
Custom IP Design using PYNQ

Due Date: By beginning of next lab section

Objectives

- Use the Vivado IP Packager to create a custom peripheral
- Create an AXI4Light interface peripheral
- Add a custom peripheral to your design
- Utilize block memory in your design

Part I – IP Packaging

In this part, you will create a custom IP LED driver utilizing the four on-board LEDs provided on the PYNQ platform. You will create and add an AXI peripheral and connect it to the LEDs, then use the IP Packager to generate the custom IP. After the custom IP is generated, you will connect the peripheral to the ZYNQ processing system and add the pin constraints for the on-board LEDs. Lastly, you will need to add a BRAM controller and BRAM before generating the bitstream. Refer to Figures 1 & 2 for the overall flow and design of this lab.

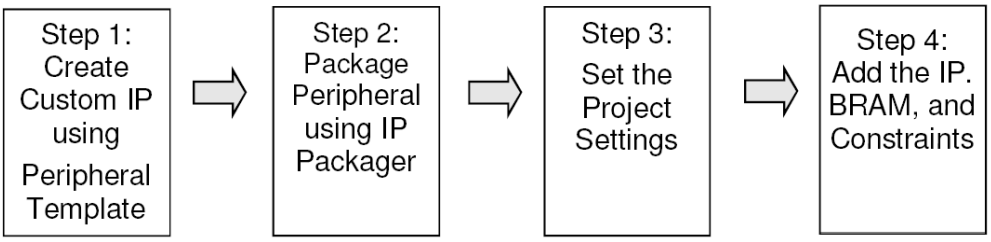

Figure 1: General flow of lab

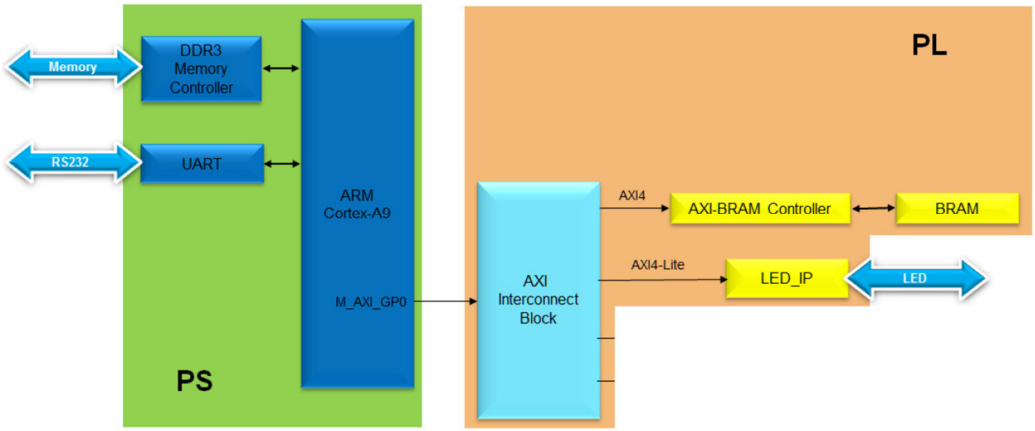

Figure 2: Completed design

1. Open Vivado and click **Manage IP** in the **Tasks** menu. Select **New IP Location** and click **Next**.

2. Select the **PYNQ-Z1** board in the **Manage IP Settings** window. See Figure 3 below. After the **PYNQ**

board is selected, click **OK** and then click **Finish**.

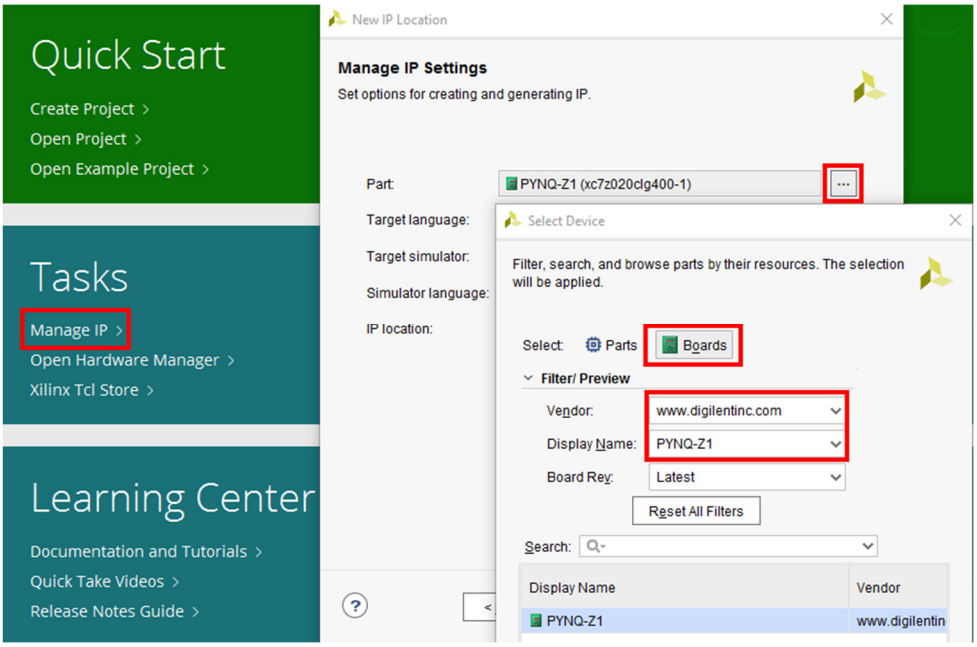

Figure 3: Creating a new IP

3. Open the **Tools** menu from the top menu bar and click **Create and Package New IP**.

4. Click **Next** to continue. Select **Create a new AXI4 peripheral** and click **Next**.

5. Rename your IP ***"led_ip"*** and click **Next**. Under **IP Location**, make sure you select an appropriate location and the destination folder should be ***ip_repo***.

6. Change the name to ***"S_AXI"*** and leave the rest of the settings as default. See the figure below to confirm the correct **AXI4** settings and then click **Next**.

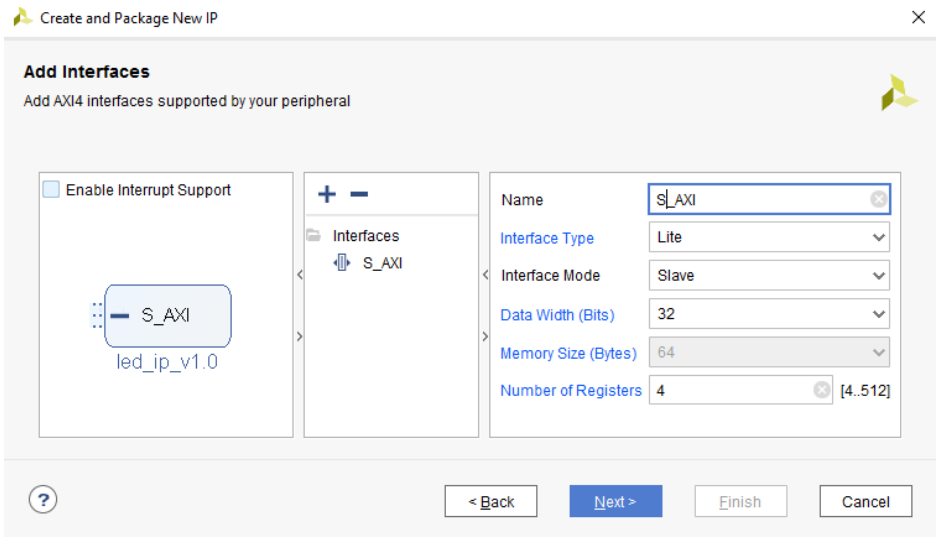

Figure 4: AXI interface settings

7. Click on **Edit IP** under ***Next Steps*** and then click **Finish** to create your peripheral. You may need to scroll down in this window to see the **Edit IP** option.

8. You will now notice that a new Vivado project has opened.

9. In the sources panel, double-click on the ***led_ip_v1_0.v*** Verilog file. This will open the Verilog HDL top module code for the newly created **IP**. This code contains the **AXI** interfacing logic template that you will use to create the custom **IP**.

10. In line 7 of the Verilog code, enter the following code:

```
7        parameter integer LED_WIDTH     = 4,
```

11. In line 18 of the Verilog code, enter the following code:

```
17        // Users to add ports here
18        output wire [LED_WIDTH-1:0]      LED,
19        // User ports ends
```

12. Your Verilog code should be similar to the code in Figure 5.

```
4     module led_ip_v1_0 #
5     (
6         // Users to add parameters here
7         parameter integer LED_WIDTH     = 4,
8         // User parameters ends
9         // Do not modify the parameters beyond this line
10
11
12        // Parameters of Axi Slave Bus Interface S_AXI
13        parameter integer C_S_AXI_DATA_WIDTH    = 32,
14        parameter integer C_S_AXI_ADDR_WIDTH    = 4
15    )
16    (
17        // Users to add ports here
18        output wire [LED_WIDTH-1:0]     LED,
19        // User ports ends
```

Figure 5: Code snippet of user parameter and port definitions

13. Scroll down to line 48 and enter the following code:

```
48        .LED_WIDTH(LED_WIDTH),
```

14. Enter the following code on line 52:

```
52        .LED(LED),
```

15. Your code should resemble that of Figure 6.

```
46  // Instantiation of Axi Bus Interface S_AXI
47      led_ip_v1_0_S_AXI # (
48          .LED_WIDTH(LED_WIDTH),
49          .C_S_AXI_DATA_WIDTH(C_S_AXI_DATA_WIDTH),
50          .C_S_AXI_ADDR_WIDTH(C_S_AXI_ADDR_WIDTH)
51      ) led_ip_v1_0_S_AXI_inst (
52          .LED(LED),
53          .S_AXI_ACLK(s_axi_aclk),
```

Figure 6: Instantiation of the LEDs

16. The code you were editing was the top-level module. Now, you will declare user ports in a lower-level module. In the sources pane, expand ***led_ip_v1_0*** and open ***led_ip_v1_0_S_AXI.v***. Edit lines 7 and 18 to resemble the code in Figure 7 below.

```
4   module led_ip_v1_0_S_AXI #
5   (
6       // Users to add parameters here
7       parameter integer LED_WIDTH             = 4,
8       // User parameters ends
9       // Do not modify the parameters beyond this line
10
11      // Width of S_AXI data bus
12      parameter integer C_S_AXI_DATA_WIDTH    = 32,
13      // Width of S_AXI address bus
14      parameter integer C_S_AXI_ADDR_WIDTH    = 4
15  )
16  (
17      // Users to add ports here
18      output wire [LED_WIDTH-1:0]     LED,
19      // User ports ends
```

Figure 7: Declaring user ports in the lower-level module

17. Scroll down to line ~400 and enter the code below to instantiate the user logic for the LED IP.

```
400     // Add user logic here
401     user_logic # (
402         .LED_WIDTH(LED_WIDTH)
403         )
404        U1 (
405             .S_AXI_ACLK(S_AXI_ACLK),
406             .slv_reg_wren(slv_reg_wren),
407             .axi_awaddr(axi_awaddr[C_S_AXI_ADDR_WIDTH-1:ADDR_LSB]),
408             .S_AXI_WDATA(S_AXI_WDATA),
409             .S_AXI_ARESETN(S_AXI_ARESETN),
410             .LED(LED)
411            );
```

Figure 8: Instantiating lower-level user module

18. Now that the lower-level module has been instantiated, the module itself needs to be created. In the flow navigator, click **Add Sources** and select **Add or create design sources** and click **Next**. Click **Create File** and name the file ***user_logic*** using the file directory where you have your **IP** saved (see Figure 9 for the path).

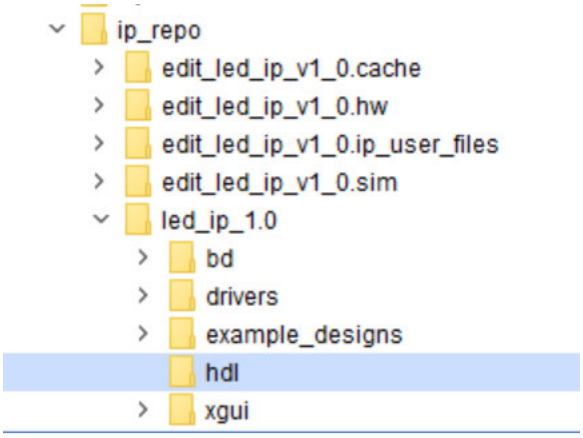

Figure 9: Path to save the *user_logic* source file

19. Now that the ***user_logic*** module has been created, copy the code shown in Figure 10 below into this module. **Now <u>Save</u> all the files.**

```
6  module user_logic # (
7      parameter LED_WIDTH = 8
8      )
9      (
10     input S_AXI_ACLK,
11     input slv_reg_wren,
12     input [2:0] axi_awaddr,
13     input [31:0] S_AXI_WDATA,
14     input S_AXI_ARESETN,
15     output reg [LED_WIDTH-1:0] LED
16     );
17
18   always @( posedge S_AXI_ACLK )
19   begin
20     if ( S_AXI_ARESETN == 1'b0 )
21         LED <= 4'b0;
22     else
23       if (slv_reg_wren && (axi_awaddr == 3'h0))
24           LED <= S_AXI_WDATA[LED_WIDTH-1:0];
25       end
26 endmodule
```

Figure 10: User logic module

20. After completing the module, click **Run Synthesis** and save the project if prompted. When the synthesis successfully completes, click **Cancel** when the dialog box appears.

21. Now it is time to package the **IP**. In the **Flow Navigator** pane, click **Package IP**.

22. Select the **Package IP** tab in the **Project Manager** to view the **IP** packager. In this window, click on the + symbol on the bottom of the window to add the **IP** to a category. This will make the **IP** easier to find when trying to add it to a project later on. Make sure the **AXI Peripheral** and the **Basic Elements** boxes are check and click **OK**. Refer to Figure 11 below.

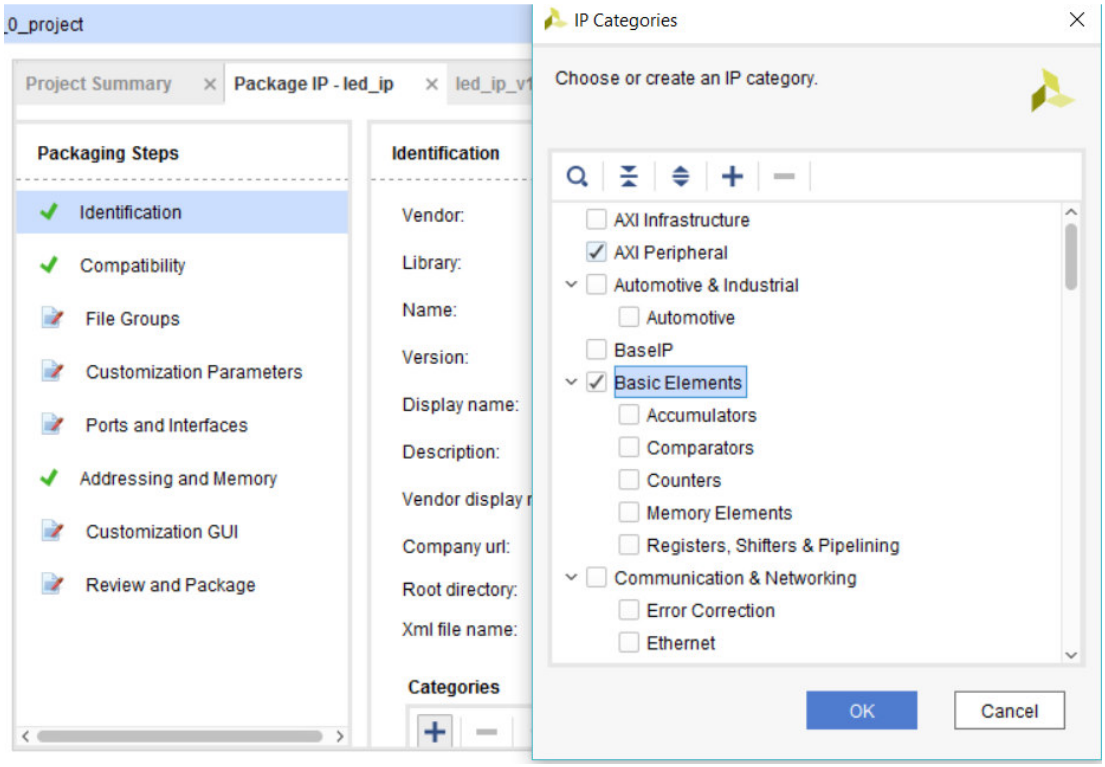

Figure 11: Setting the IP category

23. Next, click on the **Compatibility** tab. Click the + at the top of the window and click **Add Family Explicitly** as seen below in Figure 12. A window of Xilinx chip families will pop up. Select the **ZYNQ-7000** family as this is the chip on the **PYNQ** board and click **OK**.

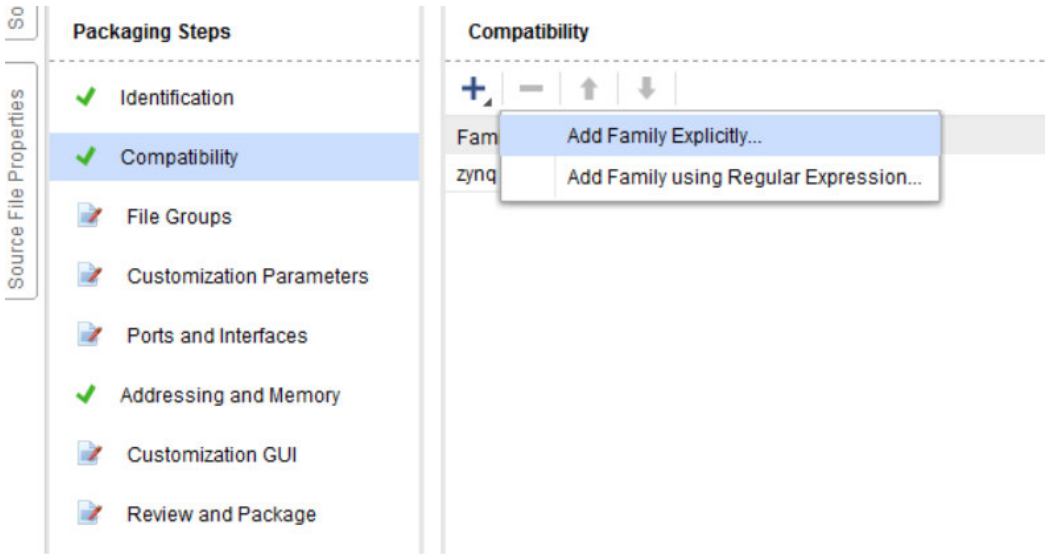

Figure 12: Compatibility IP settings

24. Click on the **File Groups** tab and click **Merge changes from File Groups Wizard** (Figure 13). Since you just added the new ***user_logic.v*** file to the **IP**, you will need to update the **IP Packager** with the changes. This step will do just that. If you expand the **Verilog Synthesis** tree, you will notice that the ***user_logic.v*** file was added.

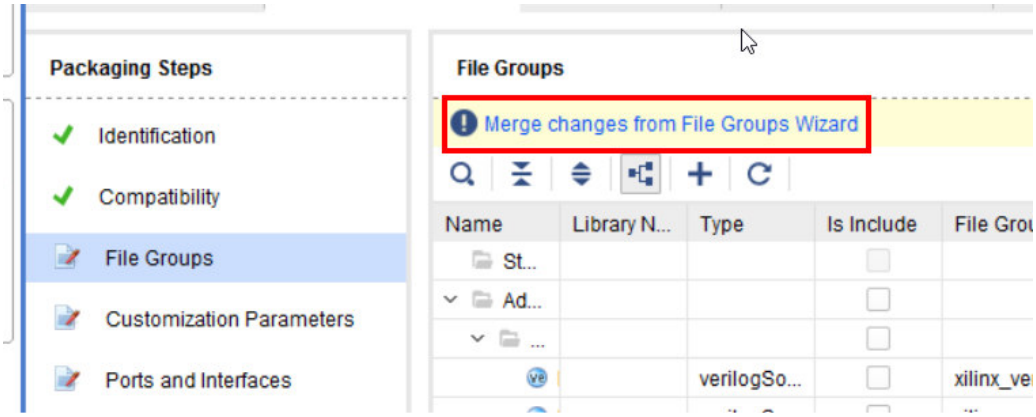

Figure 13: Merging file groups

25. Click on the **Customization Parameters** tab and click **Merge changes from Customization Parameters Wizard**. Next, expand the **Hidden Parameters** tree and right-click on **LED_WIDTH** and select **Import IP Parameters** and click **OK**.

26. Click on the **Ports and Interfaces** tab and you should notice that the **Ports and Interfaces** now shows the user created LED port.

27. Click on the **Customization GUI** tab and you will see the block diagram element of your custom **IP**.

28. Click on the **Review and Package** tab and note where the new **IP** will be created. Click **Re-Package IP** and click **Yes** to close the **IP Manager**.

29. In the original Vivado window, click **File → Close Project**.

Part II – Hardware Implementation of a Custom IP

In this part, you will add your custom **IP** to the previously completed lab 4. Lab 4 already has the buttons and switches **IP** added, so the new LED **IP** will be added to the design to provide a peripheral output for the button and switch **IPs**.

1. Open your lab 4 project called ***GPIO_app***, click **File → Save As** and rename the project ***led_ip***. Make sure that the **Create Project Subdirectory** option is checked.

2. In the **Flow Navigator** pane, select **Settings → IP → Repository**. Click on the + button and select the folder that you have the ***led_ip*** stored and click **Select**. You should see a dialog box appear that shows the new **IP** added to the repository. Click **OK** twice. See figure 13 below as a reference.

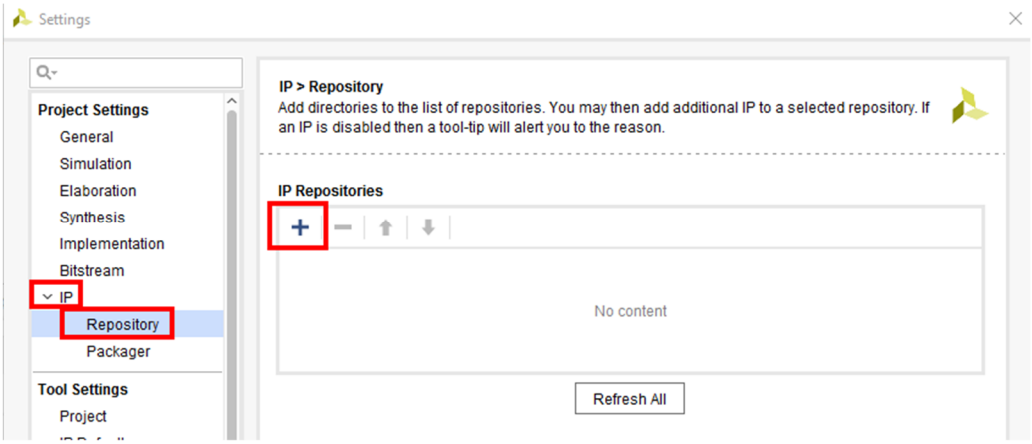

Figure 13: Adding an IP repository to the project

3. Now, add the ***led_ip*** to your design by clicking **Open Block Design**. Next, click the + to search for the LED **IP** created in the last lab. Type ***led*** in the search box and the newly created **IP** will appear. Double-click on the **IP** to add it to the block diagram.

4. Click **Run Connection Automation** in the green bar at the top of the window to connect the **IP** to the block diagram. Click **OK**.

5. Select the LED port on the ***led_ip*** by clicking on its pin. Right-click and select **Make External**.

6. Select the ***led_ip*** block and rename the **IP** to ***led_ip*** in the block properties window.

7. Next, click the **Regenerate Layout** button in the **Diagram** menu bar to arrange the block diagram to look like figure 14.

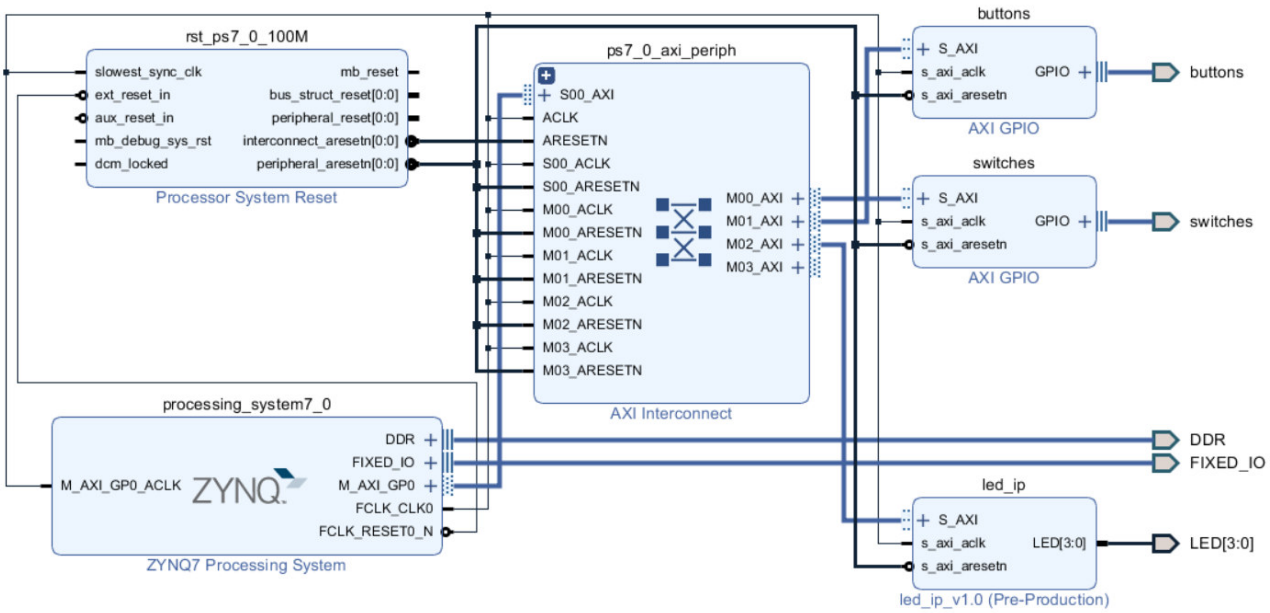

Figure 14: Block Diagram with the new LED IP

8. Select the **Address Editor** tab in the block design and verify that an address has been assigned to ***led_ip***.

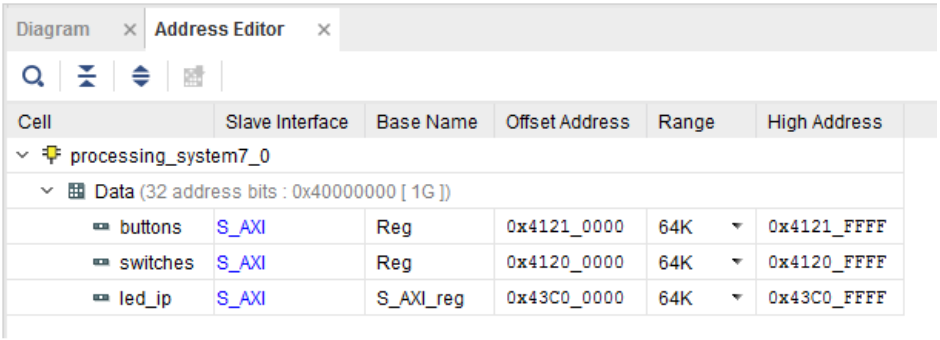

Figure 15: Address Editor

9. Now block memory, or BRAM, needs to be added to the design. In the block design, add another **IP** called AXI BRAM Controller and run the **Connection Automation** tool. Select **axi_bram_ctrl_0 → S_AXI** and click **OK**. Then click **Regenerate Layout**.

10. Double-click the **BRAM Controller** to customize it and change the number of **BRAM** interfaces to **1** and click **OK**.

11. Now, click on **Run Connection Automation** again to add a **Block Memory Generator**. Select **axi_bram_ctrl_0 → BRAM_PORTA** and then click **OK**. This could also be added manually.

12. Press **F6,** or **Tools → Validate Design** to validate the design to make sure there are no errors. Your design should look like the one below.

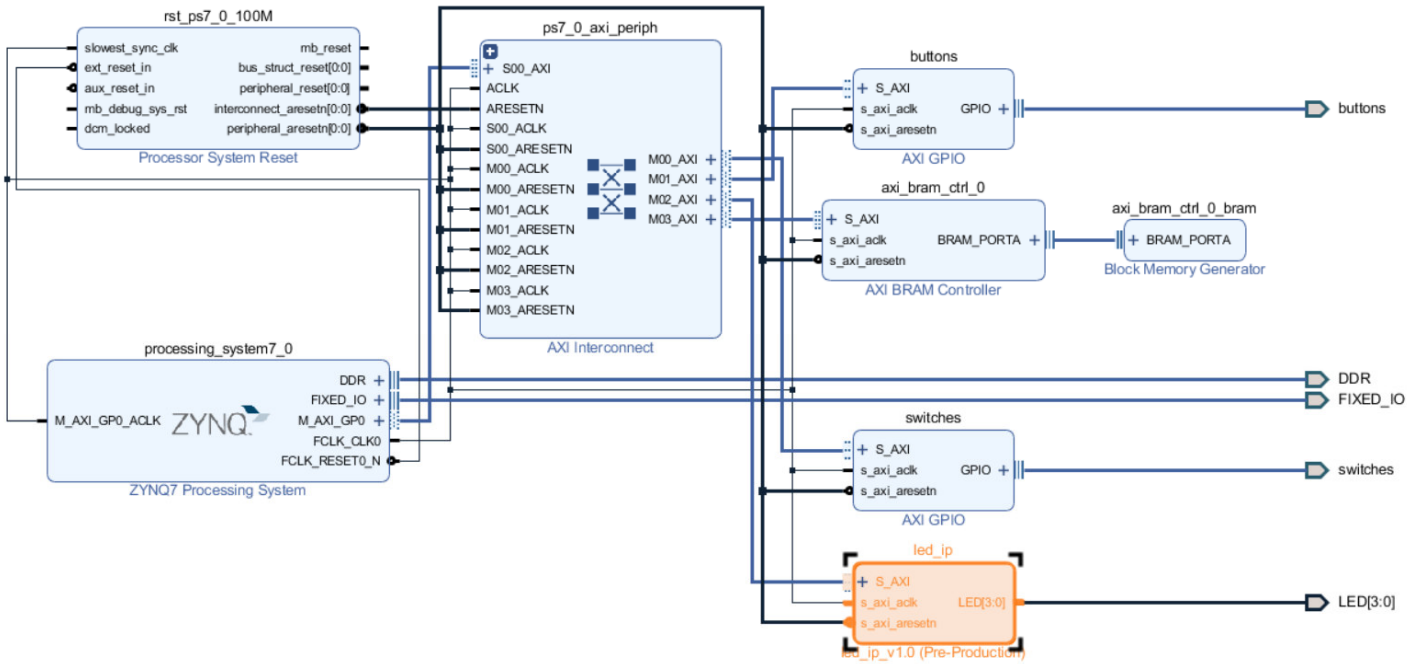

Figure 16: Validated block design of the custom LED IP

13. Now the constraints file needs to be added to the design. Add a constraints file called ***led_ip_const*** to the project. Add the LEDs to the constraints file. Look back at past labs as a reference if needed.

14. Right click on ***system.bd*** and select **Generate output products** and click **OK**.

15. Click on **Generate Bitstream** in the **Flow Navigator** and click **Yes** if prompted to save, and click **Yes** again if prompted to launch **Synthesis and Implementation**. Click **Cancel** when the **Bitstream** generation is complete.

16. Now it is time to Export the hardware to the **SDK**. Click **File → Export → Export Hardware**. Click **Include the bitstream** and then click **Yes** to overwrite.

17. Select **File > Launch SDK** and click **OK**.

18. Now that the **SDK** has launched, close any unused projects in the **Project Explorer**.

19. Select **File → New → Application Project**. Enter ***led_ip*** as the project name and choose **Create New** board support package. Click **Next** and select **Empty Application** and click **Finish**.

20. Expand ***led_ip*** in the project view and right-click on the **src** folder and select **New → Source File** and enter ***led_ip.c*** as the file name. Click **Finish** to add the empty source file to the project.

21. Copy the following code into the empty **C** source file. You will need to complete the code to output the dip switch value to the LEDs. The ***led_ip.h*** includes the function you need to use to complete this code.

```
#include <stdio.h>
#include <stdlib.h>
#include "xil_printf.h"
#include "xparameters.h"
#include "xgpio.h"
#include "led_ip.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_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);

      // output dip switches value on LED_ip device

      for (i=0; i<9999999; i++);
   }
}
```

Figure 17: SDK source code for the LED IP

22. Now you need to make sure that the board support package includes the correct drivers for the new **IP**. Select ***led_ip_bsp*** in the project view, right-click and select **Board Support Package Settings**. Select **Drivers** in the **Overview** file tree and search for ***led_ip*** in the component column. Make sure that the driver column says ***led_ip***. If it says **Generic**, click on it and select ***led_ip*** from the given drop-down menu. Click **OK** if you made a change.

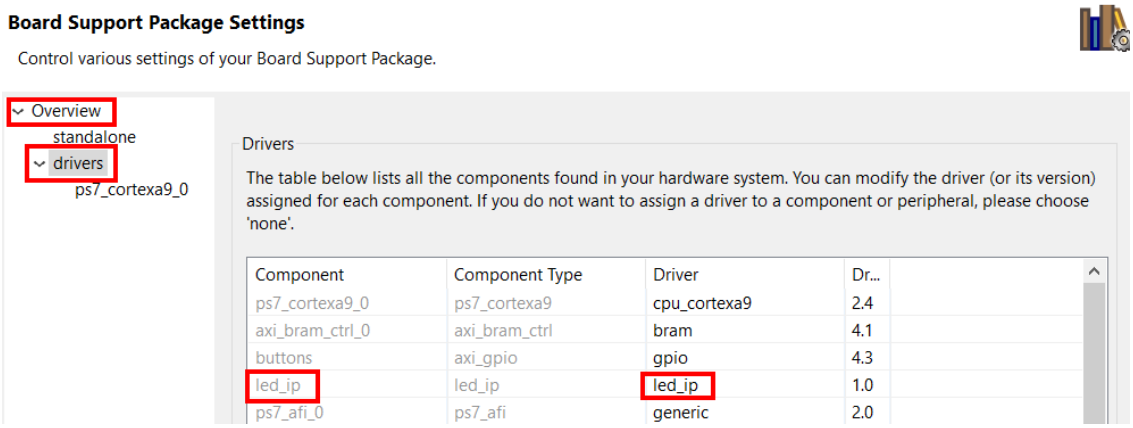

Figure 18: Board support package drivers

23. Save to recompile the project.

24. Now it is time to verify the **IP** on hardware. Connect the **PYNQ** board with the micro-usb and power it on.

25. Open a serial terminal of your choice and setup the port with a 115200 baud rate.

26. The **FPGA** needs to be programed with the system wrapper bitstream generated in step 15. Remember, the memory on an **FPGA** is volatile and needs to be reprogrammed every time the board powers down.

27. Select **Xilinx Tools → Program FPGA**. Make sure the correct hardware platform is selected and click **Program**. The green LED on the **PYNQ** board will light up when finished.

28. Select **Run → Debug As → Launch on Hardware (System Debugger)**.

29. Verify that the LEDs light up the correct binary value that you select on the dip switches. The numerical value of the dip switches and the buttons should display on the terminal window as well.

Part III – Custom Seven Segment Display IP – On your own

In this part, you will take what you have learned up to this point and apply it to the creation of a seven segment display **IP** core. Use the first two parts of this lab as a tutorial for creating the seven segment display **IP**. There will be many similarities in the design of this **IP**, but there will be a few challenges for you to overcome. This part will build upon parts I and II. By the end of part III, you will have a serial output of the buttons and switches, LED values of the dip switches, and a single seven segment display showing the value of the dip switches.

Application Behavior

1. The binary representation of the dip switches should display on the serial monitor.

2. The binary value of the buttons should display on the serial monitor. (ex. 1, 2, 4, 8)
3. The binary representation of the dip switches should display on the LEDs.
4. The binary representation of the dip switches should display on the seven segment display.

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
- State Transition Diagram (depicting the flow of your state machine)
- Simulation printouts (highlighting various conditions that you have tested)

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