Adding Custom IP to the System

Objectives

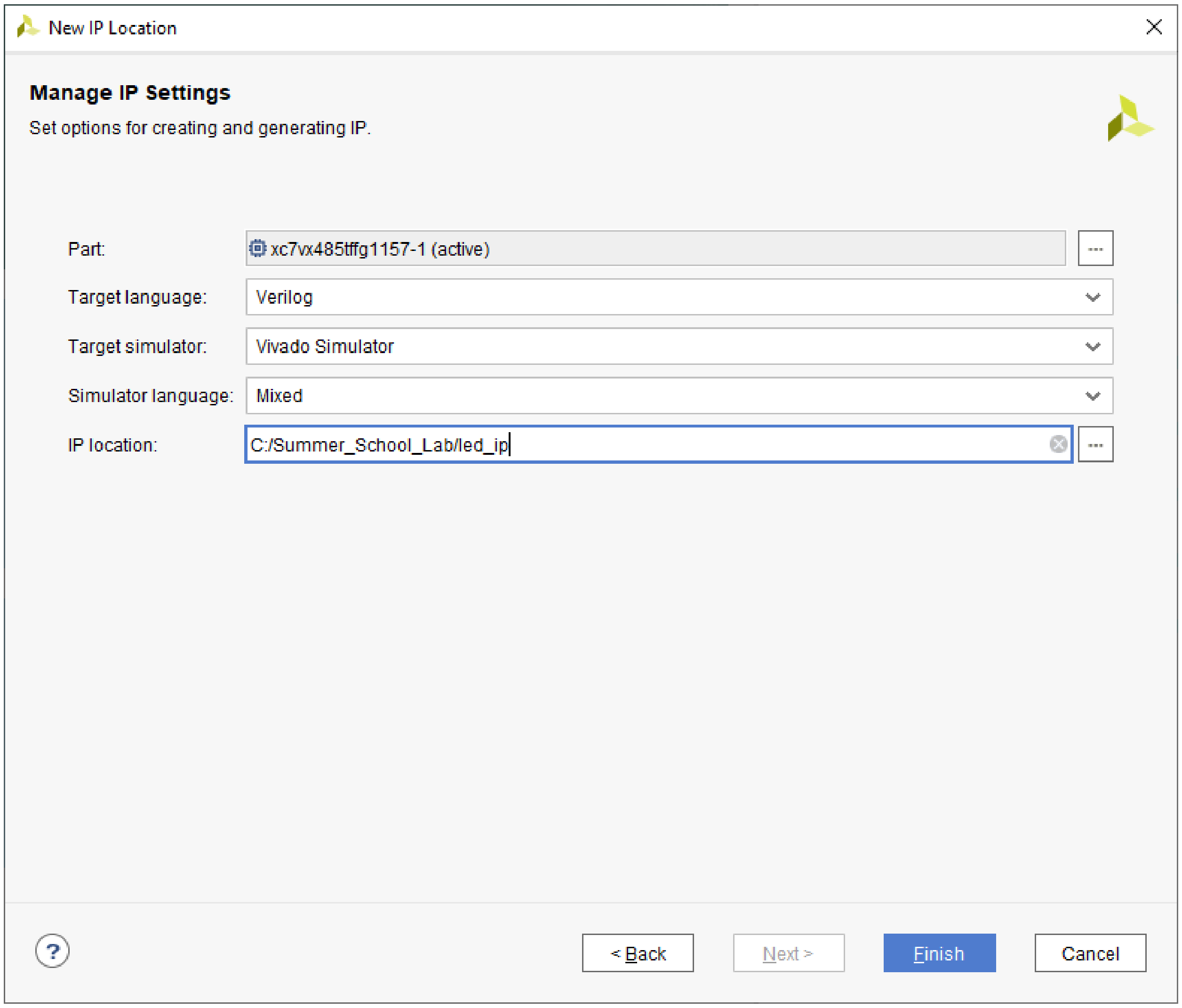
After completing this lab, you will be able to:

- 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
- Write a basic application to access an IP peripheral in SDK
- Generate an elf executable file
- Download the bitstream and application and verify on a Zynq board

Steps

Create a Custom IP using the Create and Package IP Wizard

1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2018.2 > Vivado 2018.2**
2. Click **Manage IP** and select New IP Location and click Next in the New IP Location window
3. Select Verilog 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)



Manage IP Settings

Set options for creating and generating IP.

Part:	xc7vx485tffg1157-1 (active)
Target language:	Verilog
Target simulator:	Vivado Simulator
Simulator language:	Mixed
IP location:	C:/Summer_School_Lab/led_ip

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.

Run the Create and Package IP Wizard

1. Select **Tools > Create and Package New IP**
2. In the window, click Next
3. Select **Create a new AXI4 peripheral**, and click Next
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)

5. Click Next
6. Change the Name of the interface to **S_AXI**
7. Leave the other settings as default and click Next (*Lite interface, Slave mode, Data Width: 32, Number of Registers: 4*)
8. Select Edit IP and click Finish (a new Vivado Project will open)

Create an interface to the LEDs

1. In the sources panel, double-click the led_ip_v1_0.v 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 7 where a user parameters space is provided.

2. Add the line (7 means line number, don't copy to file):

```
7   parameter integer LED_WIDTH         = 8,
```

3. Go to line 18 and add the line (Don't forget to add commas):

```
18    output wire [LED_WIDTH-1:0]      LED,
```

4. Insert the following at line ~48:

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

5. Insert the following at line ~52:

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

6. Save the file by right click and select **Save File**

7. Expand led_ip_v1_0 in the sources view if necessary, and open led_ip_v1_0_S_AXI.v

8. Add the LED parameter and port to this file too, at lines 7 and 18 (done in steps 2 and step3)

9. Scroll down to ~line 400 and insert the following code to instantiate the user logic for the LED IP (This code can be typed directly, or copied from the user_logic_instantiation.txt file in the lab3 source folder.)

```
lab3_user_logic  # (
    .LED_WIDTH(LED_WIDTH)
  )
 U1(
     .S_AXI_ACLK(S_AXI_ACLK),
     .slv_reg_wren(slv_reg_wren),
     .axi_awaddr(axi_awaddr[C_S_AXI_ADDR_WIDTH-1:ADDR_LSB]),
     .S_AXI_WDATA(S_AXI_WDATA),
     .S_AXI_ARESETN(S_AXI_ARESETN),
     .LED(LED)
   );
```

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

10. Save the file by right click and select **Save File**

11. Click on the Add Sources in the Flow Navigator pane, select Add or Create Design Sources, click Next, then click the Green Plus then Add Files…, browse to **{sources}\lab3**, select the lab3_user_logic.v file and click OK, and then click Finish to add the file.

    Check the contents of this file to understand the logic that is being implemented. Notice the formed hierarchy.

Make sure that when adding the source lab3_user_logic.v, untick the option: Copy sources into IP directory

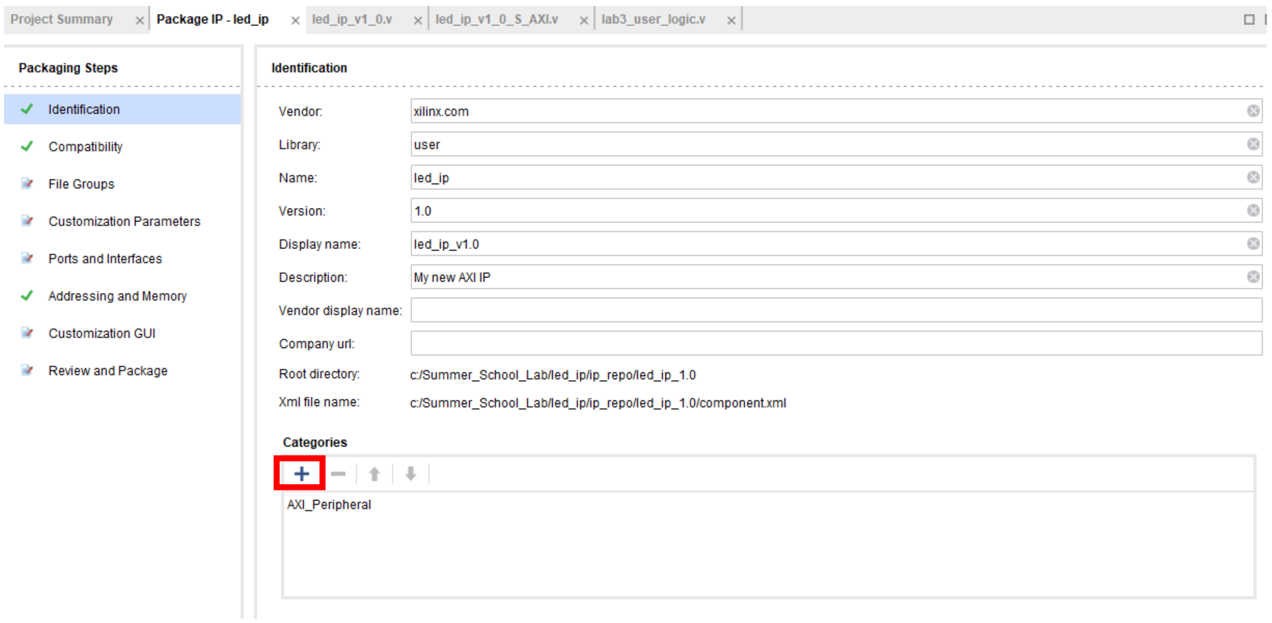
12. Click *Run Synthesis* and *Save* if prompted. (This is to check the design synthesizes correctly before packaging the IP. If this was your own design, you would simulate it and verify functionality before proceeding)

13. Check the *Messages* tab for any errors and correct if necessary before moving to the next step

When Synthesis completes successfully, click Cancel.

Package the IP

1. Click on the Package IP – led_ip tab



Identification

Vendor: xilinx.com
Library: user
Name: led_ip
Version: 1.0
Display name: led_ip_v1.0
Description: My new AXI IP
Vendor display name:
Company url:
Root directory: c:/Summer_School_Lab/led_ip/ip_repo/led_ip_1.0
Xml file name: c:/Summer_School_Lab/led_ip/ip_repo/led_ip_1.0/component.xml

Package IP

Skip the next two steps (2 and 3) if you see /Basic_Elements under the Categories section, (like the one shown in figure)

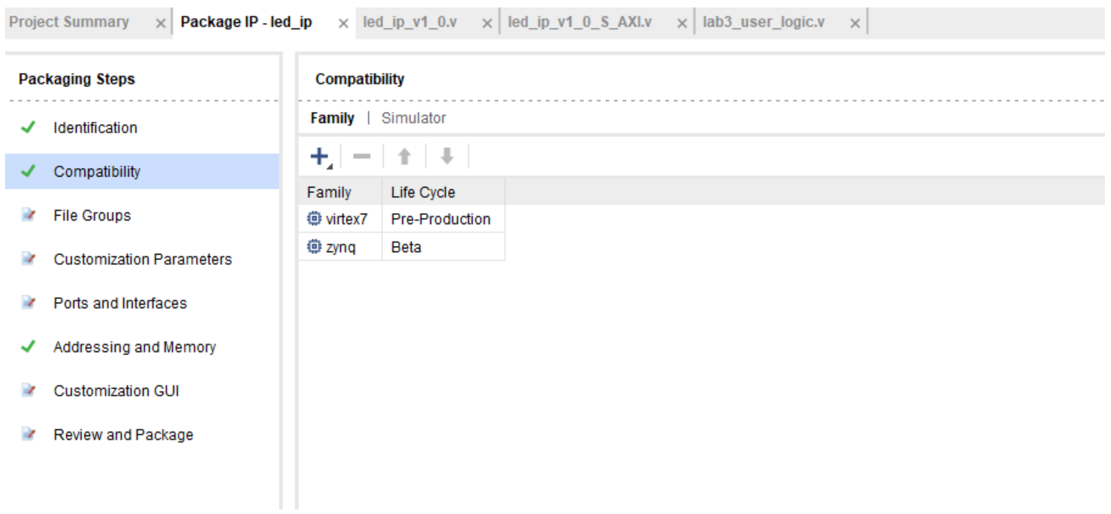
2. For the IP to appear in the IP catalog in particular categories, the IP must be configured to be part of those categories. To change which categories the IP will appear in the IP catalog click Blue Plus (Red box shown in the figure) in the Categories section. This opens the Choose IP Categories window

3. For the purpose of this exercise, uncheck the *AXI Peripheral* box and check the Basic Elements and click OK.

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

5. Click the Blue Plus then **Add Family Explicitly…** from the menu.

6. Select the Zynq family as we will be using this IP on the PYNQ-Z2, Zybo and Zedboard, and click OK. You will get something like this:

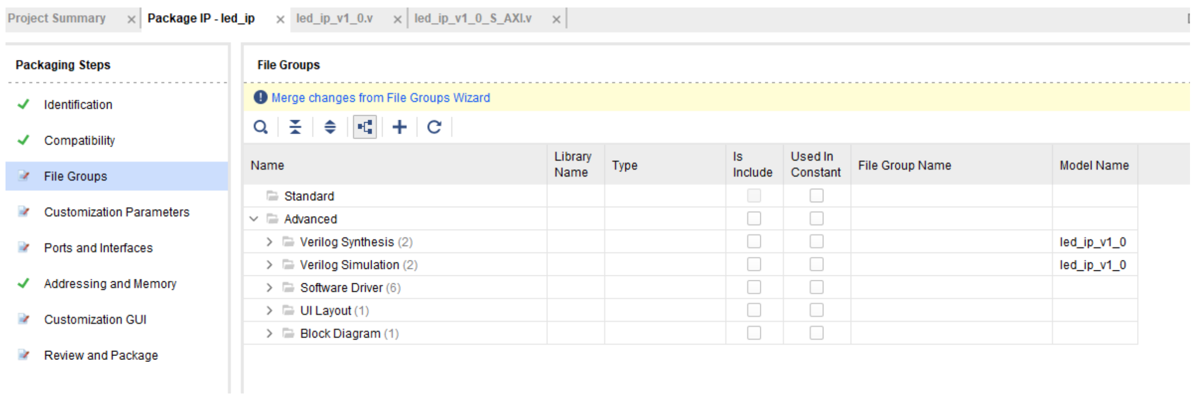


Family	Life Cycle
virtex7	Pre-Production
zynq	Beta

Compatibility under Package IP

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.

8. Click on File Groups and click Merge changes from File Groups Wizard



Name	Library Name	Type	Is Include	Used In Constant	File Group Name	Model Name
Standard			☐	☐		
Advanced			☐	☐		
Verilog Synthesis (2)			☐	☐		led_ip_v1_0
Verilog Simulation (2)			☐	☐		led_ip_v1_0
Software Driver (6)			☐	☐		
UI Layout (1)			☐	☐		
Block Diagram (1)			☐	☐		

Compatibility under Package IP

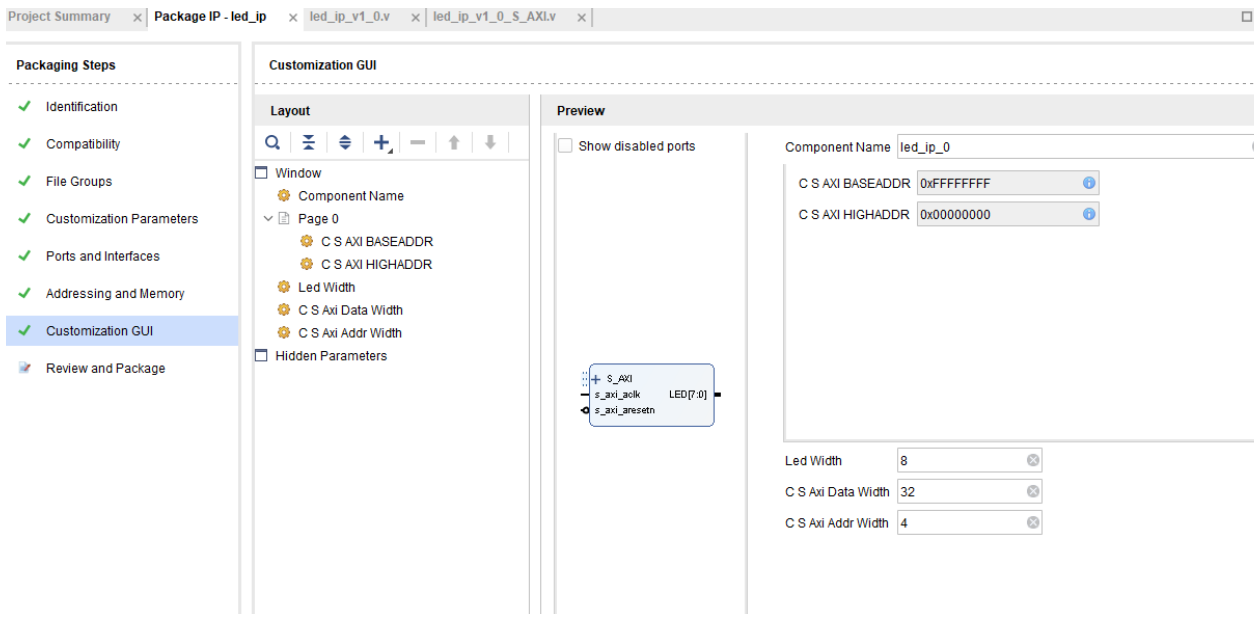
This is to update the IP Packager with the changes that were made to the IP and the lab3_user_logic_*.v file that was added to the project.

9. Expand Verilog Synthesis and notice lab3_user_logic.v has been included

10. Click on Customization Parameters and again Merge changes from Customization Parameters Wizard Notice that the Ports and Interfaces view now shows the user created LED port

11. Select **Customization Parameters**, expand **Hidden Parameters**, right-click on *LED_WIDTH*, and select **Import IP Parameters…** and click OK.

12. Select Customization GUI and notice that the Led Width is visible.



Component Name: led_ip_0
C S AXI BASEADDR: 0xFFFFFFFF
C S AXI HIGHADDR: 0x00000000
Led Width: 8
C S Axi Data Width: 32
C S Axi Addr Width: 4

Customization GUI under Package IP

13. Select Review and Package, and notice the path where the IP will be created.

14. Click Package IP. Click Yes and the project will close when complete.

15. In the original Vivado window click File > Close Project

Modify the Project Settings

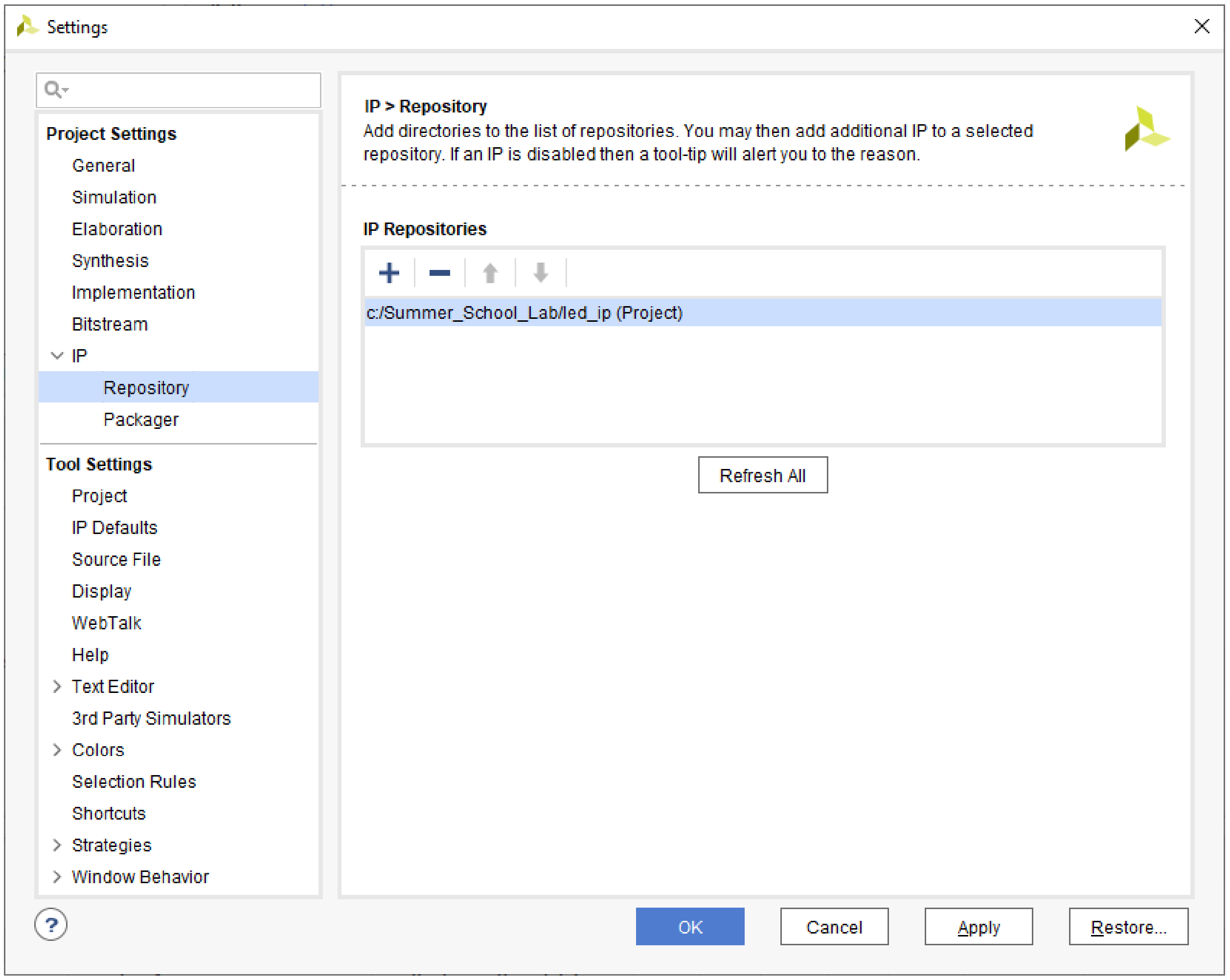
1. Start the Vivado if necessary and open either the lab2 project you created in the previous lab or the lab2 project in the labsolution directory

2. 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.

3. Click **Settings** in the Flow Navigator pane.

4. Select **IP > Repository** in the left pane of the Project Settings form.

5. Click on the Blue Plus button, browse to **{labs}\led_ip** and click Select. The led_ip_v1.0 IP will appear the IP in the Selected Repository window.



IP > Repository

Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

IP Repositories

c:/Summer_School_Lab/led_ip (Project)

Specify IP Repository

6. Click OK.

Add the Custom IP and the Constraints

1. Click Open Block Design under IP Integrator in the Flow Navigator pane
2. Right Click on the Diagram window and **add IP…**. Search for **led_ip_v1.0** in the catalog by typing **"led"** in the search field.
3. Double-click led_ip_v1_0 to add the core to the design.

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

5. Double click the block to open the configuration properties

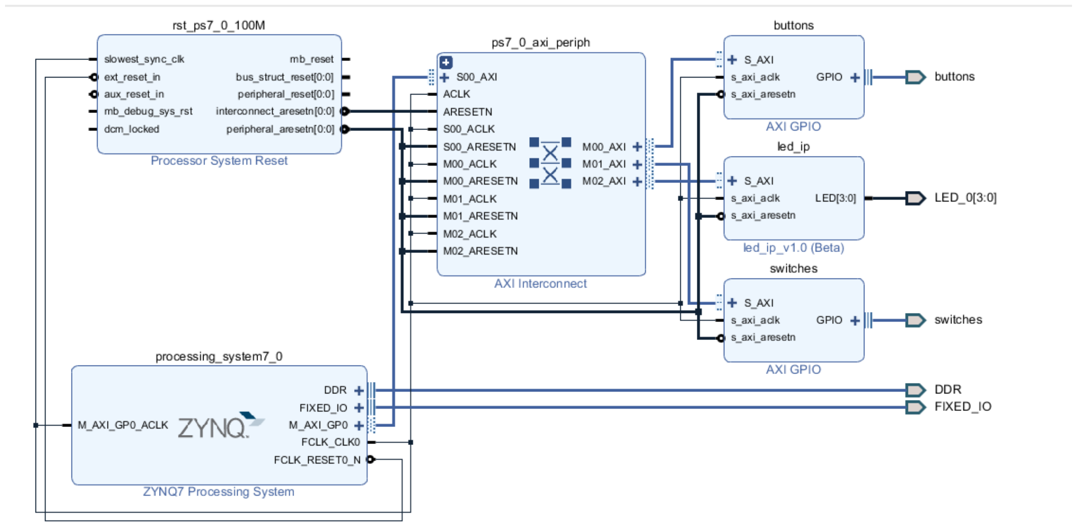
6. For the *ZedBoard*, leave the Led Width set to 8, or for the *Zybo* and *PYNQ-Z2*, set the width to 4.

7. Click OK.

8. 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.

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



LED external port added and connected

10. Select the LED_0[3:0] and change the interface name to LED in the properties view.

11. Select the Address Editor tab and verify that an address has been assigned to led_ip.

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

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

14. Click the Blue Plus button, and then **Add Files…**, browse to the **{sources}\lab3** folder, select lab3_zed.xdc for the *ZedBoard*, lab3_pynq_z2.xdc for *PYNQ-Z2* or lab3_Zybo.xdc for the *Zybo*

15. Click **Finish** to add the file.

16. 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.

17. Right click on system.bd and select Generate output products

18. 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

Export to SDK and create Application Project

1. Click **File > Export > Export Hardware**.

2. Click on the checkbox of **Include the bitstream** and then click **Yes** to overwrite.

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

4. To tidy up the workspace and save unnecessary building of a project that is not being used, right click on the *lab2*, *standalonebsp0*, and the *system_wrapper_hw_platform_1* projects from the previous lab, and click Close Project, as these projects will not be used in this lab. They can be reopened later if needed.

5. Select **File > New > Application Project**.

6. Enter lab3 as the Project Name, and for Board Support Package, choose Create New **lab3_bsp** (should be the only option).

7. Click Next, and select **Empty Application** and click Finish.

8. Expand lab3 in the project view and right-click in the src folder and select Import.

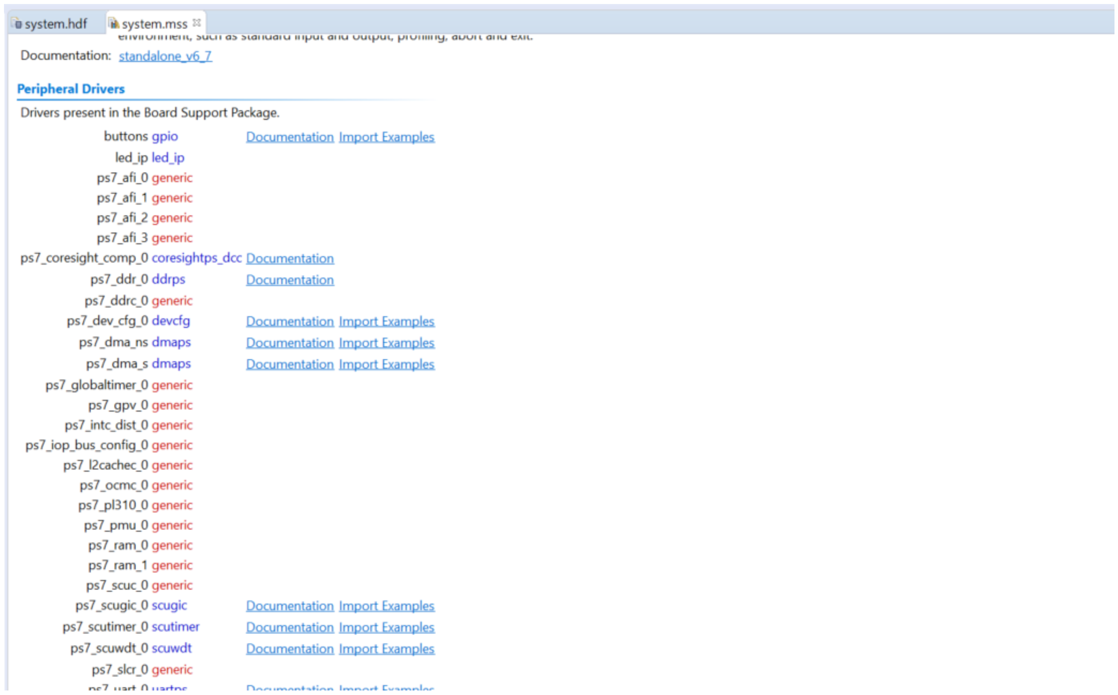
9. Expand General category and double-click on File System.

10. Browse to **{sources}\lab3** folder and click OK.

11. Select lab3.c and click Finish to add the file to the project. (Ignore any errors for now).

12. Expand lab3_bsp and open the system.mss

13. Click on **Documentation** link corresponding to buttons peripheral under the Peripheral Drivers section to open the documentation in a default browser window. As our led_ip is very similar to GPIO, we look at the mentioned documentation.



Documentation: standalone_v6_7

Peripheral Drivers

Drivers present in the Board Support Package.

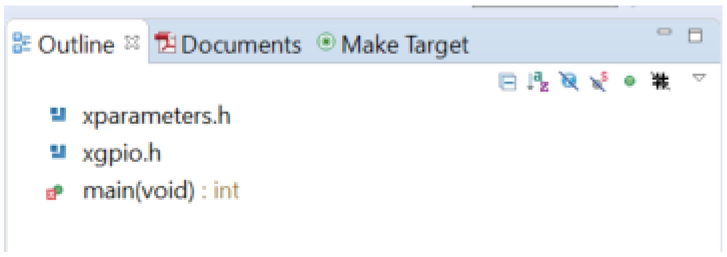
buttons	gpio	Documentation Import Examples
led_ip	led_ip	
ps7_afi_0	generic	
ps7_afi_1	generic	
ps7_afi_2	generic	
ps7_afi_3	generic	
ps7_coresight_comp_0	coresightps_dcc	Documentation
ps7_ddr_0	ddrps	Documentation
ps7_ddrc_0	generic	
ps7_dev_cfg_0	devcfg	Documentation Import Examples
ps7_dma_ns	dmaps	Documentation Import Examples
ps7_dma_s	dmaps	Documentation Import Examples
ps7_globaltimer_0	generic	
ps7_gpv_0	generic	
ps7_intc_dist_0	generic	
ps7_iop_bus_config_0	generic	
ps7_l2cachec_0	generic	
ps7_ocmc_0	generic	
ps7_pl310_0	generic	
ps7_pmu_0	generic	
ps7_ram_0	generic	
ps7_ram_1	generic	
ps7_scuc_0	generic	
ps7_scugic_0	scugic	Documentation Import Examples
ps7_scutimer_0	scutimer	Documentation Import Examples
ps7_scuwdt_0	scuwdt	Documentation Import Examples
ps7_slcr_0	generic	
ps7_uart_0	uartps	Documentation Import Examples

Accessing device driver documentation

14. View the various C and Header files associated with the *GPIO* by clicking Files List at the top of the page.

15. Double-click on lab3.c in the Project Explorer view to open the file. This will populate the *Outline tab*.

16. Double click on *xgpio.h* in the Outline view on the right of the screen and review the contents of the file to see the available function calls for the GPIO.



Outline View

The following steps must be performed in your software application to enable reading from the GPIO: 1) Initialize the GPIO, 2) Set data direction, and 3) Read the data

Find the descriptions for the following functions:

XGpio_Initialize (XGpio *InstancePtr, u16 DeviceId) *InstancePtr* is a pointer to an XGpio instance. The memory the pointer references must be pre-allocated by the caller. Further calls to manipulate the component through the XGpio API must be made with this pointer.

DeviceId is the unique id of the device controlled by this XGpio component. Passing in a device id associates the generic XGpio instance to a specific device, as chosen by the caller or application developer.

XGpio_SetDataDirection (XGpio *InstancePtr, unsigned Channel, u32 DirectionMask)

InstancePtr is a pointer to the XGpio instance to be worked on.

Channel contains the channel of the GPIO (1 or 2) to operate on.

DirectionMask is a bitmask specifying which bits are inputs and which are outputs. Bits set to 0 are output and bits set to 1 are input.

XGpio_DiscreteRead(XGpio *InstancePtr, unsigned channel)

InstancePtr is a pointer to the XGpio instance to be worked on.

Channel contains the channel of the GPIO (1 or 2) to operate on

17. Open the header file **xparameters.h** by double-clicking on **xparameters.h** in the Outline tab

The xparameters.h file contains the address map for peripherals in the system. This file is generated from the hardware platform description from Vivado. Find the following #define used to identify the switches peripheral:

```
#define XPAR_SWITCHES_DEVICE_ID 1
```

Note the number might be different

Notice the other #define XPAR_SWITCHES* statements in this section for the switches peripheral, and in particular the address of the peripheral defined by: XPAR_SWITCHES_BASEADDR

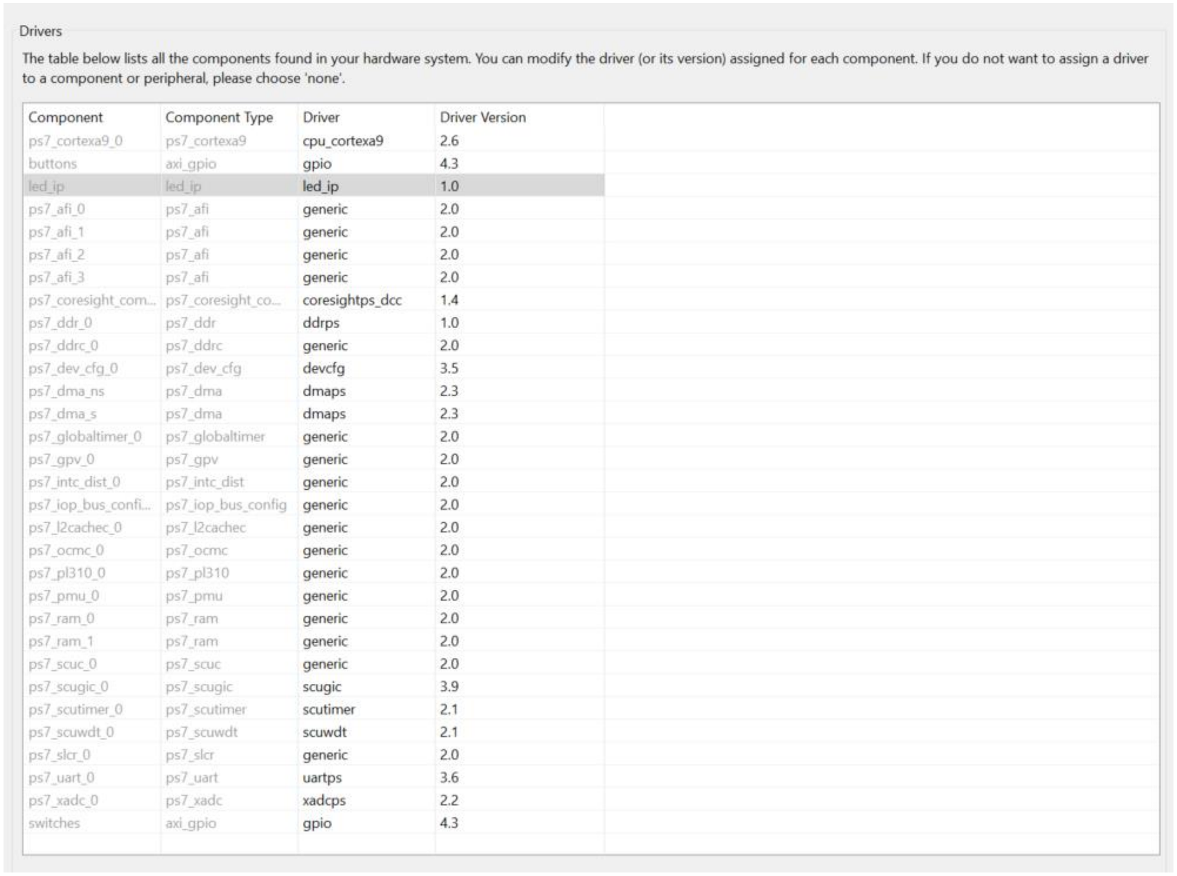
15. Modify line 14 of lab4.c to use this macro (#define) in the XGpio_Initialize function.

```
1 #include "xparameters.h"
2 #include "xgpio.h"
3
4 //====================================================
5
6 int main (void)
7 {
8
9    XGpio dip, push;
10   int i, psb_check, dip_check;
11
12   xil_printf("-- Start of the Program --\r\n");
13
14   XGpio_Initialize(&dip, XPAR_DIP_DEVICE_ID); // Modify this
15   XGpio_SetDataDirection(&dip, 1, 0xffffffff);
16
17   XGpio_Initialize(&push, XPAR_PUSH_DEVICE_ID); // Modify this
18   XGpio_SetDataDirection(&push, 1, 0xffffffff);
19
20
21   while (1)
22   {
23        psb_check = XGpio_DiscreteRead(&push, 1);
24        xil_printf("Push Buttons Status %x\r\n", psb_check);
25        dip_check = XGpio_DiscreteRead(&dip, 1);
26        xil_printf("DIP Switch Status %x\r\n", dip_check);
27
28        // output dip switches value on LED_ip device
29
30        for (i=0; i<9999999; i++);
31   }
32 }
```

16. Do the same for the **BUTTONS**; find the macro (#define) for the **BUTTONS** peripheral in **xparameters.h**, and modify line 17 in lab3.c, and save the file.

The project will be rebuilt. If there are any errors, check and fix your code. Your C code will eventually read the value of the switches and output it to the led_ip.

17. Select **lab3_bsp** in the project view, right-click, and select Board Support Package Settings.
18. Select drivers on the left (under Overview)
19. If the led_ip driver has not already been selected, select Generic under the Driver column for led_ip to access the dropdown menu. From the dropdown menu, select led_ip, and click OK.


Drivers

The table below lists all the components found in your hardware system. You can modify the driver (or its version) assigned for each component. If you do not want to assign a driver to a component or peripheral, please choose 'none'.

Component	Component Type	Driver	Driver Version
ps7_cortexa9_0	ps7_cortexa9	cpu_cortexa9	2.6
buttons	axi_gpio	gpio	4.3
led_ip	led_ip	led_ip	1.0
ps7_afi_0	ps7_afi	generic	2.0
ps7_afi_1	ps7_afi	generic	2.0
ps7_afi_2	ps7_afi	generic	2.0
ps7_afi_3	ps7_afi	generic	2.0
ps7_coresight_com...	ps7_coresight_co...	coresightps_dcc	1.4
ps7_ddr_0	ps7_ddr	ddrps	1.0
ps7_ddrc_0	ps7_ddrc	generic	2.0
ps7_dev_cfg_0	ps7_dev_cfg	devcfg	3.5
ps7_dma_ns	ps7_dma	dmaps	2.3
ps7_dma_s	ps7_dma	dmaps	2.3
ps7_globaltimer_0	ps7_globaltimer	generic	2.0
ps7_gpv_0	ps7_gpv	generic	2.0
ps7_intc_dist_0	ps7_intc_dist	generic	2.0
ps7_iop_bus_confi...	ps7_iop_bus_config	generic	2.0
ps7_l2cachec_0	ps7_l2cachec	generic	2.0
ps7_ocmc_0	ps7_ocmc	generic	2.0
ps7_pl310_0	ps7_pl310	generic	2.0
ps7_pmu_0	ps7_pmu	generic	2.0
ps7_ram_0	ps7_ram	generic	2.0
ps7_ram_1	ps7_ram	generic	2.0
ps7_scuc_0	ps7_scuc	generic	2.0
ps7_scugic_0	ps7_scugic	scugic	3.9
ps7_scutimer_0	ps7_scutimer	scutimer	2.1
ps7_scuwdt_0	ps7_scuwdt	scuwdt	2.1
ps7_slcr_0	ps7_slcr	generic	2.0
ps7_uart_0	ps7_uart	uartps	3.6
ps7_xadc_0	ps7_xadc	xadcps	2.2
switches	axi_gpio	gpio	4.3

Assign led_ip driver

Examine the Driver code

The driver code was generated automatically when the IP template was created. The driver includes higher level functions which can be called from the user application. The driver will implement the low level functionality used to control your peripheral.

1. In windows explorer, browse to **led_ip\ip_repo\led_ip_1.0\drivers\led_ip_v1_0\src** Notice the files in this directory and open **led_ip.c**. This file only includes the header file for the IP.
2. Close led_ip.c and open the header file **led_ip.h** and notice the macros:

```
LED_IP_mWriteReg( … )
LED_IP_mReadReg( … )
e.g: search for the macro name LED_IP_mWriteReg:
/**
 *
 * Write a value to a LED_IP register. A 32 bit write is performed.
 * If the component is implemented in a smaller width, only the least
 * significant data is written.
 *
 * @param   BaseAddress is the base address of the LED_IP device.
 * @param   RegOffset is the register offset from the base to write to.
 * @param   Data is the data written to the register.
 *
 * @return  None.
 *
 * @note
 * C-style signature:
 *      void LED_IP_mWriteReg(Xuint32 BaseAddress, unsigned RegOffset,    Xuint32 
Data)
  *
 */
#define LED_IP_mWriteReg(BaseAddress, RegOffset, Data) \
        Xil_Out32((BaseAddress) + (RegOffset), (Xuint32)(Data))
```

For this driver, you can see the macros are aliases to the lower level functions *Xil_Out32( )* and *Xil_Out32( )*. The macros in this file make up the higher level API of the led_ip driver. If you are writing your own driver for your own IP, you will need to use low level functions like these to read and write from your IP as required. The low level hardware access functions are wrapped in your driver making it easier to use your IP in an Application project.

3. Include the header file:

```
#include "led_ip.h"
```

4. Include the function to write to the IP (insert before the for loop):

```
LED_IP_mWriteReg(XPAR_LED_IP_S_AXI_BASEADDR, 0, dip_check);
```

Remember that the hardware address for a peripheral (e.g. the macro XAR_LED_IP_S_AXI_BASEADDR in the line above) can be found in xparameters.h

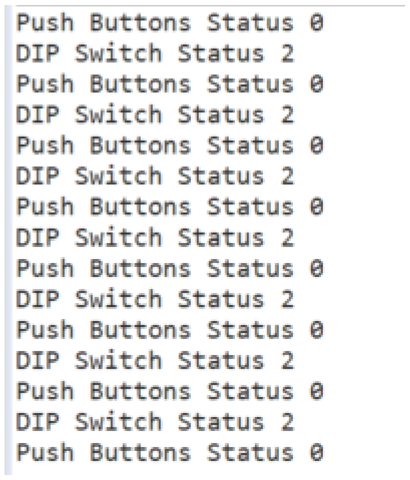
```
#include "xparameters.h"
#include "xgpio.h"
#include "led_ip.h"
//====================================================
int main (void)
{

   XGpio dip, push;
   int i, psb_check, dip_check;

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

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

   XGpio_Initialize(&push, XPAR_BUTTONS_DEVICE_ID); // Modify this
   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
          LED_IP_mWriteReg(XPAR_LED_IP_S_AXI_BASEADDR, 0, dip_check);

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

5. Save the file and the program will be compiled again.

Verify in Hardware

1. Make sure that micro-USB cable(s) is(are) connected between the board and the PC. Turn ON the power.
2. Select the tab. If it is not visible then select **Window > Show view > Other.. > Terminal**.
3. Click on the connect button 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).
4. Select **Xilinx Tools > Program FPGA**.
5. Click the Program button to program the FPGA.

6. Select lab3 in Project Explorer, right-click and select **Run As > Launch on Hardware (System Debugger)** to download the application, execute ps7_init, and execute lab3.elf



```
Push Buttons Status 0
DIP Switch Status 2
Push Buttons Status 0
DIP Switch Status 2
Push Buttons Status 0
DIP Switch Status 2
Push Buttons Status 0
DIP Switch Status 2
Push Buttons Status 0
DIP Switch Status 2
Push Buttons Status 0
DIP Switch Status 2
Push Buttons Status 0
DIP Switch Status 2
Push Buttons Status 0
```

DIP switch and Push button settings displayed in SDK terminal

Note: Setting the DIP switches and push buttons will change the results displayed.

Meanwhile, LEDs on board will also result the status of DIP switches.

7. When finished, click on the Terminate button in the Console tab.
8. Exit SDK and Vivado.
9. Power **OFF** the board.

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.

Use SDK to define, develop, and integrate the software components of the embedded system. You can define a device driver interface for each of the peripherals and the processor. SDK imports an hdf file, creates a corresponding MSS file and lets you update the settings so you can develop the software side of the processor system. You can then develop and compile peripheral-specific functional software and generate the executable file from the compiled object code and libraries.