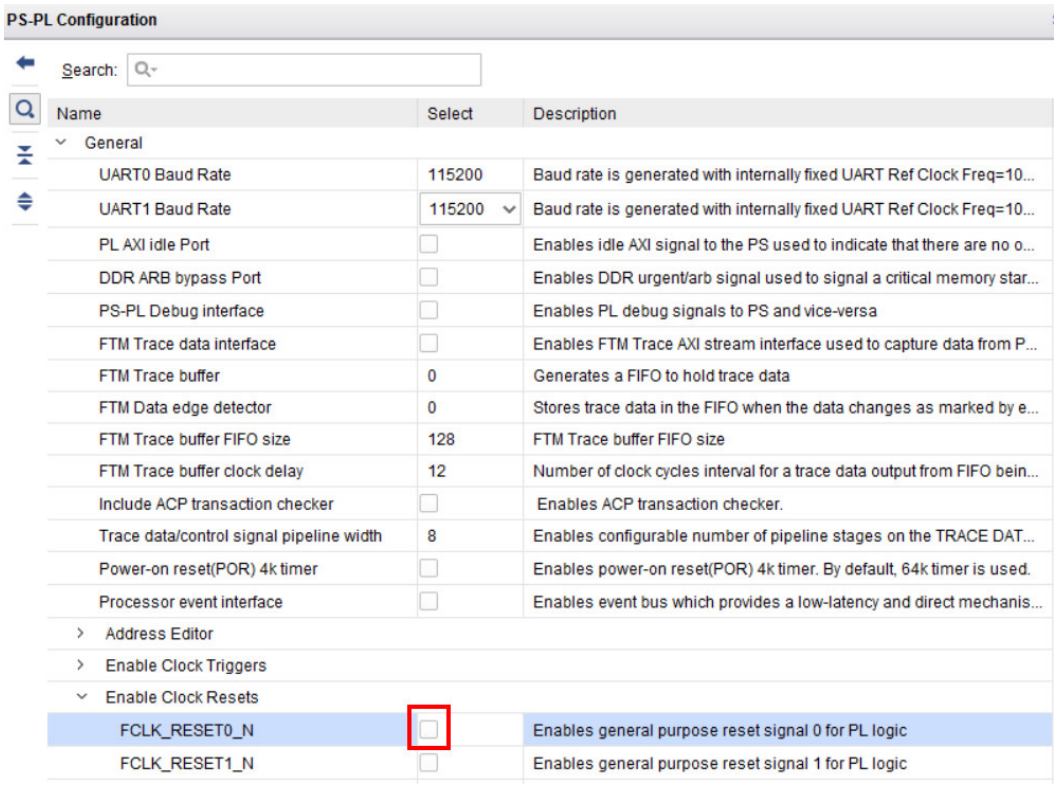

Figure 9: More PS-PL Configurations

10. Select the **Clock Configuration** tab in the **Page Navigator**. Expand the **PL Fabric Clocks** and deselect the **FCLK_CLK0**. Now we can click **OK** to finish the **IP** setup.

11. Click the **Regenerate Layout** button to update the ZYNQ block.

Figure 10: Regenerate Layout button

12. Next, click on **Validate Design** and make sure you see no errors in your design.

Figure 11: Validate Design button

13. Press **CTRL+S** to save your design.

14. It is now time to generate the **Top-Level** HDL and Export to the **SDK**. Right click on ***"system.bd"*** in the sources panel and select **Generate Output Products**, then click ***Generate***. This process will generate the implementation, simulation, and synthesis files for the design.

15. Right click ***"system.bd"*** again and select **Create HDL Wrapper**. Leave the default option of **Auto-Update** selected and click ***OK.*** The wrapper acts as a top-level Verilog module. Notice the ***"system_wrapper.v"*** file in the sources panel that was generated.

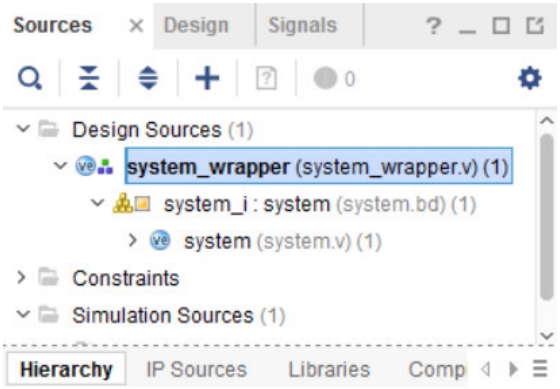

Figure 12: Sources panel showing the top-level HDL file

16. Double click on the ***"system_wrapper.v"*** file. Look through and understand the generated Verilog file.

17. Select **File → Export → Export Hardware** and click ***OK***.

18. Select **File → Launch SDK** leaving the default settings. Click ***OK***.

19. When the **SDK** opens, you should see the ***.hdf*** (Hardware Description File) in the preview pane. Double click the ***system.hdf*** file to open it if it's not already. This file shows basic information about the hardware configuration of the project like the address map.

20. Select **File → New → Application Project**

21. Name the project ***"HELLO_WORLD".*** Leave the default settings and click **Next**.

22. In the **New Project** window, select the ***"Hello World"*** template and click **Finish**.

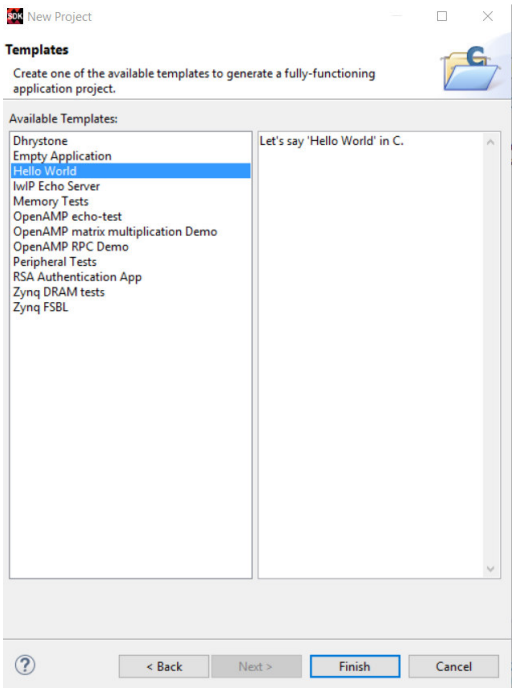

Figure 13: New project window

23. You should now see the newly created project and board support package in the **Project Explorer**. Expand the ***"HELLO_WORLD"*** and the ***"src"*** file trees. Double click on the ***"helloworld.c"*** file and examine the code.

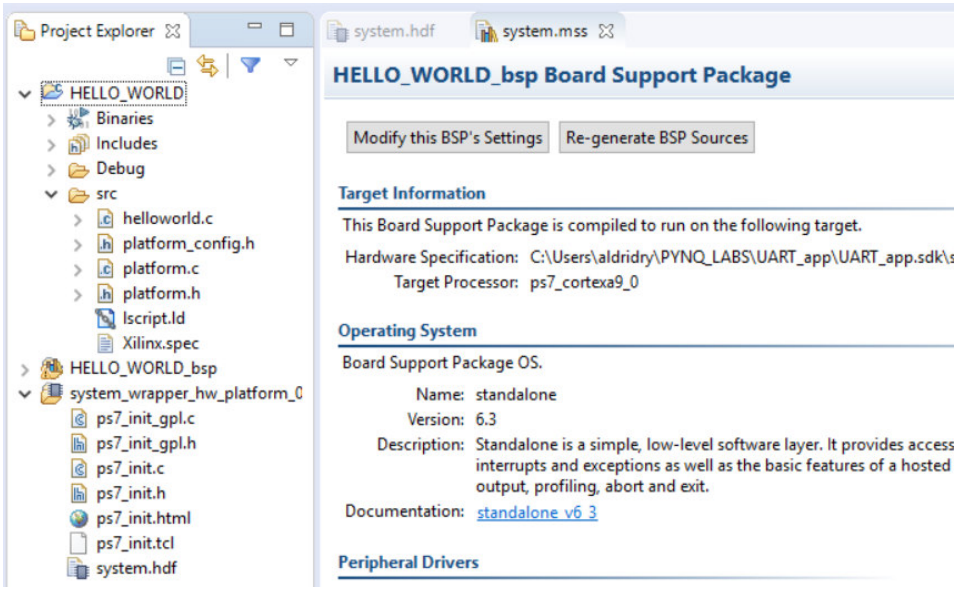

Figure 14: Project Explorer view

24. Connect the **PYNQ** board to your PC with the USB cable and power the board ON.

25. Click on the **Terminal** tab to bring the terminal window forward, then click on the **Connect to Serial Port** icon to bring up the serial port settings window. Select the COM port that the **PYNQ** board is connected to and make sure the BAUD rate is set to 115200. Leave the default settings for the rest of the fields and click **OK** to connect to your board. See Figure 15 below. Go to the Windows Device Manager to find which COM port the **PYNQ** board is connected to. Alternatively, use **Tera Term** instead of the **SDK** terminal.

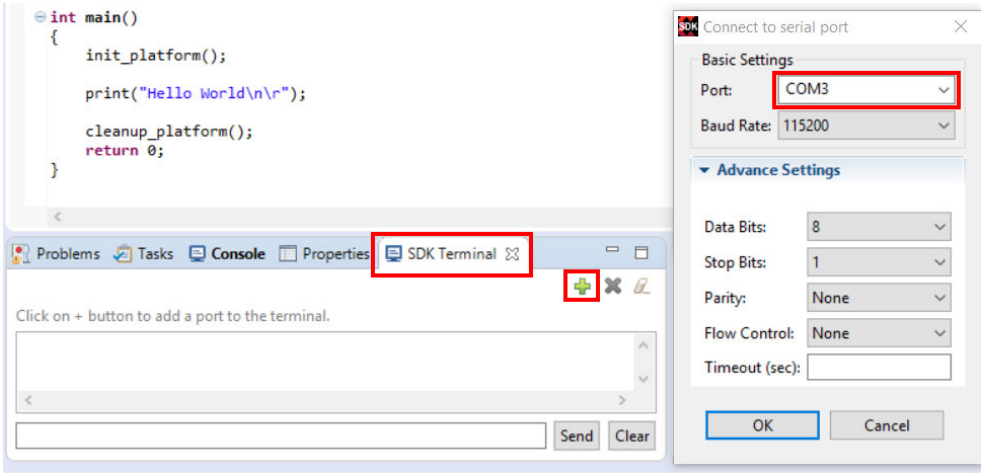

Figure 15: Connecting the PYNQ board to the serial port

26. Select the ***"HELLO_WORLD"*** project in the **Project Explorer**, right click and select **Debug As → Launch on Hardware (System Debugger)**. This will download the application to the board and execute the application. Press F8 or click the **Resume** icon in the **Debugger Toolbar** to run your application. You should see ***"Hello World"*** displayed on the terminal window.

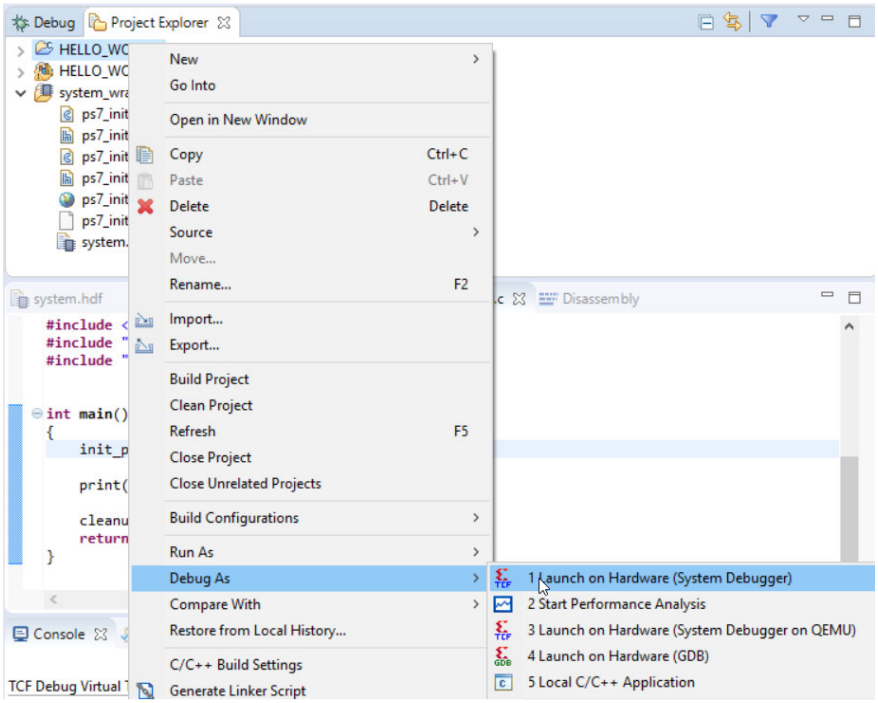

Figure 16: Downloading to the board

27. Now it is your turn. Modify the ***"helloworld.c"*** file to display a logo of your choice.

Demonstrate your logo to your instructor

Part II – Adding IP Cores for GPIO

In this part, you will expand your knowledge of software processor systems by adding **IP Cores** for the **GPIO** (General-Purpose Input/Output) interface. You will create a button/switch status indicator that prints to the **UART** interface. You will also learn how to connect to external ports, generate a bitstream, and finally verify the design in hardware.

1. Create a new **RTL** project using Vivado. Name the project ***GPIO_app*** and select the device ***xc7z020clg400-1*** or the ***PYNQ*** board. Remember to select the ***"Create project subdirectory"*** option.

2. Complete steps 2-6 from Part I to add the ZYNQ7 Processing System to your design.

3. Double click on the ZYNQ IP Core to open the ZYNQ block design.

4. Select the **MIO Configuration** tab in the **Page Navigator** and expand **I/O Peripherals**. Deselect everything except **UART 0.** Make sure **UART 0** is enabled and **MIO 14 & 15** are selected from the dropdown I/O field. Expand **GPIO** and deselect **GPIO MIO**.

5. Select the **PS-PL Configuration** tab and expand **AXI Non Secure Enablement → GP Master AXI Interface** and make sure the **M AXI GP0 Interface** is enabled.

6. Expand **General → Enable Clock Resets** and make sure the **FCLK_RESET0_N** option is selected.

7. Select the **Clock Configuration** tab and expand **PL Fabric Clocks**. Make sure the **FCLK_CLK0** option is selected with a 100 MHz frequency. Click ***OK***. The ZYNQ Processing System should now show the AXI_GP0 interface along with the required clocks as shown in Figure 17.

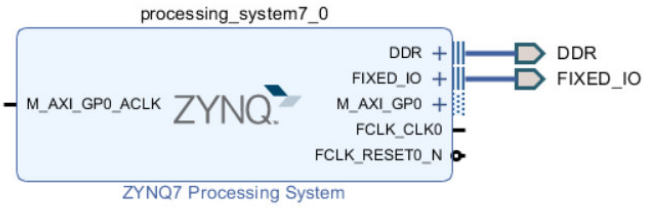

Figure 17: ZYNQ Processing System

8. Click on the **ADD IP** icon and search for **AXI GPIO** in the catalog. Select **AXI GPIO** and press enter.

9. Click on the **AXI GPIO** block and change the name to ***"switches"***.

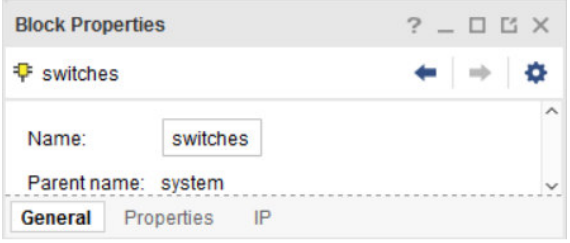

Figure 18: Renaming the GPIO IP

10. Now, double click on the **AXI GPIO** block to open the **Customization Window**. Set the **Board Interface** to ***"sws 2bits"*** for **GPIO**. Click ***OK.***

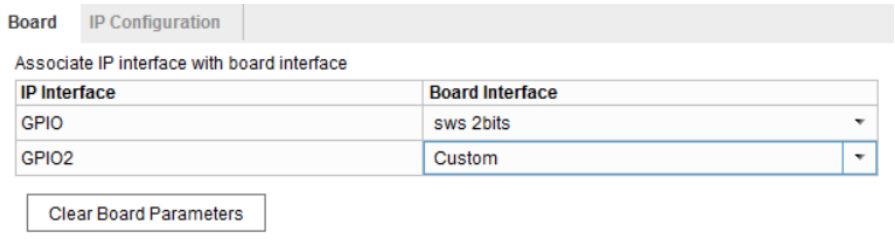

Figure 19: Customizing the GPIO IP

11. Vivado offers automated design assistance for making **IP** block connections. To take advantage of this assistance, click ***"Run Connection Automation"*** in the green bar at the top of the block diagram window. Select ***"switches"*** and ***"S_AXI"*** to connect the switches to the **AXI** interconnect and click ***OK.***

12. Click **Regenerate Layout**. You should see a block diagram as shown in Figure 20.

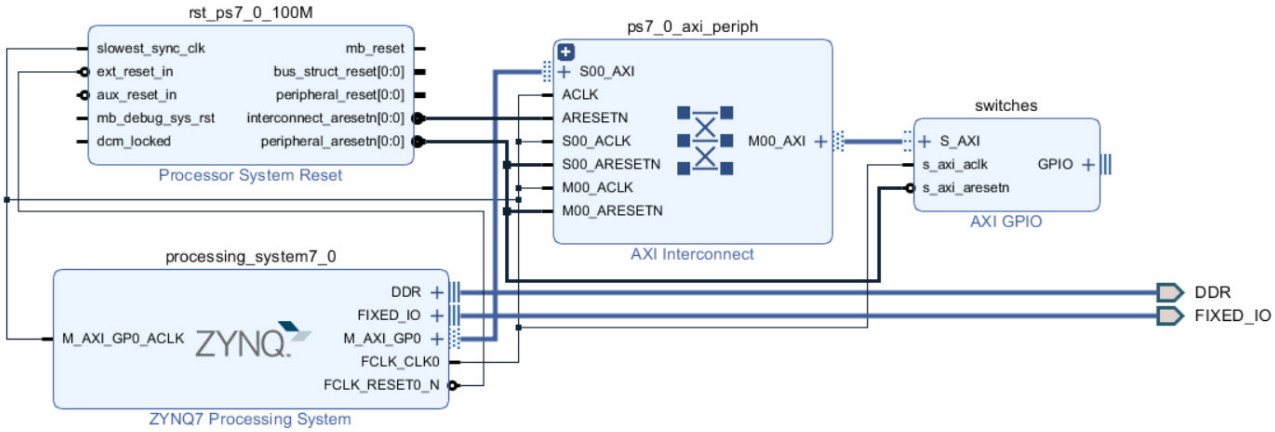

Figure 20: Updated block diagram

13. Now, another IP needs to be added for the switches. Repeat steps 8 & 9 and name the block ***"buttons"***.

14. Double click the ***"buttons"*** block and change the **GPIO Board Interface** to ***"btns 4bits"***. Click ***OK.***

15. We could run the **Run Connection Automation** wizard, instead, we will connect the block manually.

16. Double click the **AXI Interconnect** and change the **Number of Master Interfaces** to 2 and click ***OK***.

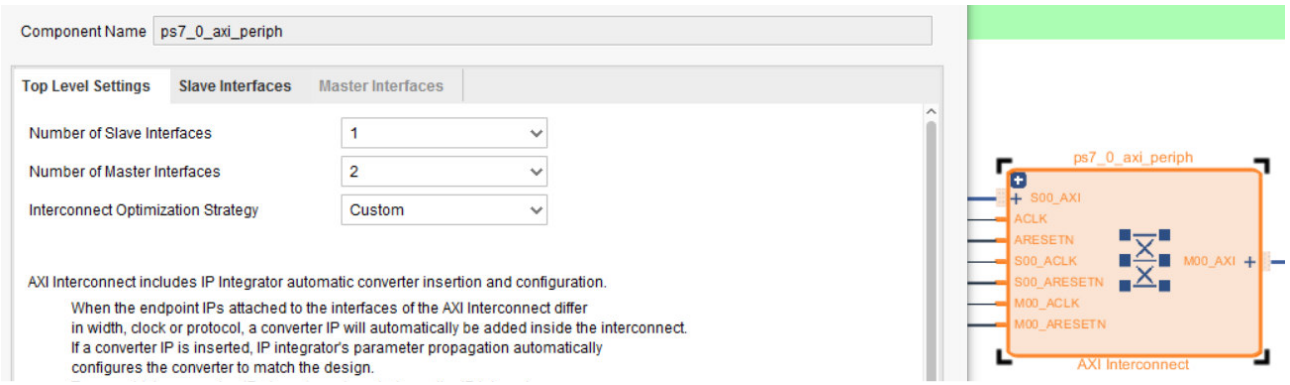

Figure 21: Changing the number of AXI master interfaces

17. Click, hold, and drag on the **S_AXI** port on the buttons block (you will see a pencil appear). Drag this port towards the **M01_AXI** port on the **AXI Interconnect** and you should see a green checkmark appear where the connection needs to be made. Make the connection.

18. Now, make the following connections the same way you did in step 17:

    a. **buttons (*s_axi_aclk*) → ZYNQ (*FCLK_CLK0*)**
    b. **buttons (*s_axi_aresetn*) → Processor System Reset (*peripheral_aresetn*)**
    c. **AXI Interconnect (*M01_ACLK*) → ZYNQ (*FCLK_CLK0*)**
    d. **AXI Interconnect (*M01_ARESETN*) → Processor System Reset (*peripheral_aresetn*)**

    The block diagram should now look like Figure 22 after you click **Regenerate Layout.**

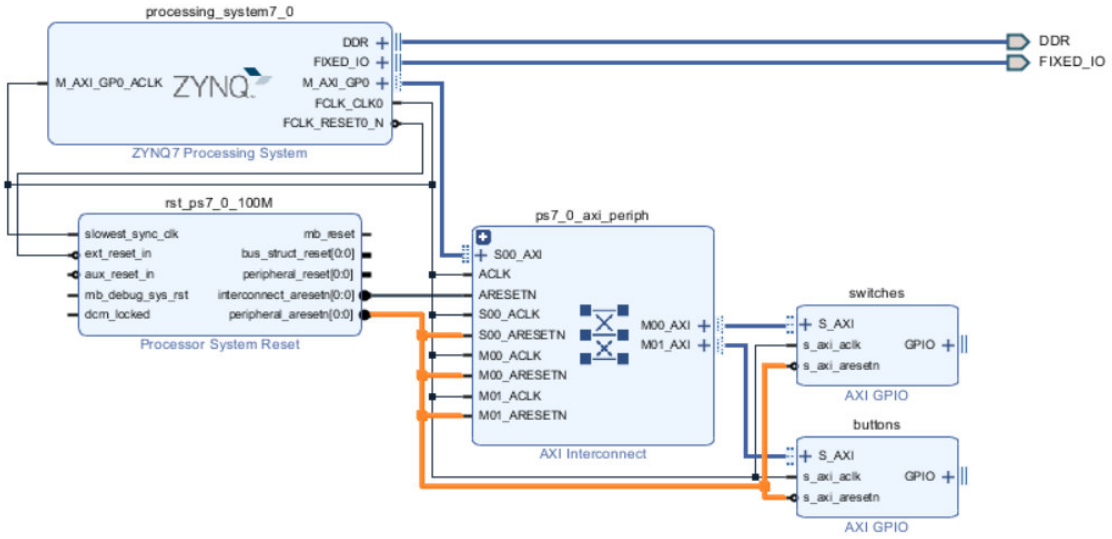

Figure 22: Updated block diagram

19. Click on the **Address Editor** tab and expand **processing_system7_0 → Data → Unmapped Slaves**. Here we see that the buttons have not been assigned an address. This is because we made the connections manually. Right click on **buttons** and select **Assign Address.** The buttons have now been assigned with an address.

20. We now need to connect the buttons and switches to the corresponding pins on the **FPGA**. This can be done manually, or by using the **Designer Assistance**. Click on **Run Connection Automation** in the green bar at the top of the diagram view.

21. Select **All Automation** and select **buttons → GPIO** and make sure the part interface is set to ***"btns_4bits"***. Select **switches → GPIO** and make sure the part interface is set to ***"sws_2bits"***. Click ***OK.*** Vivado is "board aware" and will automatically apply the constraints to the port.

22. Click on the newly created external ports and change the name to ***"switches"*** and ***"buttons"*** in the **External Interface Properties** window.

23. Validate your design and make sure there are no errors.

24. The final block design should look like Figure 23 below.

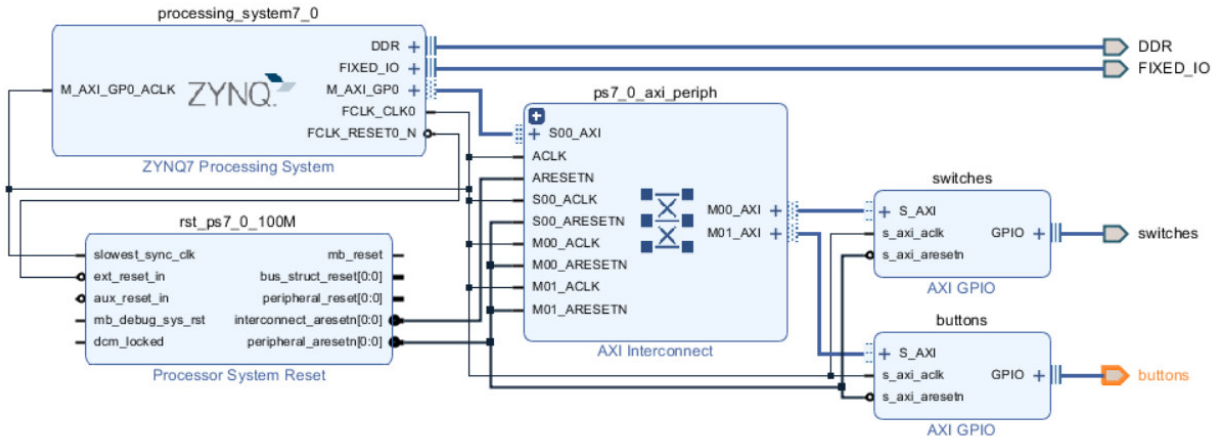

Figure 23: Finished block design

25. Before we can move on, we need a **Top-Level** system wrapper. Right click on ***"system.bd"*** and select **Create HDL Wrapper**. Select **Let Vivado manage and auto-update** and click ***OK***.

26. Click on **Run Synthesis** in the **Flow Navigator**. Make sure your design synthesizes without errors.

27. When the synthesis completes, select **Open Synthesized Design** and click ***OK***.

28. Select **Layout → I/O Planning** in the top menu bar. This is where you can assign pin locations on the **FPGA**. Expand **GPIO_41639 → switches_tri_i**. You will notice that the pins have been assigned automatically. This is where you would assign pins if needed.

Tcl Console | Messages | Log | Reports | Design Runs | Package Pins | I/O Ports

Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco
switches_tri_i (2)	IN					☑	35	LVCMOS33*	3.300
switches_tri_i[1]	IN	sws_2bits_tr...			M19	☑	35	LVCMOS33*	3.300
switches_tri_i[0]	IN	sws_2bits_tr...			M20	☑	35	LVCMOS33*	3.300
Scalar ports (0)									
GPIO_43611 (4)	IN					☑	35	LVCMOS33*	3.300
buttons_tri_i (4)	IN					☑	35	LVCMOS33*	3.300
buttons_tri_i[3]	IN	btns_4bits_tr...			L19	☑	35	LVCMOS33*	3.300
buttons_tri_i[2]	IN	btns_4bits_tr...			L20	☑	35	LVCMOS33*	3.300
buttons_tri_i[1]	IN	btns_4bits_tr...			D20	☑	35	LVCMOS33*	3.300
buttons_tri_i[0]	IN	btns_4bits_tr...			D19	☑	35	LVCMOS33*	3.300

Figure 24: Pin assignment

29. Click on **Generate Bitstream** in the **Flow Navigator**. Click **Yes** if prompted to launch implementation. Alternatively, you could have clicked **Run Implementation** and then **Generate Bitstream**. If there is no up-to-date implementation when you generate the bitstream, Vivado will ask you to run the implementation first. Make sure the implementation and bitstream generation complete successfully and click **Cancel** when it completes.

30. Click **File → Export → Export Hardware**. In Part I, we did not have any hardware in programmable logic. This time we do and need to **Include bitsream**. Then click ***OK***.

31. Click **File → Launch SDK** and click ***OK***.

32. Click **File → New → Board Support Package** and click ***Finish***.

33. Click ***OK*** on the **Board Support Package Settings** window.

34. Click **File → New → Application Project**.

35. Name the project ***GPIO_status*** and select **Use existing** board support package. Click ***Next***.

36. Select **Empty Application** and click ***Finish***. This will create a new empty project for you to design your GPIO status project.

37. In the **Project Explorer**, expand **GPIO_status → src.** Right click on **src** and click **New → Source File**.

38. Name the new source file **GPIO_status.c** and click ***Finish***.

39. Type the code below into your **GPIO_status.c** file. This application will initialize the switches and buttons, then continuously check the status and print to the terminal.

```
#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 25: C Program for checking switch and button status

40. Now, connect the **PYNQ** board to the PC with the USB cable and power the board ON.

41. Open the **SDK** terminal and create a connection to the board, or connect to the board using **Tera Term**.

42. Before we can load the software application on the board, we need to program the hardware with the **.bit** file generated from the block design. In the top menu, select **Xilinx Tools → Program FPGA**. Use Figure 26 to make sure you have the proper settings and then click **Program** to program the FPGA. The

green **DONE** LED will turn on when the bitstream upload is complete.

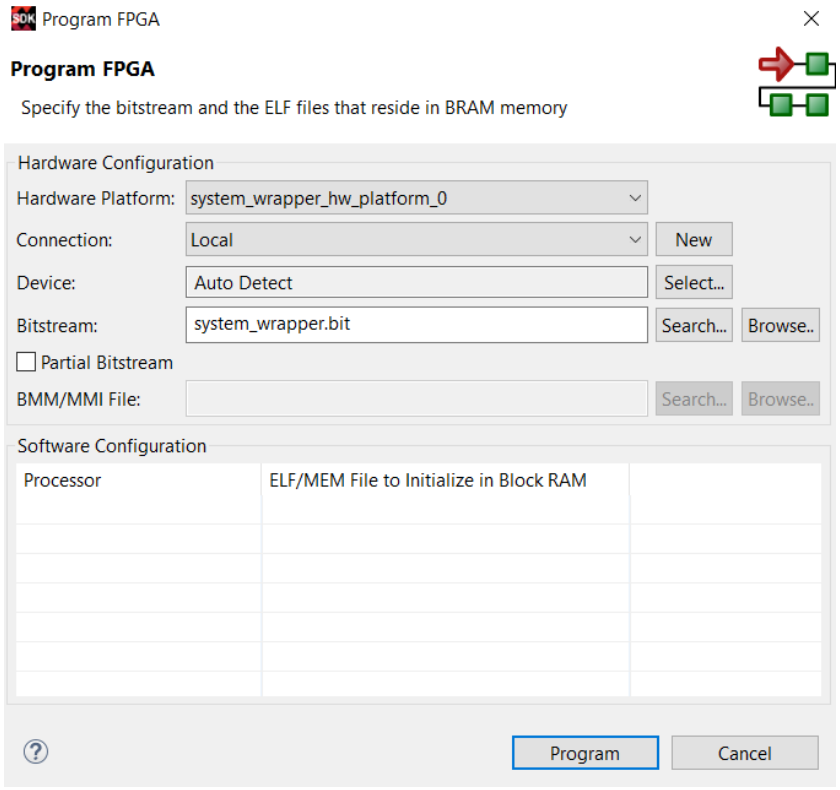

Figure 26: Programming the FPGA

43. Finally, select the **GPIO_status** project in the **Navigator**. Right click the **GPIO_status** project and select **Debug As → Launch On Hardware (System Debugger)**.

44. When the application finishes loading onto the board, the debugger will launch and wait for you to run the program. Press **F8** or click **Resume**. Bring your terminal window up and you should see the program running on your terminal window. Verify the functionality by toggling the switches and pressing the buttons on the **PYNQ** board. You should see the status update on the terminal window.

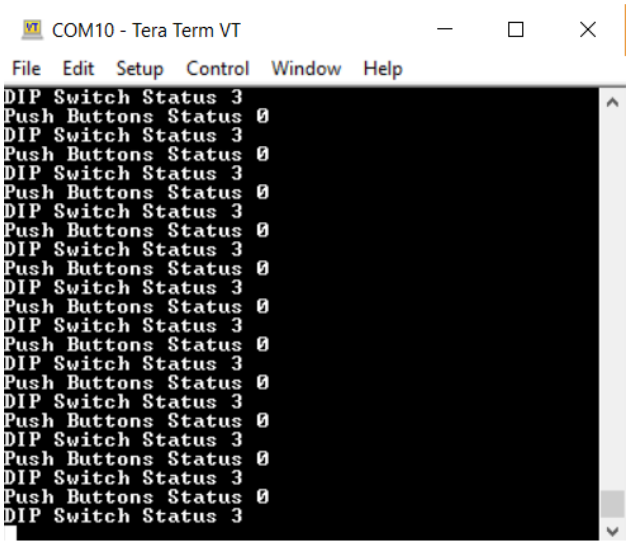

Figure 27: GPIO Application output

Part III – Let's Make a Deal Gameshow Application

In this part, you will create a new application that runs on the same hardware design that you created in Part II. You do not need to create a new block design and IP cores. Start by returning to the **GPIO_app** Vivado project and launch the **SDK**. You will be creating a game called **Let's Make a Deal.** Modify the **GPIO_status** application project in the **SDK** to pick a random door and then ask the user to guess which door is the correct door that the computer selected. The terminal should then print ***"You Win!"*** if the correct door was guessed, or ***"You Suck!"*** if the wrong door was guessed.

Application Behavior

1. On downloading the BIT file:
    a. The terminal window should welcome the user to the game and prompt for a number between 1 and 5 to be entered.
    b. This number is arbitrary and has no impact on the game. The time between when the program launches and when the user enters the number is what is used to seed the random number generator.
    c. Include ***"xtime_l.h"*** in your program and use the functions in this library to get the reaction time from the user. One good function to use is shown below.

```
XTime_GetTime(&tStart);
```

2. The only user input to the terminal window is to seed the random number generator. The door is selected by using one of the 4 buttons on the **PYNQ** board.
3. The reaction time (seed) from the user needs to be passed to a random number generator. The trick is to convert this random number into integers between 0 and 3. ***Hint: Modulus***
4. The converted random number should be compared to the button that the user pressed to determine the game result.
5. A sample run of the program is shown below:

- User selects 4
- Based on the amount of time user takes to enter his/her selection, computer chose 1
- {Btn3, Btn2, Btn1, Btn0} is the configuration for buttons that is equivalent to the configuration of Doors (Door3, Door2, Door1, Door0}.
    - Binary representation of 1 is {0,0,0,1}indicating that Door0 is selected by the computer
- After computer selects the door, the program waits for user to press the button.
- If user presses any button on the PYNQ board besides Btn0, then it displays the message "***You suck !!!***"

6. Another sample run of the program is shown below:

   - User selects 5
   - Based on the amount of time user takes to enter his/her selection, computer chose 4
   - Binary representation of 4 is {0,1,0,0}indicating that Door2 is selected by the computer
   - After computer selects the door, the program waits for user to press the button.
   - If user presses Btn2 on the PYNQ board, then it displays the message "***You can read my mind !!!***"

<u>NOTE</u>: You should print the message "*Computer chose __*" only for debugging purposes in order to implement true essence of the game.

Laboratory Deliverables

You are required to turn in a hard copy of the report. Report should have the following items:

- Cover page with one page description of your design
- Printout of your code with comments
  - Logo
  - Let's Make a Deal

You also have to demonstrate your design and turn in a compressed version of the project