Adding Custom IP to the System

Introduction

This lab guides you through the process of creating and adding a custom peripheral to a processor system by using the Vivado IP Packager. You will create an AXI4Lite interface peripheral.

Objectives

After completing this lab, you will be able to:

- Use the IP Packager feature of Vivado to create a custom peripheral
- Modify the functionality of the IP
- Add the custom peripheral to your design
- Add pin location constraints
- Add block memory to the system

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 4 primary steps: You will use a peripheral template to create a peripheral, Package the IP using IP Packager, import, add and connect the IP in the design, and add the Block RAM (BRAM) Memory.

Design Description

You will extend the Lab 2 hardware design by creating and adding an AXI peripheral (refer to LED_IP in **Figure 1**) to the system, and connecting it to the LEDs on the Zynq board you are using. You will use the IP Packager to generate the custom IP. Next, you will connect the peripheral to the system and add pin location constraints to connect the LED display controller peripheral to the on-board LED display. Finally, you will add BRAM Controller and BRAM before generating the bitstream.

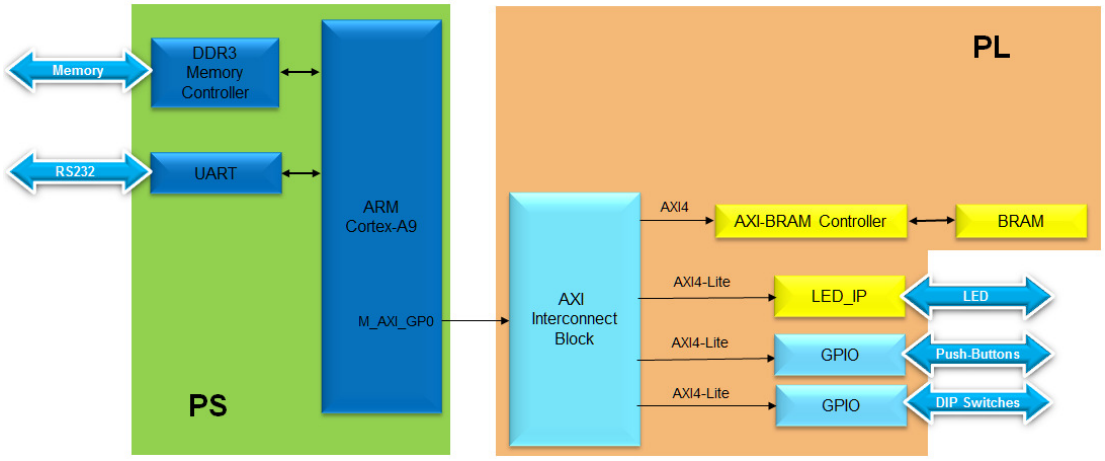
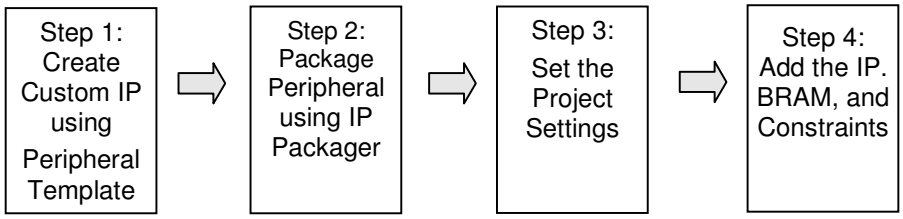


Figure 1. Design Updated from Previous Lab

General Flow for this Lab



In the instructions below;
{***sources***} refers to: EmbeddedSystem_labs\lab3
{***labs***} refers to : EmbeddedSystem_labs\lab3
{***labsolutions***} or for the Zybo refers to: EmbeddedSystem_labs\lab2

Create a Custom IP using the Create and Package IP Wizard Step 1

1-1. Use the provided axi_lite slave peripheral template and the custom IP source code to create a custom IP.

1-1-1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2015.4 > Vivado 2015.4**

1-1-2. Click **Manage IP** and select *New IP Location* and click **Next** in the *New IP Location* window

1-1-3. Select **VHDL** as the *Target Language*, **Mixed** as the *Simulator language*, and for *IP location*, type *{labs}\led_ip* and click **Finish** (leave other settings as defaults and click **OK** if prompted to create the directory)

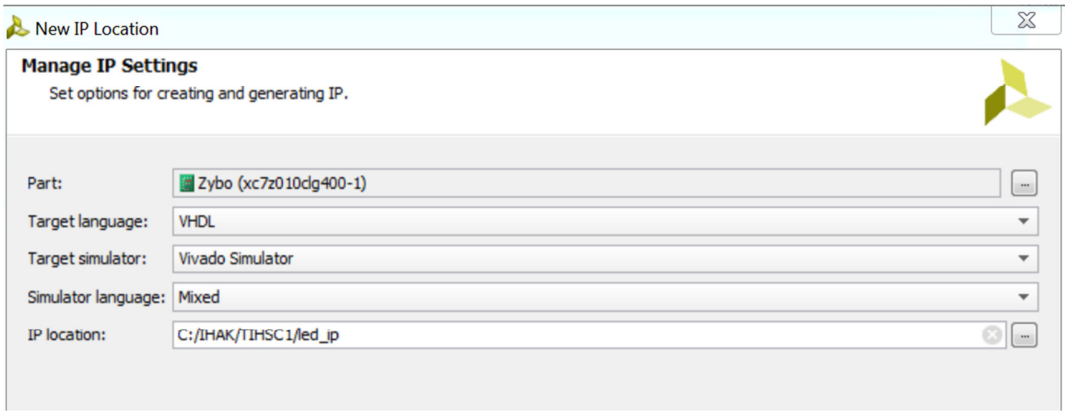


Figure 2. New IP Location form

A Virtex 7 part is chosen for this project, but later compatibility for other devices will be added to the packaged IP.

1-2. Run the Create and Package IP Wizard

1-2-1. Select *Tools > Create and Package IP*

1-2-2. In the window, click **Next**

1-2-3. Select *Create new AXI4 peripheral*, specify the *IP Definition location* as *{labs}\led_ip* and click **Next**

1-2-4. Fill in the details for the IP

Name: **led_ip**
Display Name: **led_ip_v1_0**
(Fill in a description, Vendor Name, and URL)

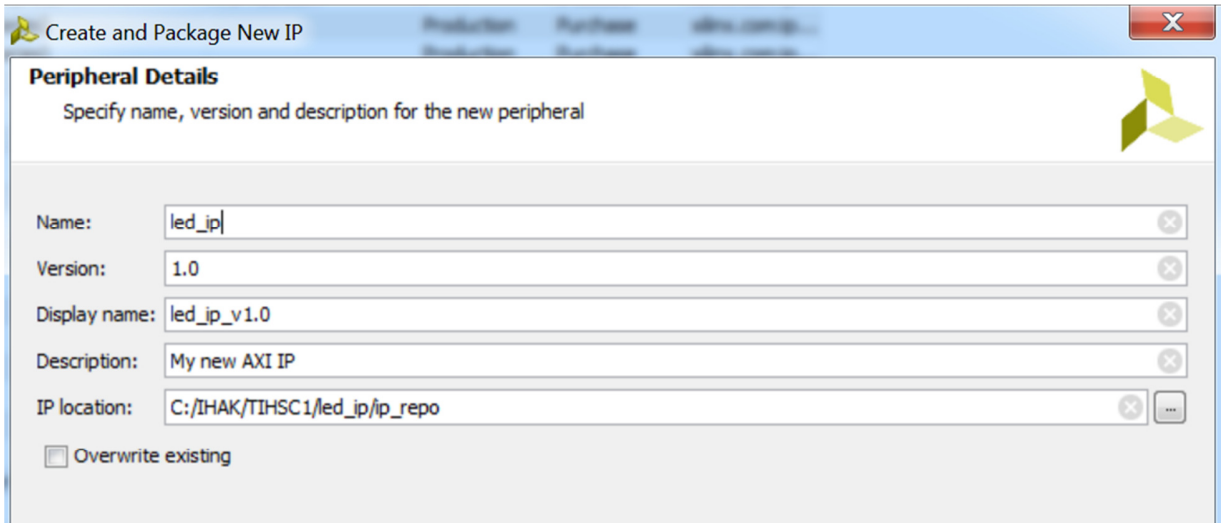


Figure 3. Updating Peripheral Details form

1-2-5. Click **Next**

1-2-6. Change the Name of the interface to **S_AXI**

1-2-7. Leave the other settings as default and click **Next** (Lite interface, Slave mode, Data Width 32, Registers 4)

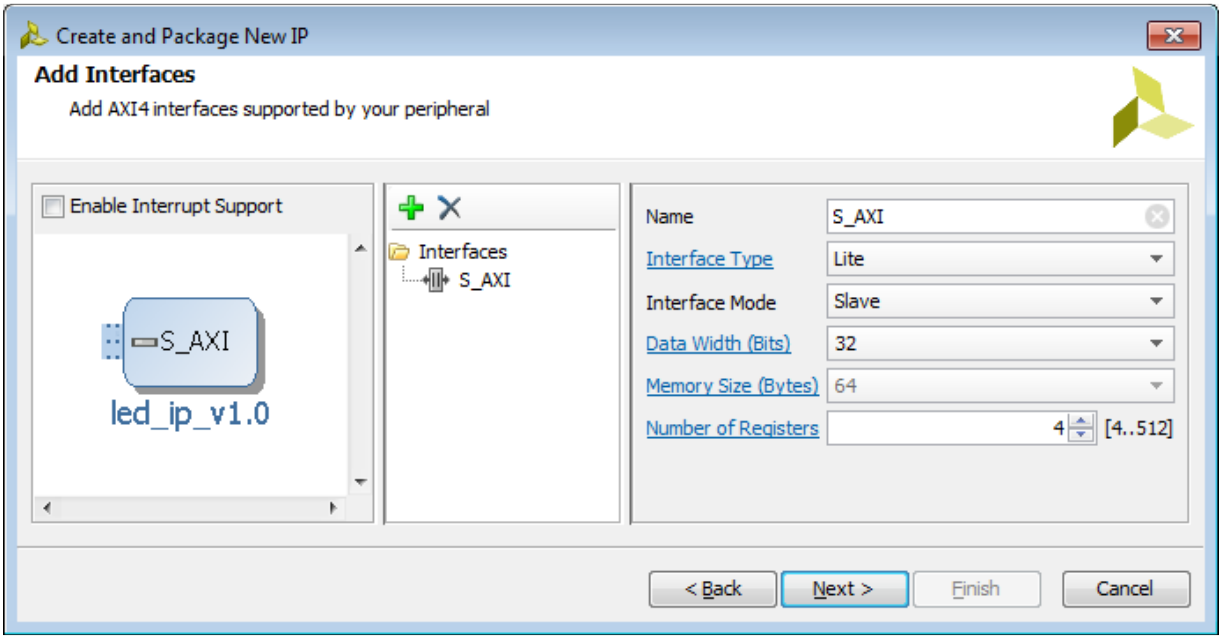


Figure 4. Naming the AXI interface

1-2-8. Select *Edit IP* and click **Finish** (a new Vivado Project will open)

1-3. Create an interface to the LEDs

1-3-1. In the sources panel, double-click the **led_ip_v1_0.vhd** file.

This file contains the HDL code for the interface(s) selected above. The top level file contains a module which implements the AXI interfacing logic, and an example design to write to and read from the number of registers specified above. This template can be used as a basis for creating custom IP. A new parameterized output port to the LEDs will be created at the top level of the design, and the AXI write data in the sub-module will be connected back up to the external LED port.

Scroll down to line 8 where a user *parameters* space is provided.

1-3-2. Add the line:

```
    LED_WIDTH : integer := 8;
```

1-3-3. Go to line 19 and add the line:

```
    LED : out std_logic_vector(LED_WIDTH-1 downto 0);
```

(Notice the extra semicolon needed at the end of each line)

c:/ihak/tihsc1/led_ip/ip_repo/led_ip_1.0/hdl/led_ip_v1_0.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity led_ip_v1_0 is
6      generic (
7          -- Users to add parameters here
8          LED_WIDTH : integer := 8;
9          -- User parameters ends
10         -- Do not modify the parameters beyond this line
11
12
13         -- Parameters of Axi Slave Bus Interface S_AXI
14         C_S_AXI_DATA_WIDTH  : integer   := 32;
15         C_S_AXI_ADDR_WIDTH  : integer   := 4
16     );
17     port (
18         -- Users to add ports here
19         LED : out std_logic_vector(LED_WIDTH-1 downto 0);
20         -- User ports ends
21         -- Do not modify the ports beyond this line
22
23
24         -- Ports of Axi Slave Bus Interface S_AXI
25         s_axi_aclk  : in std_logic;
```

Figure 5. Adding users parameter, and port definition

1-3-4. Insert the following at line ~51:

```
LED_WIDTH : integer := 8;
```

1-3-5. Insert the following at line ~59:

```
LED : out std_logic_vector(LED_WIDTH-1 downto 0);
```

1-3-6. Insert the following at line ~89:

```
LED_WIDTH => LED_WIDTH,
```

1-3-7. Insert the following at line ~94:

```
LED => LED,
```

c:/ihak/tihsc1/led_ip/ip_repo/led_ip_1.0/hdl/led_ip_v1_0.vhd

```
45          s_axi_rready    : in std_logic
46      );
47  end led_ip_v1_0;
48
49  architecture arch_imp of led_ip_v1_0 is
50
51      -- component declaration
52      component led_ip_v1_0_S_AXI is
53          generic (
54          LED_WIDTH : integer := 8;
55          C_S_AXI_DATA_WIDTH  : integer    := 32;
56          C_S_AXI_ADDR_WIDTH  : integer    := 4
57          );
58          port (
59          LED : out std_logic_vector(LED_WIDTH-1 downto 0);
60          S_AXI_ACLK  : in std_logic;
61          S_AXI_ARESETN   : in std_logic;
```

.....

```
86  -- Instantiation of Axi Bus Interface S_AXI
87  led_ip_v1_0_S_AXI_inst : led_ip_v1_0_S_AXI
88      generic map (
89          LED_WIDTH => LED_WIDTH,
90          C_S_AXI_DATA_WIDTH  => C_S_AXI_DATA_WIDTH,
91          C_S_AXI_ADDR_WIDTH  => C_S_AXI_ADDR_WIDTH
92      )
93      port map (
94          LED => LED,
95          S_AXI_ACLK  => s_axi_aclk,
96          S_AXI_ARESETN   => s_axi_aresetn,
97          S_AXI_AWADDR    => s_axi_awaddr,
98          S_AXI_AWPROT    => s_axi_awprot,
```

Figure 6. Adding port connection with a lower-level module

1-3-8. Save the file by selecting **File > Save File**

1-3-9. Expand *led_ip_v1_0* in the sources view if necessary, and open **led_ip_v1_0_S_AXI.vhd**

1-3-10. Add the LED parameter and port to this file too, at lines 8 and 19

c:/ihak/tihsc1/led_ip/ip_repo/led_ip_1.0/hdl/led_ip_v1_0_S_AXI.vhd

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity led_ip_v1_0_S_AXI is
6     generic (
7         -- Users to add parameters here
8         LED_WIDTH : integer := 8;
9         -- User parameters ends
10        -- Do not modify the parameters beyond this line
11
12        -- Width of S_AXI data bus
13        C_S_AXI_DATA_WIDTH  : integer   := 32;
14        -- Width of S_AXI address bus
15        C_S_AXI_ADDR_WIDTH  : integer   := 4
16    );
17    port (
18        -- Users to add ports here
19        LED : out std_logic_vector(LED_WIDTH-1 downto 0);
20        -- User ports ends
21        -- Do not modify the ports beyond this line
22
23        -- Global Clock Signal
24        S_AXI_ACLK  : in std_logic;
25        -- Global Reset Signal. This Signal is Active LOW
```

Figure 7. Declaring users port in the lower-level module for the Zybo

1-3-11. Scroll down to ~line 382 and insert the following code to control the LEDS. They are controlled by writing to the first 4 bits of register 0 that is memory mapped to the first address of the led_ip core.

```
LED <= slv_reg0(LED_WIDTH-1 downto 0);
```

```
361
362     -- Output register or memory read data
363     process( S_AXI_ACLK ) is
364     begin
365       if (rising_edge (S_AXI_ACLK)) then
366         if ( S_AXI_ARESETN = '0' ) then
367           axi_rdata  <= (others => '0');
368         else
369           if (slv_reg_rden = '1') then
370             -- When there is a valid read address (S_AXI_ARVALID) with
371             -- acceptance of read address by the slave (axi_arready),
372             -- output the read dada
373             -- Read address mux
374               axi_rdata <= reg_data_out;     -- register read data
375           end if;
376         end if;
377       end if;
378     end process;
379
380
381     -- Add user logic here
382     LED <= slv_reg0(LED_WIDTH-1 downto 0);
383     -- User logic ends
384
385 end arch_imp;
```

Figure 8. Writing to LEDS from contents of register 0

Check all the signals that are being connected and where they originate.

1-3-12. Save the file by selecting **File > Save File**

1-3-13. Click **Run Synthesis** and *Save* if prompted. (This is to check the design synthesizes correctly before packaging the IP. If this was your own design, you would simulate it and verify functionality before proceeding)

1-3-14. **Check the *Messages* tab for any errors and correct if necessary before moving to the next step**

When Synthesis completes successfully, click **Cancel**.

1-4. Package the IP

1-4-1. Click on the **Package IP – led_ip** tab

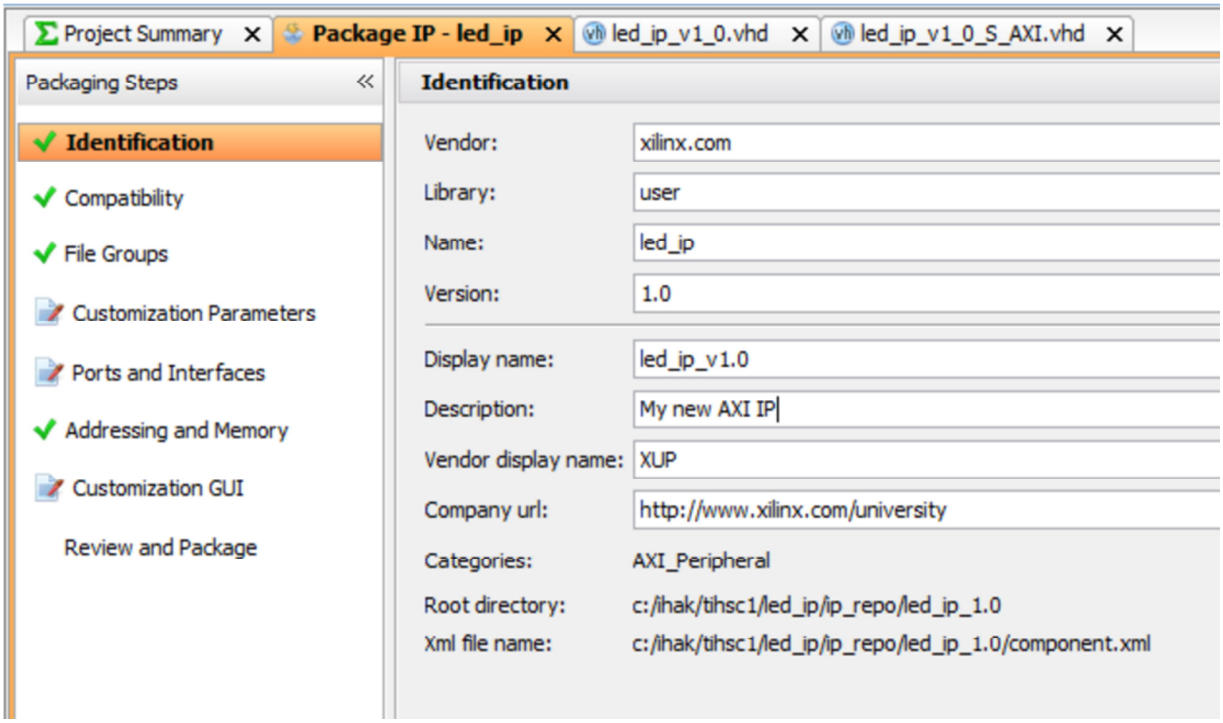


Figure 9. Package IP

1-4-2. For the IP to appear in the IP catalog in particular categories, the IP must be configured to be part of those categories. To change which categories the IP will appear in the IP catalog click the browse box on the *Categories* line. This opens the Choose IP Categories window

1-4-3. For the purpose of this exercise, uncheck the **AXI Peripheral** box and check the **IO Interfaces** and click **OK**.

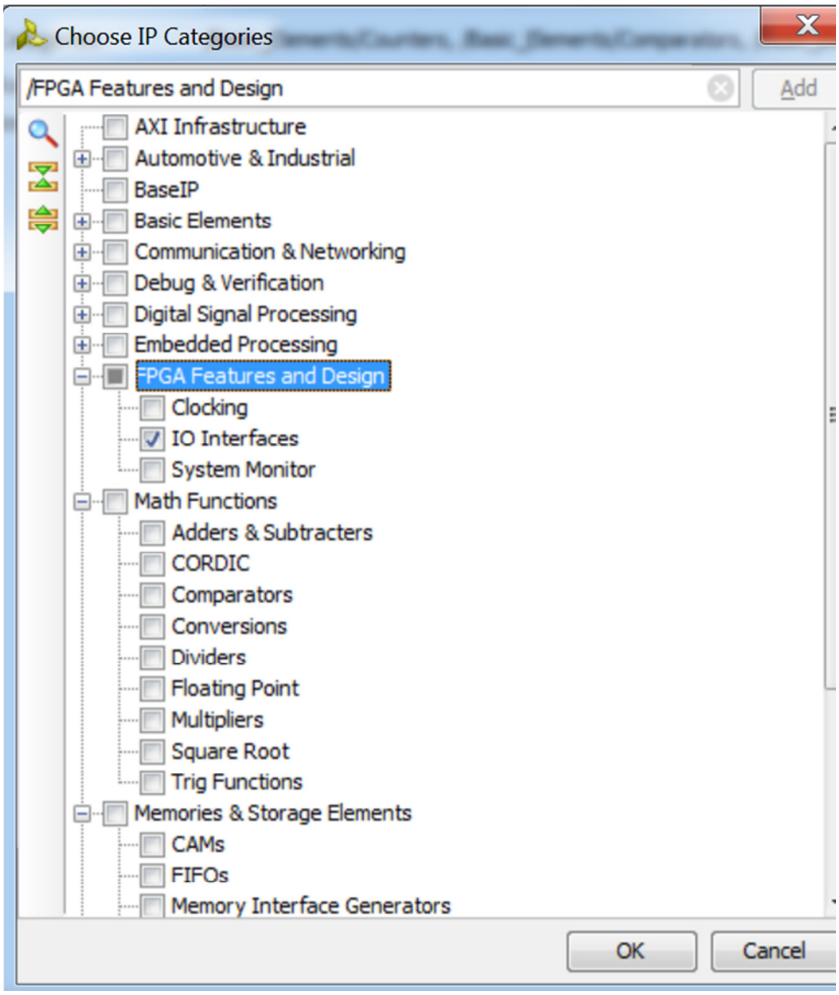


Figure 10. Specify the category for IP Packager IP

1-4-4. Select **IP Compatibility**. This shows the different Xilinx FPGA Families that the IP supports. The value is inherited from the device selected for the project.

1-4-5. Right click in the *Family Support table* and select **Add Family…** from the menu.

1-4-6. Select the **Zynq** family as we will be using this IP on the Zybo board, and click **OK**.

1-4-7. You can also customize the address space and add memory address space using the **IP Addressing and Memory** category. We won't make any changes.

1-4-8. Click on **IP File Groups** *and click Merge changes from IP File Groups Wizard*

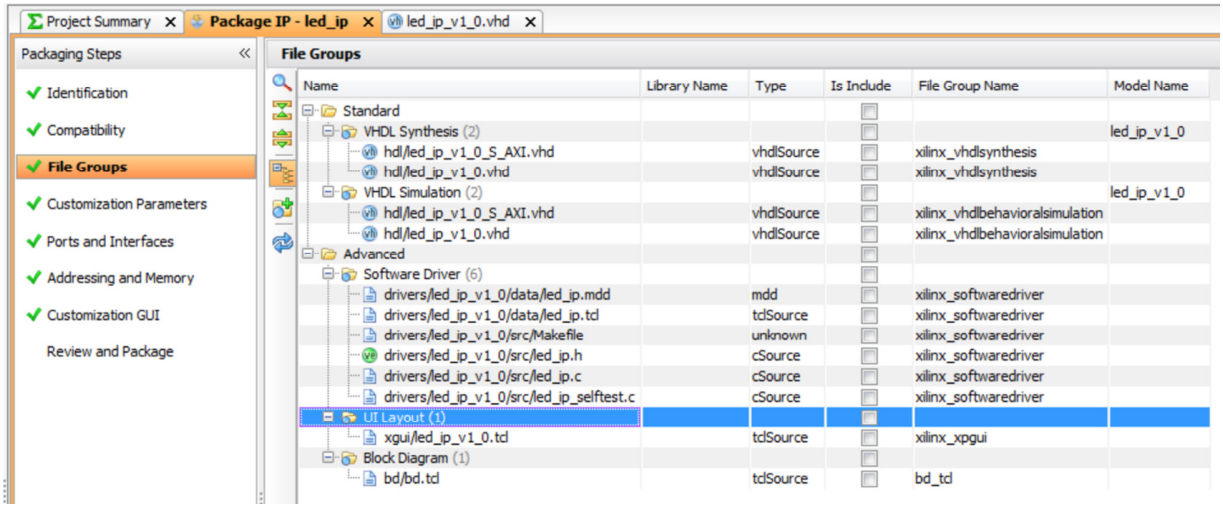


Figure 11. Updating the file group

1-4-9. Click on IP Customization Parameters and again *Merge changes from IP Customization Parameters Wizard*

Notice that the *IP Ports* view now shows the user created *LED* port

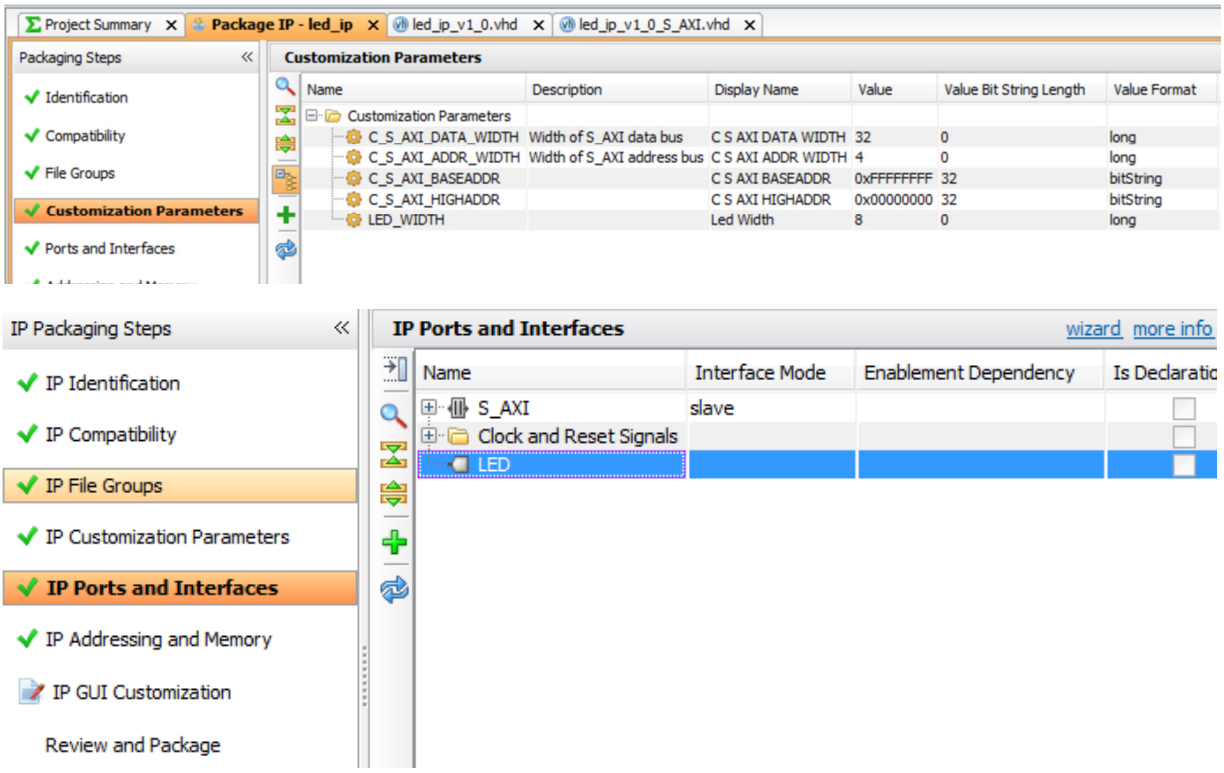


Figure 12. User parameter and port imported

1-4-10. Select Customization GUI and Add Parameter LED_WIDTH as shown in figure 13.

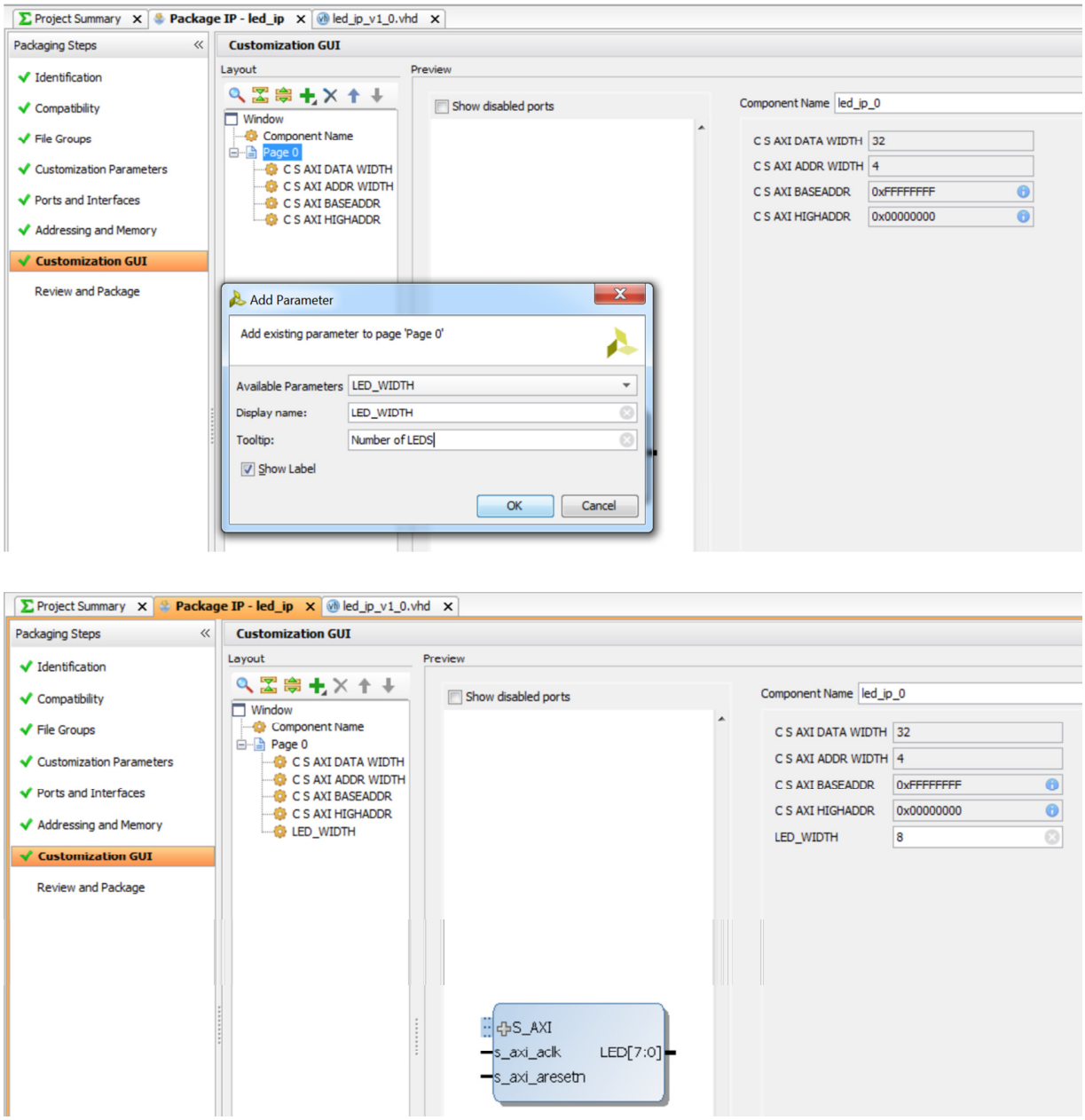


Figure 13. GUI customization adding LED_WIDTH

1-4-11. Select *Review and Package*, and notice the path where the IP will be created.

1-4-12. Click **Re-Package IP**. The project will close when complete.

1-4-13. In the original Vivado window click **File > Close Project**

Modify the Project Settings Step 3

2-1. Open the lab2 project from the EmbeddedSystems_labs directory, and save the project as lab3. Set Project Settings to point to the created IP repository.

2-1-1. Start the Vivado if necessary and open either the lab2 project you created in the previous lab or the lab2 project in the labsolution directory

2-1-2. Select **File > Save Project As …** to open the *Save Project As* dialog box. Enter **lab3** 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 lab3 directory and save the project and associated directory with lab3 name.

2-1-3. Click **Project Settings** in the *Flow Navigator* pane.

2-1-4. Select **IP** in the left pane of the *Project Settings* form.

2-1-5. Click on the **Add Repository…** button, browse to **{labs}\led_ip** and click **Select**.

The led_ip_v1_0 IP will appear the **IP in the Selected Repository** window.

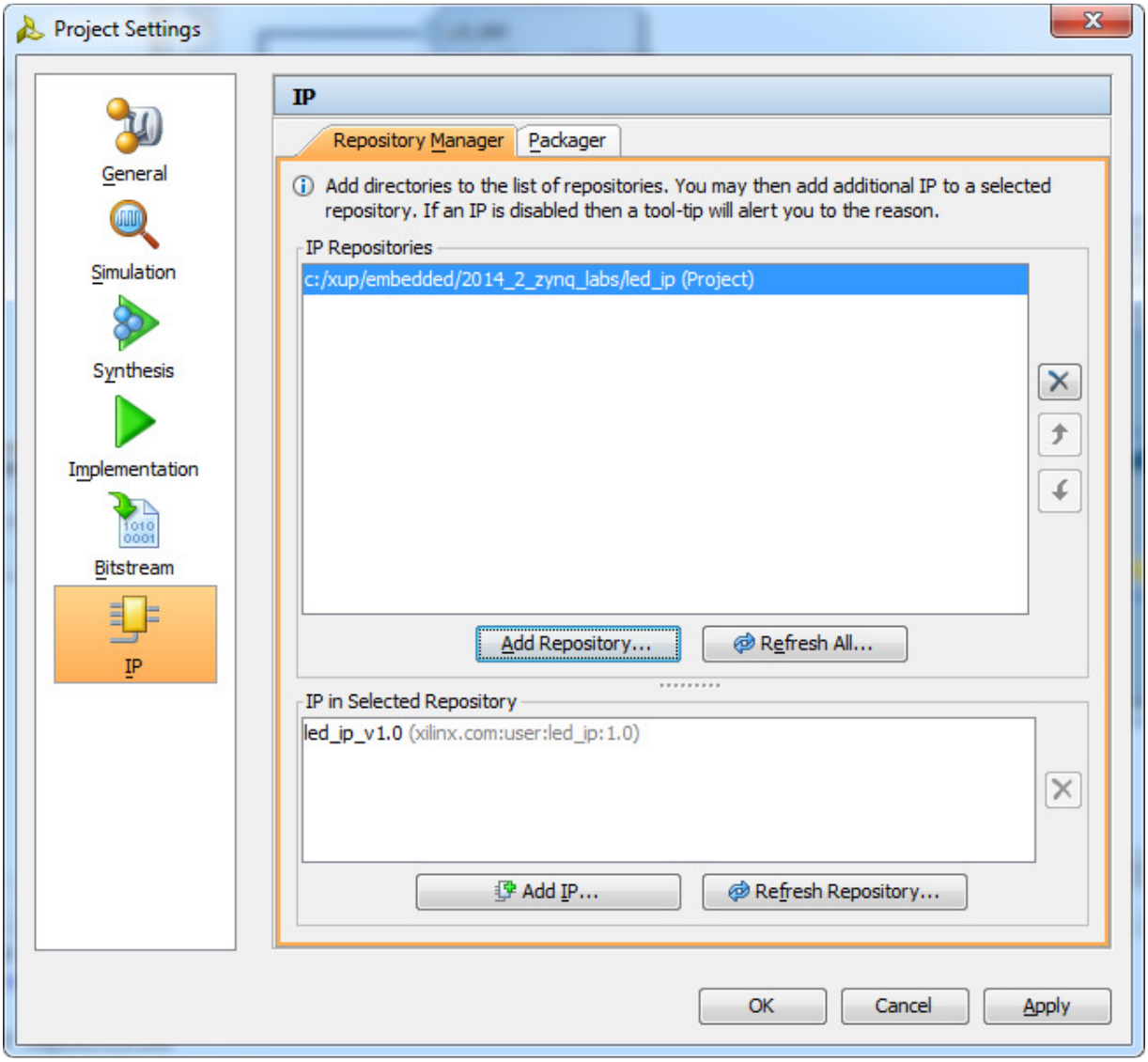


Figure 13. Specify IP Repository

2-1-6. Click **OK**.

Add the Custom IP, BRAM, and the Constraints Step 4

3-1. Add led_ip to the design and connect to the AXI4Lite interconnect in the IPI. Make internal and external port connections. Establish the LED port as external FPGA pins.

3-1-1. Click **Open Block Design** under **IP Integrator** in the Flow Navigator pane

3-1-2. Click the Add IP icon and search for **led_ip_v1_0** in the catalog by typing “led” in the search field.

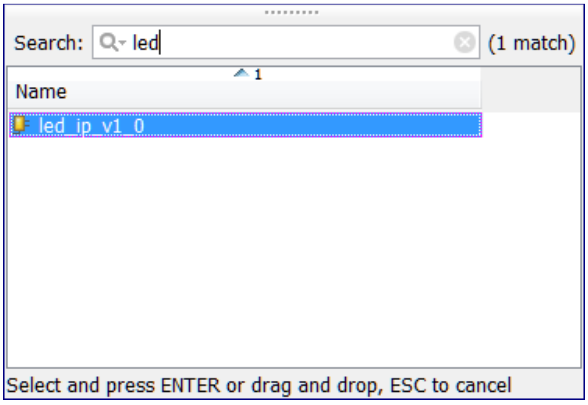


Figure 12. Searching for led_ip in the IP Catalog

3-1-3. Double-click **led_ip_v1_0** to add the core to the design.

3-1-4. Select the IP in the block diagram and change the instance name to **led_ip** in the properties view.

3-1-5. Double click the block to open the configuration properties

3-1-6. For the ZedBoard, leave the *Led Width* set to 8, or for the Zybo, set the width to 4

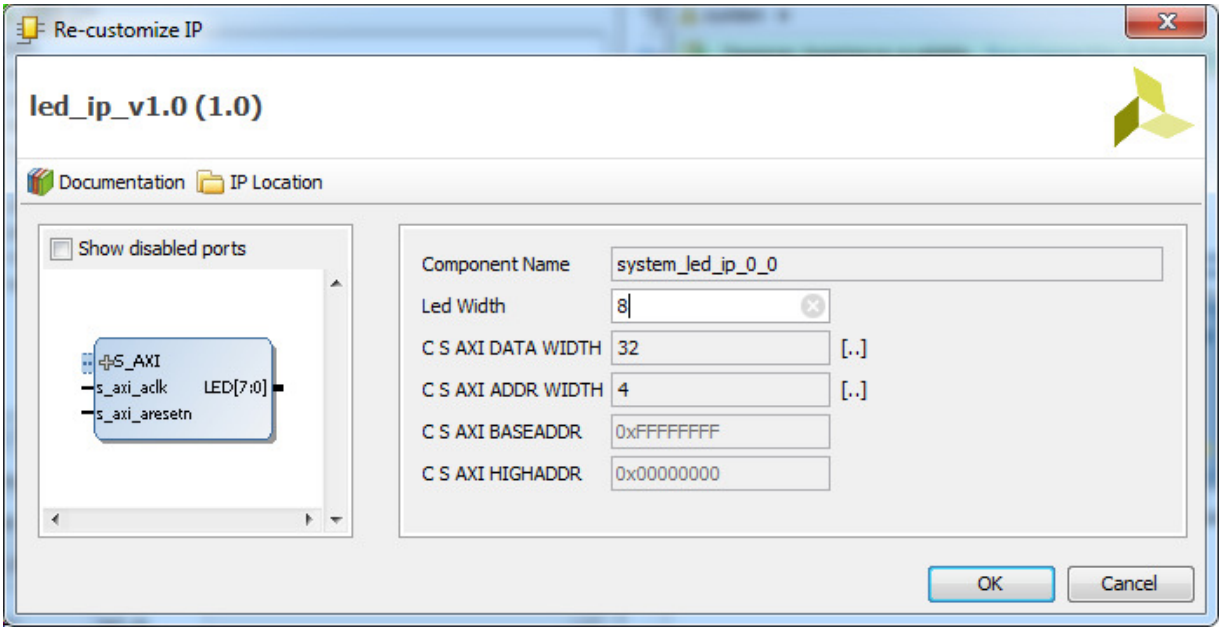


Figure 13. Configure the LED IP LED_WIDTH

3-1-7. Click on **Run Connection Automation**, select **/led_ip/S_AXI** and click **OK** to automatically make the connection from the AXI Interconnect to the IP.

Click the regenerate button ( ) to redraw the diagram.

3-1-8. Select the *LED* port on the led_ip instance (by clicking on its pin), right-click and select **Make External**.

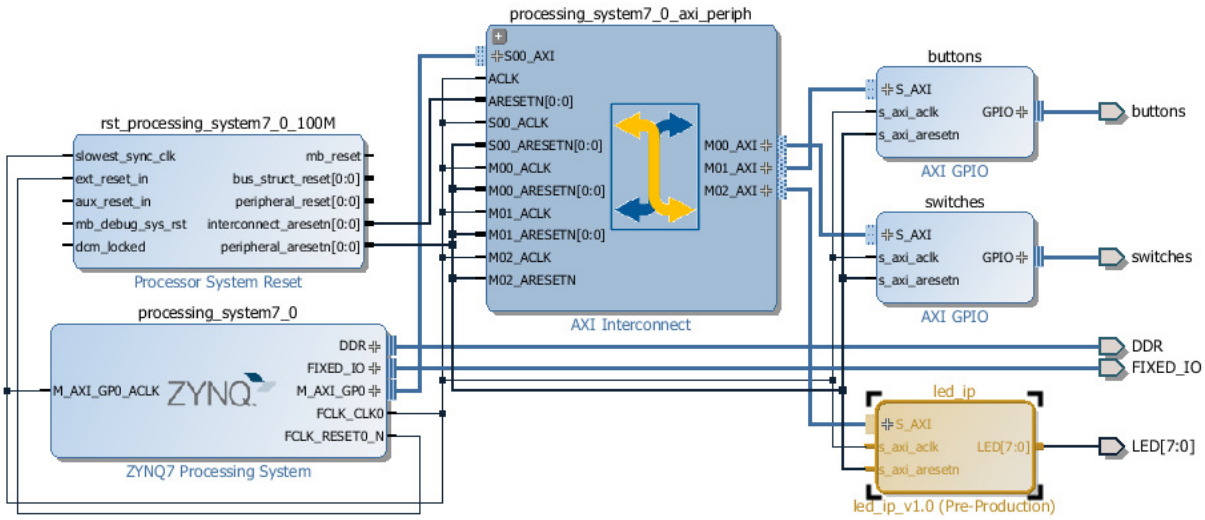


Figure 14. LED external port added and connected

3-1-9. Select the **Address Editor** tab and verify that an address has been assigned to *led_ip*.

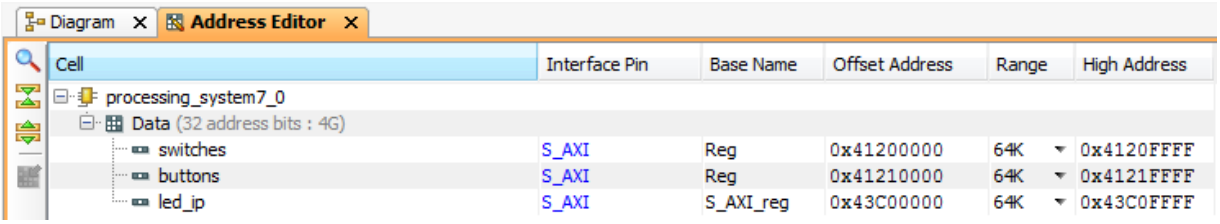


Figure 15. Address assigned for led_ip

3-1-10. Change the **AXI GPIO leds** to control **buttons** instead of leds as shown in figure 14.

3-1-11. Remove the external MIO btn pin from the ZYNQ processing system as illustrated in figure 14.

3-2. Add BRAM to the design

3-2-1. In the Block Diagram, click the Add IP icon and search for BRAM and add one instance of the **AXI BRAM Controller**

3-2-2. Run *Connection Automation* on **axi_bram_ctrl_0/S_AXI** and click **OK** when prompted to connect it to the **M_AXI_GP0 Master.**

3-2-3. Double click on the block to customize it and change the number of BRAM interfaces to 1 and click **OK**.

Notice that the AXI Protocol being used is AXI4 instead of AXI4Lite since BRAM can provide higher bandwidth and the controller can support burst transactions.

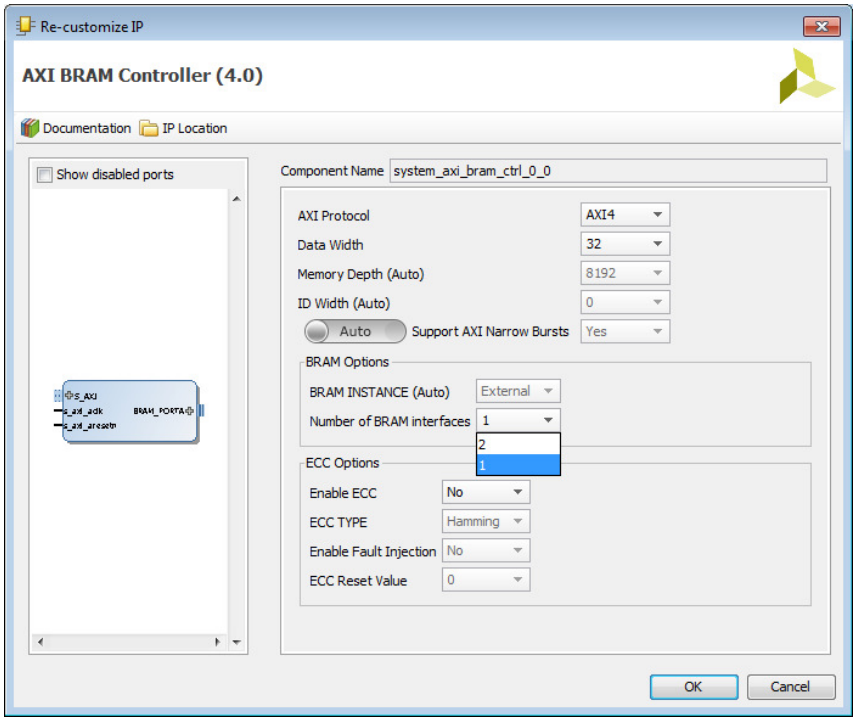


Figure 16. Customize BRAM controller

3-2-4. Click on *Run Connection Automation* to add and connect a **Block Memory Generator by selecting axi_bram_ctrl_0/BRAM_PORTA and click OK** (This could be added manually)

3-2-5. Validate the design to ensure there are no errors (F6), and click the regenerate button ( ) to redraw the diagram.

The design should look similar to the figure below.

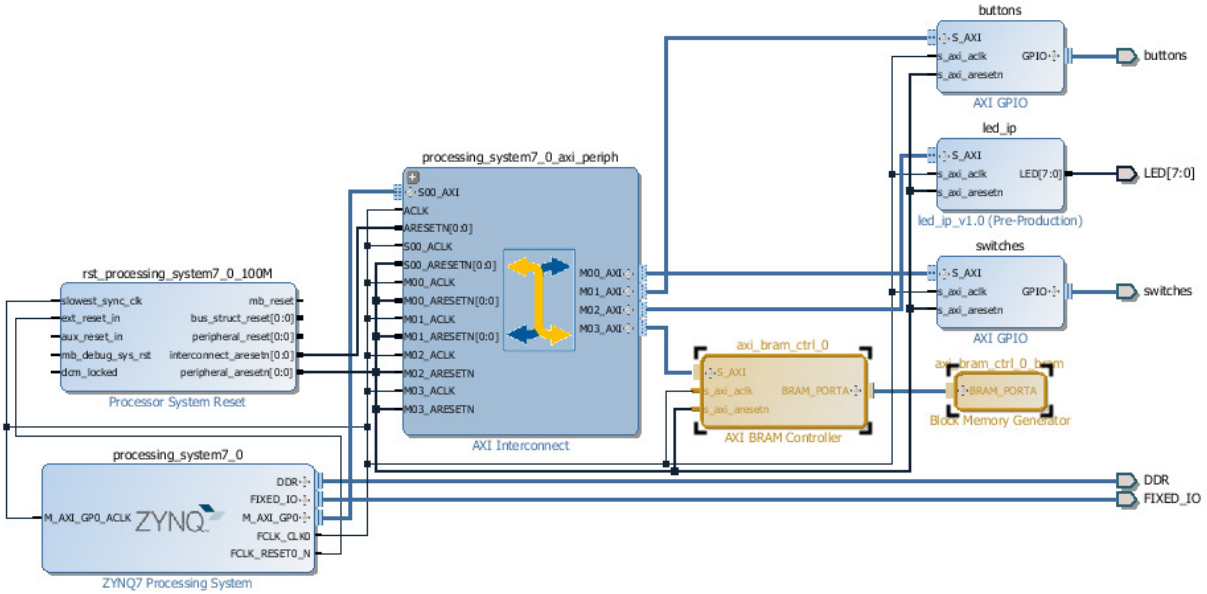


Figure 17. Completed Block Diagram

3-2-6. In the Address editor, increase the Range of the **axi_bram_ctrl_0** to **8K**

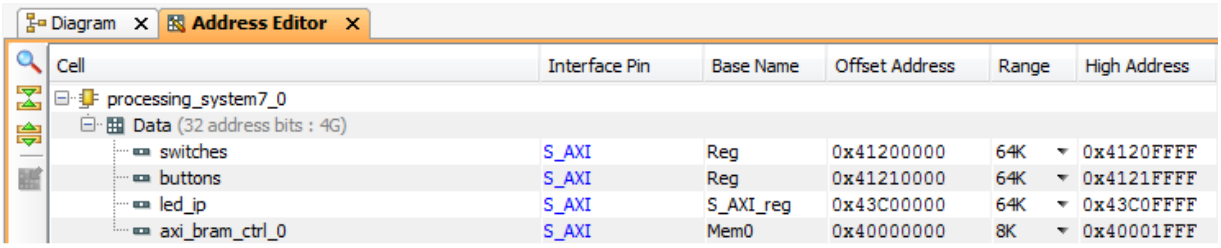


Figure 20. Adjusting memory size

3-2-7. Press **F6** to validate the design one last time.

3-3. Update the top-level wrapper and add the provided lab3_*.xdc constraints file.

3-3-1. Click **Add Sources** in the *Flow Navigator* pane, select **Add or Create Constraints**, and click **Next**.

3-3-2. Click the **Add Files** button, browse to the {**sources**}**\lab3** folder, select **lab3_zed.xdc** for the ZedBoard, or **lab3_Zybo.xdc** for the Zybo

3-3-3. Click **Finish** to add the file.

3-3-4. Expand Constraints folder in the *Sources* pane, and double click the **lab3_*.xdc** file entry to see its content. This file contains the pin locations and IO standards for the LEDs on the Zynq board. This information can usually be found in the manufacturer's datasheet for the board.

3-3-5. Right click on *system.bd* and select *Generate output products*

3-3-6. Click on **Generate Bitstream** and click **Yes** if prompted to save the Block Diagram, and click **Yes** again if prompted to launch synthesis and implementation. Click **Cancel** when prompted to *Open the Implemented Design*

Conclusion

Vivado IP packager was used to import a custom IP block into the IP library. The IP block was then added to the system. Connection automation was run where available to speed up the design of the system by allowing Vivado to automatically make connections between IP. An additional BRAM was added to the design. Finally, pin location constraints were added to the design.