

Lab 1 - Zynq RTL Design Flow

Pin-Hung Kuo

May 11, 2018

1 INTRODUCTION

In this lab, we are going to build a simple Zynq system on ZedBoard. This system works as following: the CPU sends an 8-bits integer value to our IP, then this IP shows the value by 8 PL LEDs on the board. During the lab, you will learn how to build a Zynq block design, how the CPU of Zynq to communicate with peripherals, and how to build an IP with RTL flow. The following steps will lead you to build such a system.

2 CREATING IP IN RTL

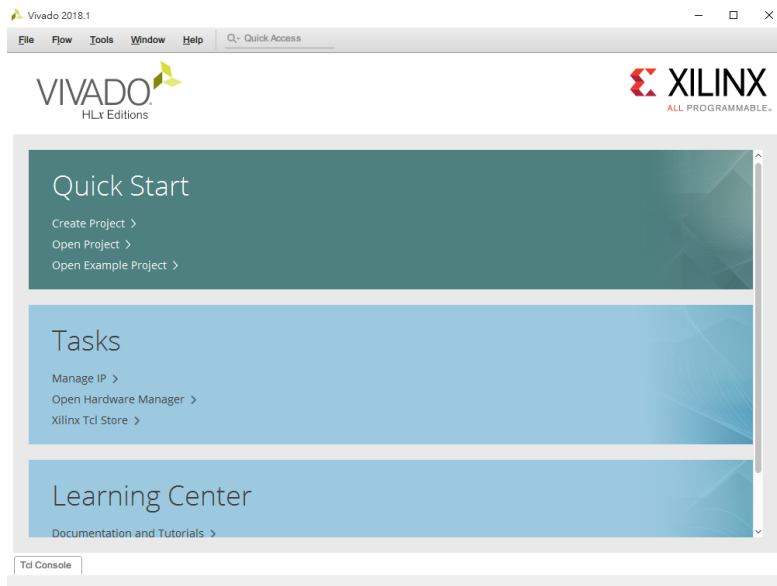


Figure 2.1

Open Vivado, select **Create New Project** as Fig.2.1. Click **Next** in *New Project* dialogue. At the Project Name dialogue, enter **Lab1** as the **Project name** and **C:/MSOC**(the directory where you unzip the lab file) as **Project location**. Make sure that the **Create project subdirectory** is ticked. Please refer to Fig.2.2 for the above step. Click **Next**. Before next step, please copy the **source** folder in the lab file to **C:/MSOC/**.

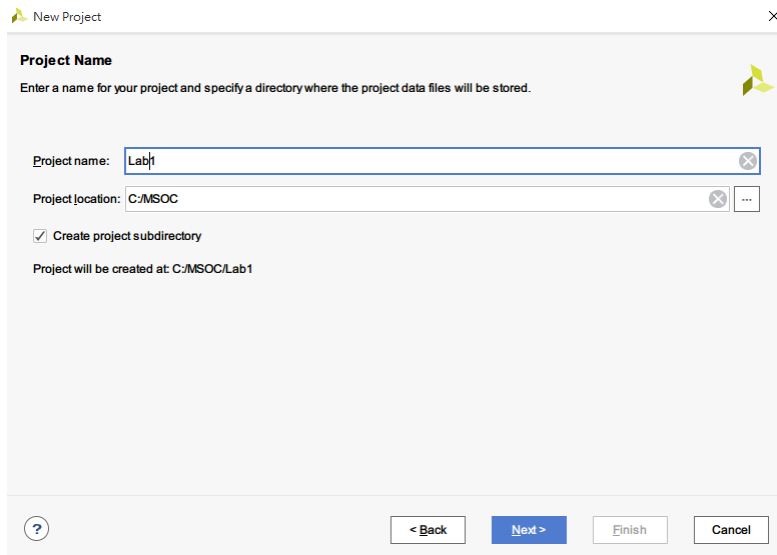


Figure 2.2

In **Project Type**, select **RTL Project** and do not specify sources as shown in Fig.2.3.

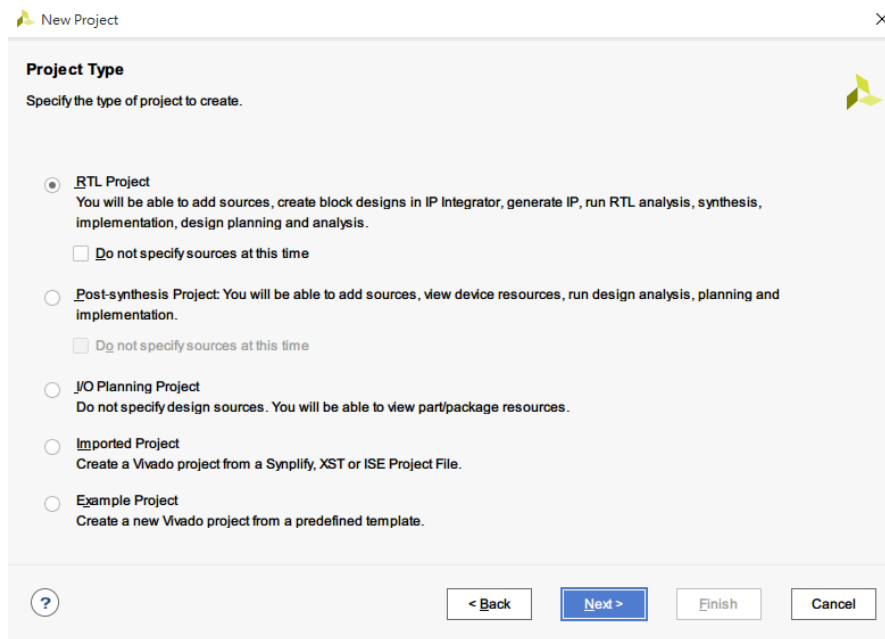


Figure 2.3

Select **Verilog** as **Target language** in the *Add Sources* dialogue as in Fig.2.4. In *Add Existing IP(optional)* dialogue, we can add existing IP here. As we do not have that, click **Next**.

The *Add Constraints(optional)* dialogue will open. This is the stage were any physical or timing constraints files could be added to the project. As we do not have that, click **Next**.

The *Default Part* dialogue will open. Select **Boards** from the *Select dialogue* and **ZedBoard Zynq Evaluation and Development Kit** as Fig.2.5.

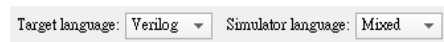


Figure 2.4

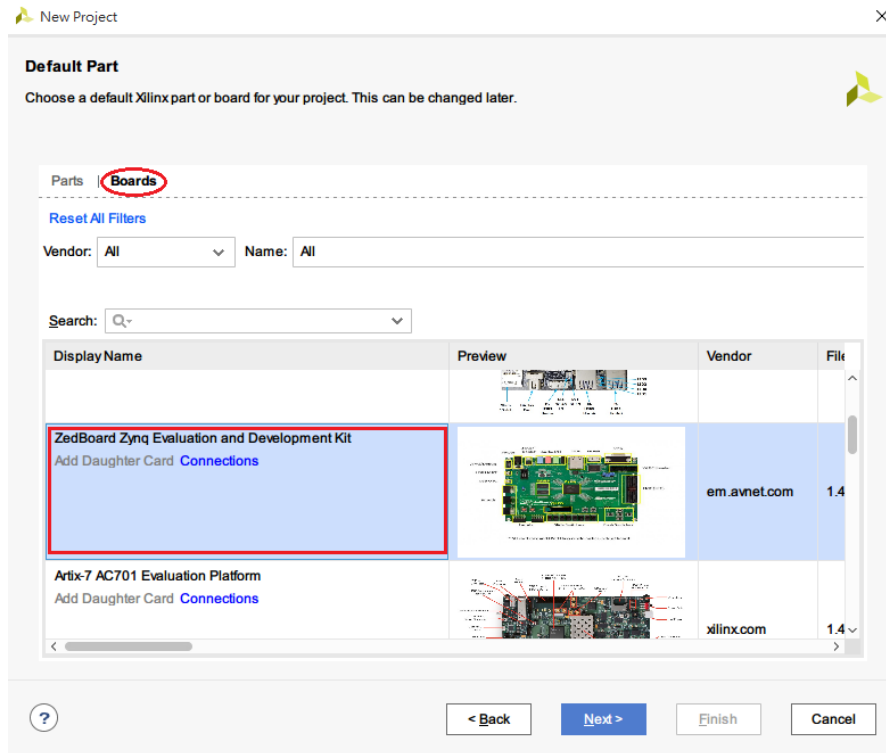


Figure 2.5

Click **Finish** to create the project.

From the menu bard, select **Tools>Create and Package New IP...**, as shown in Fig.2.6

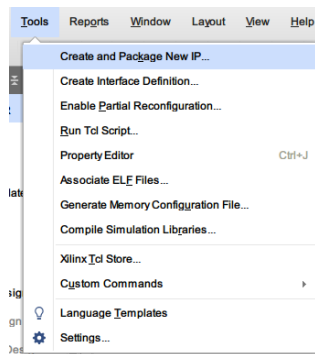


Figure 2.6

The *Create and Package New IP* dialogue will launch, Click **Next**.

Then, select **Create new AXI4 peripheral** here and click **Next 2.7**.

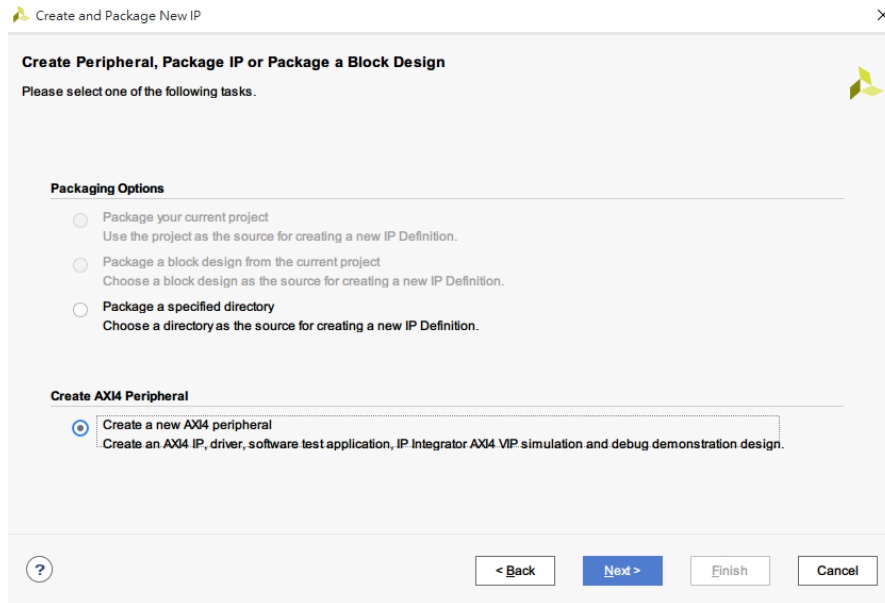


Figure 2.7

Enter **led_controller** as the **Name** 2.8.

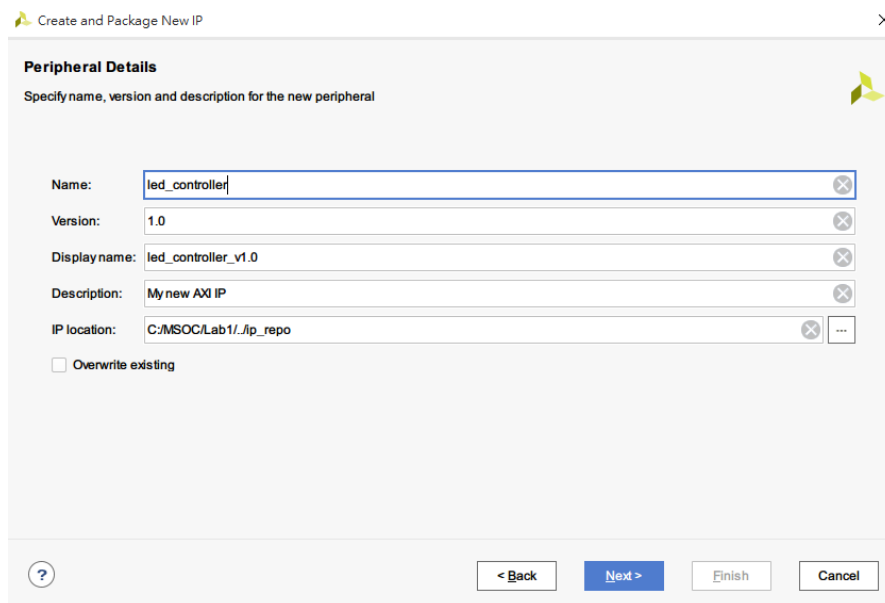


Figure 2.8

Click **Next**.

The *Add Interface* dialogue allows you to specify the AXI4 interface(s) that will be present in your custom peripheral. Here you can specify:

- Number of interfaces
- Interface type (AXI-Lite, AXI-Stream or AXI-Full)
- Interface mode(slave or master)
- Interface data width

Features specific to individual interface types will also be available when the corresponding type is selected.

As our peripheral is a simple controller for the LEDs, which only requires single values to be transferred to it, an AXI-Lite slave interface is sufficient. Only one memory mapped register is required for our simple controller, but as the minimum number that can be specified in the dialogue is 4, we will chose that. Specify the *Add Interface* dialogue as shown in Fig.2.9.

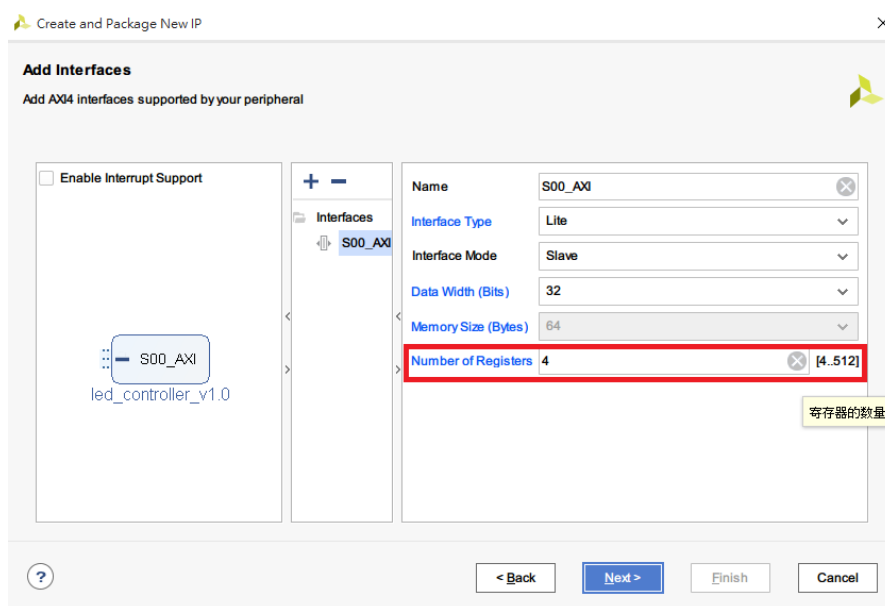


Figure 2.9

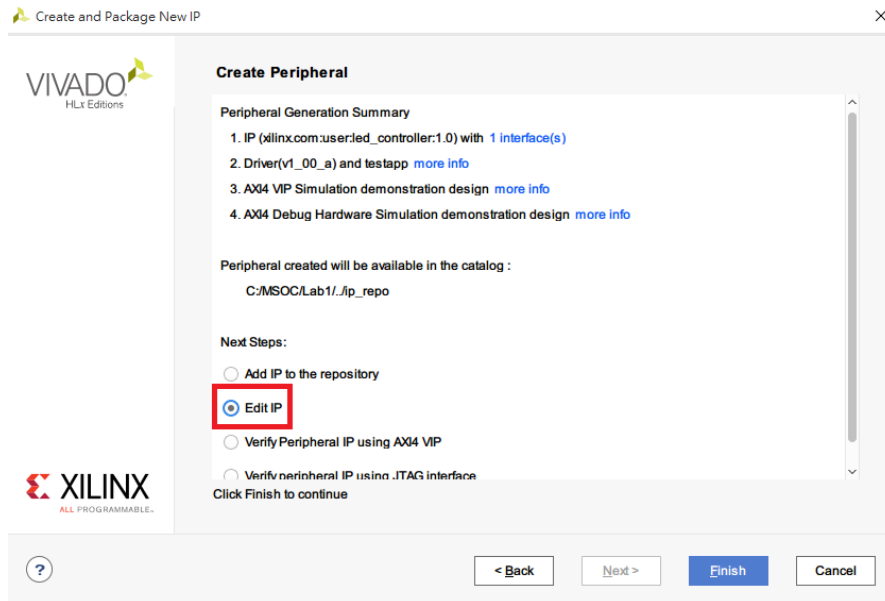


Figure 2.10

Review the information in the *Create Peripheral* dialogue, which details the output files which will be created. Select the option to **Edit IP 2.10**. This will create the IP peripheral files and create a new Vivado project where the functionality of the peripheral can be modified in the source HDL code, and the packaged. Click **Finish** to close the *Wizard* and create the peripheral template.

A new Vivado project, named **edit_led_controller_v1_0**, will open.

In the *Sources* pane, you should see two HDL source files (you may need to expand the file selection):

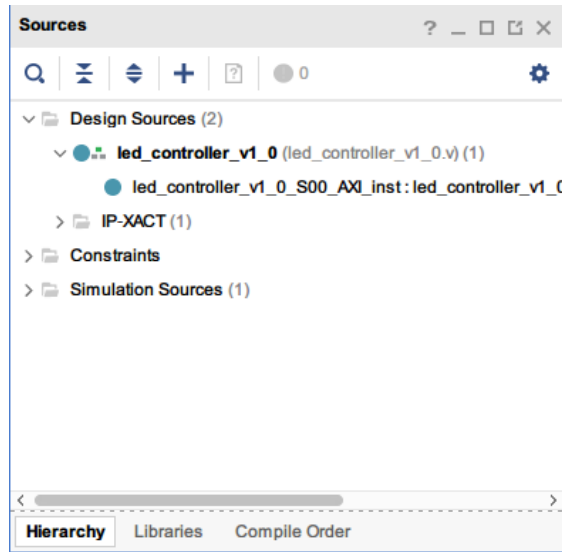


Figure 2.11

The two source files are:

- **led_controller_v1_0.v** - This file instantiates all AXI-Lite interfaces. In this case, only one interface is present.
- **led_controller_v1_0_S00_AXI.v** - This file contains the AXI4-Lite interface functionality which handles the interactions between the peripheral in the PL and the software running on the PS.

The *IP Packer* pane will also be open in the *Workspace*:

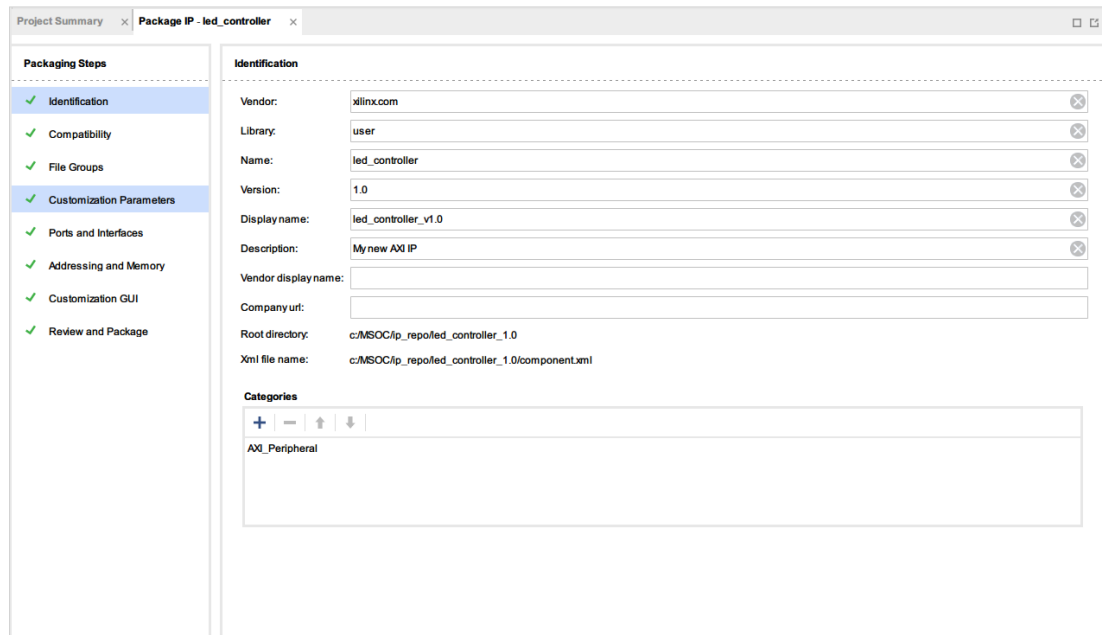


Figure 2.12

The information that we specified about our peripheral in previous steps will be visible.

We can now add the functionality to our **led_controller** peripheral. We will be adding a new output port the peripheral template to allow it to connect to the LED pins on the Zynq device, as well as assigning the value received from the Zynq PS to the new output port.

Open **led_controller_v1_0_S00_AXI.v** by double-click on it in the *Sources* pane. The file will be opened in the Workspace.

Add the declaration

```
output wire [7:0] LEDs_out,
```

as shown in Fig.2.13.

```

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 [7:0] LEDs_out,
19 | // User ports ends
20 | // Do not modify the ports beyond this line
21 |
22 | // Global Clock Signal

```

Figure 2.13

And connect this output port to register `slv_reg0`

```
assign LEDs_out = slv_reg0[7:0];
```

as Fig.2.14.

```
395 |         axi_rdata <= reg_data_out;    //register read data
396 |     end
397 | end
398 | end
399 |
400 | // Add user logic here
401 | assign LEDs_out = slv_reg0[7:0];
402 | // User logic ends
403 |
404 | endmodule
405 |
```

Figure 2.14

Save the file by **File>Save File** or **Ctrl+s**.

Open **led_controller_v1_0.v**. We must once again create a new output port to the top-level source file, and map it to the equivalent port that we created in the AXI4-Lite interface file in the previous step.

As in **led_controller_v1_0_S00_AXI.v**, we add the declaration of `LEDs_out` again

```
output wire [7:0] LEDs_out,
```

as shown in Fig.2.15.

```
15 | )
16 | (
17 | // Users to add ports here
18 | output wire [7:0] LEDs_out,
19 | // User ports ends
20 | // Do not modify the ports beyond this line
21 |
22 |
23 | // Ports of Axi Slave Bus Interface S00_AXI
```

Figure 2.15


and connect to the port by




```
.LEDs_out(LEDs_out)
```

in the instance of **led_controller_v1_0.v**. Fig.2.16 shows this step.
NOTE: You should add a "," in the end of the port before LEDs_out.

```
64 |         .S_AXI_ARADDR(s00_axi_araddr),
65 |         .S_AXI_ARPROT(s00_axi_arprot),
66 |         .S_AXI_ARVALID(s00_axi_arvalid),
67 |         .S_AXI_ARREADY(s00_axi_arready),
68 |         .S_AXI_RDATA(s00_axi_rdata),
69 |         .S_AXI_RRESP(s00_axi_rresp),
70 |         .S_AXI_RVALID(s00_axi_rvalid),
71 |         .S_AXI_RREADY(s00_axi_rready),
72 |         .LEDs_out(LEDs_out)
73 |     );
74 |
75 |     // Add user logic here
76 |
```


Figure 2.16


Return to **IP Packager** by selecting the **Package IP - led_controller**. IP Packager will detect the changes to the source files, and the areas which need refreshed will be highlighted with the following icon: . You should see that the following three areas of interest need

-  **File Groups**
-  **Customization Parameters**
-  **Ports and Interfaces**

refreshed :

Select **Customization Parameters** in the *Packager* pane. You should see the following information message as the top of the pane:

 [Merge changes from Customization Parameters Wizard](#)

Click **Merge changes from Customization Parameters Wizard** to update the IP Packager information to the changes made in the HDL source files. Perform the similar process to **File Groups** to eliminate all  icon except for **Review and Package**.

To verify that IP Packager has updated the Ports and Interfaces area, we will open it and check.

Select **Ports and Interfaces** from the *IP Packager* pane.

You should notice that the LEDs_out port that we added to the source files has been added to the IP Ports pane and has a length of 8:

The final step in creating our new IP peripheral is to package the IP. Select **Review and Package** from the **IP Packager** pane.

In the *After Packaging* panel, click **edit packaging settings** at the bottom:

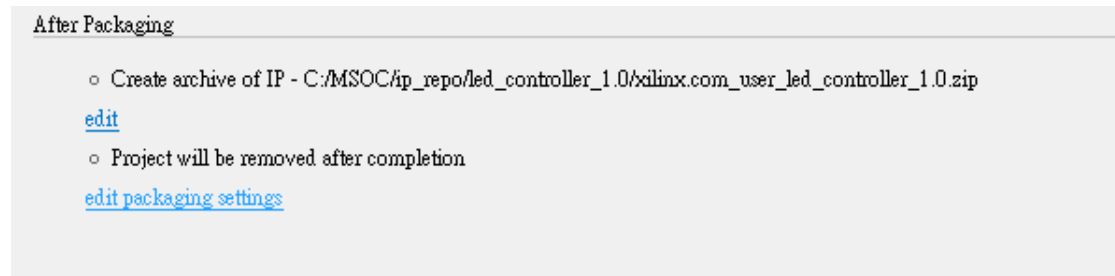


Figure 2.17

In the *Automatic Behaviour* panel, enable the option to **Create archive of IP, Close IP Packager window** and **Add IP to the IP Catalog of the Current Project**. You may **Delete project after packaging** if you wish (but the Verilog files can still be found even you select this option)

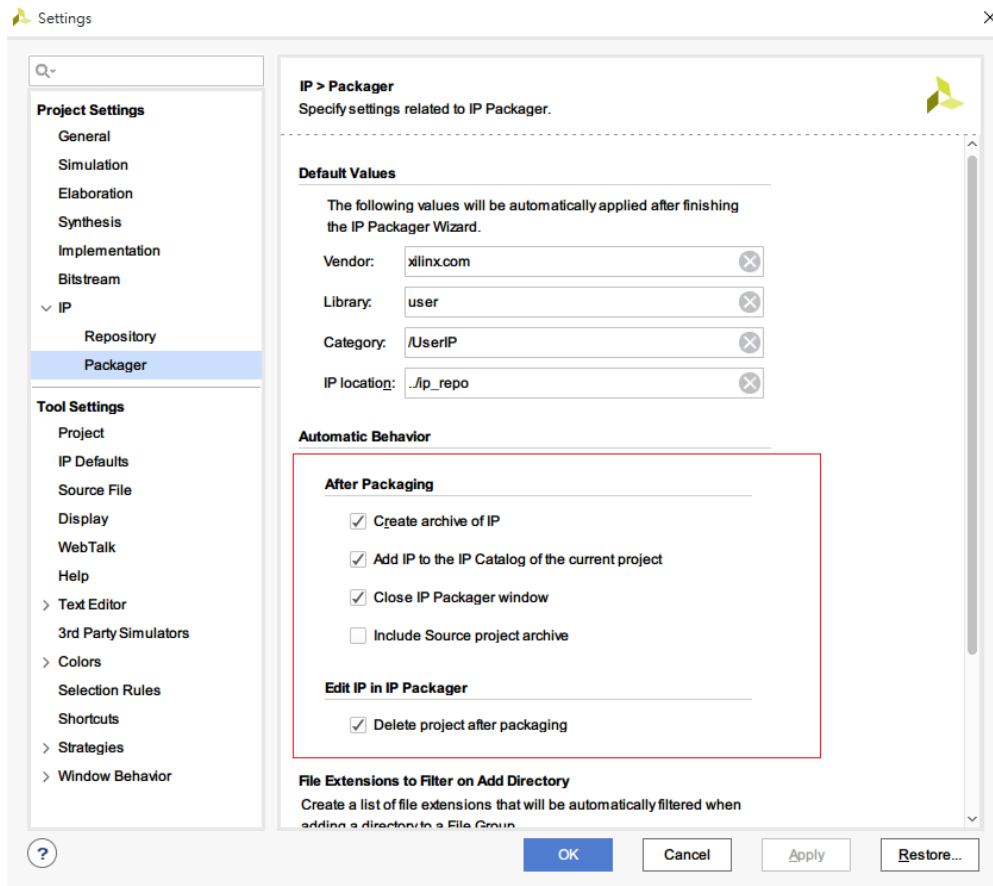


Figure 2.18

Click **OK** to apply the setting. Review the information provided in the *Review and Package* window, and click **Re-Package IP**. A dialogue box will appear asking if you want to close the project, click **Yes**. The changes made to the IP peripheral will be included in the repackaged IP, and the Vivado project will close.

3 CREATE SYSTEM BLOCK

We will return to our original Vivado project **Lab1** after we finish the IP creation.

To start, we will create a new Block Design and add the IP peripheral which we just created to the design. In the *Flow Navigator* window, select **Create Block Design** from the *IP Integrator* section. Enter **lab_1** in the *Design name* box, and click **OK** to create the blank design. Now, we want to add IP in the *Vivado IP Integrator Diagram canvas*. This can be achieved by 3 methods:

- right click anywhere and select **Add IP** as Fig.3.1

- Hot key **Ctrl+i**
- The tool bar at the left of the canvas as Fig.3.2

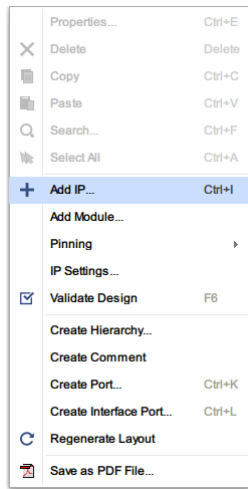


Figure 3.1: Right click list



Figure 3.2: Tool bar

Enter **led** in the *Search* box, and double-click **led_controller_v1.0** to add an instance of the LED controller IP to the design. Fig.3.3 shows the instance of our IP. To enable the peripheral to connect to the LEDs on the ZedBoard, we must make the LEDs_out port external. This allows the output port to be connected to specific physical pins on the Zynq device, which are connected to the LEDs. Right click on the port **LEDs_out[7:0]** and select **Make External**.

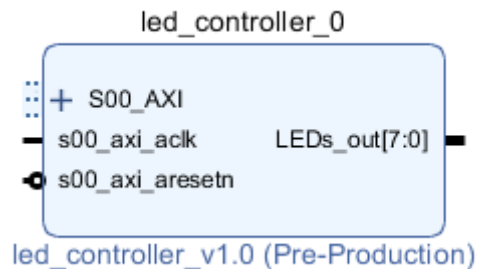


Figure 3.3: led_controller block

The block design should now resemble Fig.3.4.

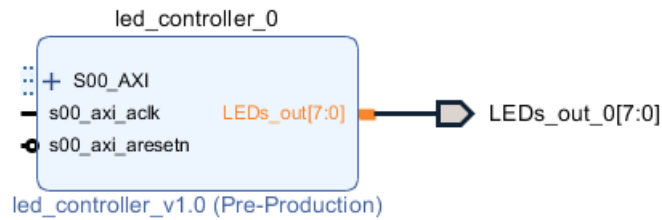


Figure 3.4: led_controller block with external port

The next step is to add a Zynq Processing System block, which stands for the PS part on the ZedBoard. Add an instance of **Zynq7 Processing System** by the similar way as we add the IP led_controller_v1.0. Type **zynq** in the search box to add the **Zynq7 Processing System**. The *Designer Assistance* message at the top of the canvas will appear:



Click **Run Block Automation**. An information message will appear. Ensure that **Apply Board Preset** is selected, and click **OK**. This will make all necessary modifications to the Zynq processing system that relate to the board preset and make required external connections.

We must now connect the LED Controller to the Zynq Processing System. This step can also be carried out using Designer Assistance.

In the *Designer Assistance* message, click **Run Connection Automation**. An information message will appear, select **led_controller_0/S00_AXI** and click **OK**. This will add some additional blocks to the design which are require to connect the LED Controller to the Zynq Processing System.

Our block design is now complete. Validate the design by selectin **Tools>Validate Design** from the *Menu Bar*, or select **Validate Design** in right-click menu, or just press keyboard **F6**. If we pass the validation, our block system is done.

Now we should generate the HDL files for the design.

In the *Sources* pane, find **lab_1(lab_1.bd)** under **Design Sources(1)**. Right-click on it and select **Create HDL Wrapper**. Select **Let Vivado manage wrapper and auto-update** as Fig.3.5 and click **OK**. This will create the top-level HLD file for the design.

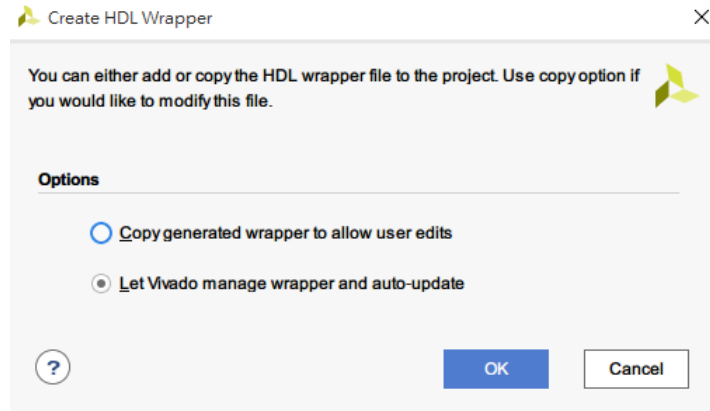


Figure 3.5: Create HDL Wrapper

We must now connect the LEDs_out port of the design to the correct pins on the Zynq device. This is done through the specification of constraints in an XDC file.

In the *Flow Navigator* window, select **Add Sources** from the *Project Manager* section. The *Add Sources* dialogue will open. Select **Add or Create Constraints**, and click **Next**. Click the

+ symbol and the click **Create File ...** as shown in Fig.3.6.

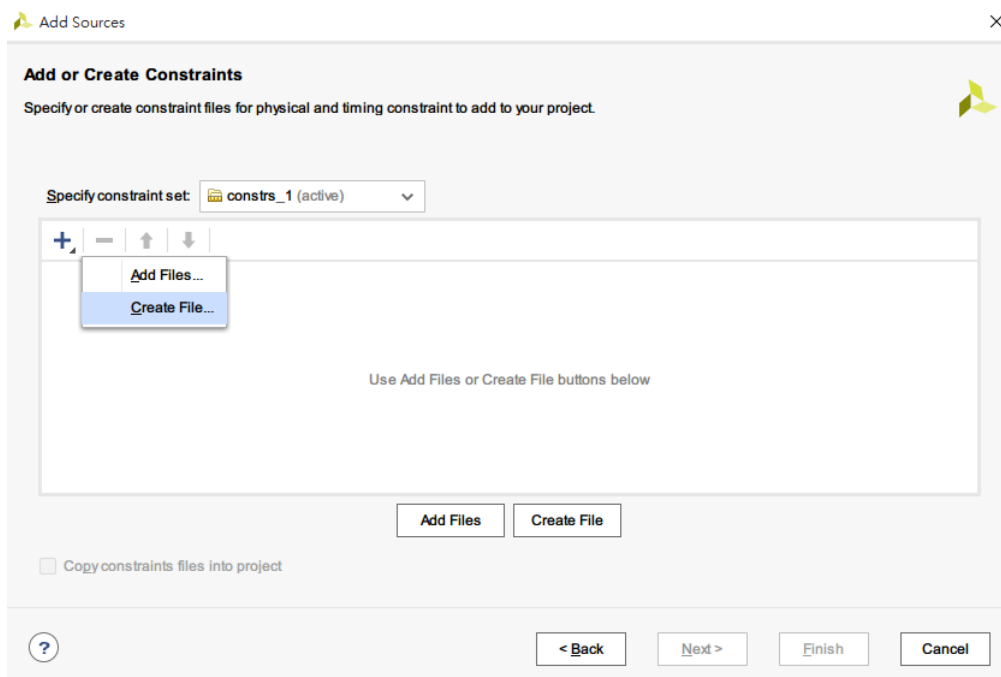


Figure 3.6: Add or Create Constraints Dialogue Window

The *Create Constraints File* dialogue will open. Select **XDC** as the *File type* and enter **led_constraints** as the *File name*. Click **OK**.

Click **Finish** to create the file and close the dialogue.

In the **Sources** tab, expand the **Constraints** entry and open the newly created XDC file by double-clicking on **led_constraints.xdc**. The file will open in the *Workspace*. Add the following constraints into the file. You can find the same constraints code in **c:/MSOC/source/Lab1/led_constraints.xdc**.

```
set_property PACKAGE_PIN T22 [get_ports {LEDs_out_0[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[0]}]
set_property PACKAGE_PIN T21 [get_ports {LEDs_out_0[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[1]}]
set_property PACKAGE_PIN U22 [get_ports {LEDs_out_0[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[2]}]
set_property PACKAGE_PIN U21 [get_ports {LEDs_out_0[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[3]}]
set_property PACKAGE_PIN V22 [get_ports {LEDs_out_0[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[4]}]
set_property PACKAGE_PIN W22 [get_ports {LEDs_out_0[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[5]}]
set_property PACKAGE_PIN U19 [get_ports {LEDs_out_0[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[6]}]
set_property PACKAGE_PIN U14 [get_ports {LEDs_out_0[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[7]}]
```

This constrains connect each individual bit of the LEDs_out port to a specific pin on the Zed-Board and specify the voltage of these pins.

Save the XDC file. Now, we have already finished our system architecture. The next step is to put our design on the ZedBoard. Select **Generate Bitsream** in *Flow Navigator*. If a dialogue appears prompting you to save your design, click **Save**. A dialogue window may requesting that you launch synthesis and implementation before starting the *Generate Bitstream* process. Click **Yes** if the request appears. It may take several minutes for synthesis, implementation and bitstream generation, you can take a break.

When bitstream generation is complete, a dialogue window will open to inform you. Select **Open Implemented Design**, and Click **OK**. You can observe how the FPGA resources are used by your design here.

Now, we just finished hardware part of our design. The next step is to write our software executed on the Zynq ARM Cortex-A9 CPU.

Select **File>Export>Export Hardware...** from the *Menu Bar*. The *Export Hardware for SDK* dialogue window will open. Ensure that the option to **Include bitstream** is selected, and click **OK**. Launch the SDK in Vivado by selecting **File>Launch SDK** from the *Menu Bar* and click **OK**.

The SDK will launch.

3.1 SOFTWARE APPLICATION

Once the SDK has launched, create a new **Application Project** by selecting **File > New > Application Project** from the *Menu Bar*. In the *New Project* dialogue, enter **Lab1** as shown in Fig.3.7. You can notice that at the bottom of *New Project* dialogue is **Board Support Package(BSP)**. We choose **Create New** here to let SDK generate the necessary support package (or drivers) depending on our hardware in the previous steps. We can also generate a full BSP by **File > New > Board Support Package** before we create a new application project. Click **Next>** and select **Empty Application** in *Available Template* window as Fig.3.8. Click **Finish**.

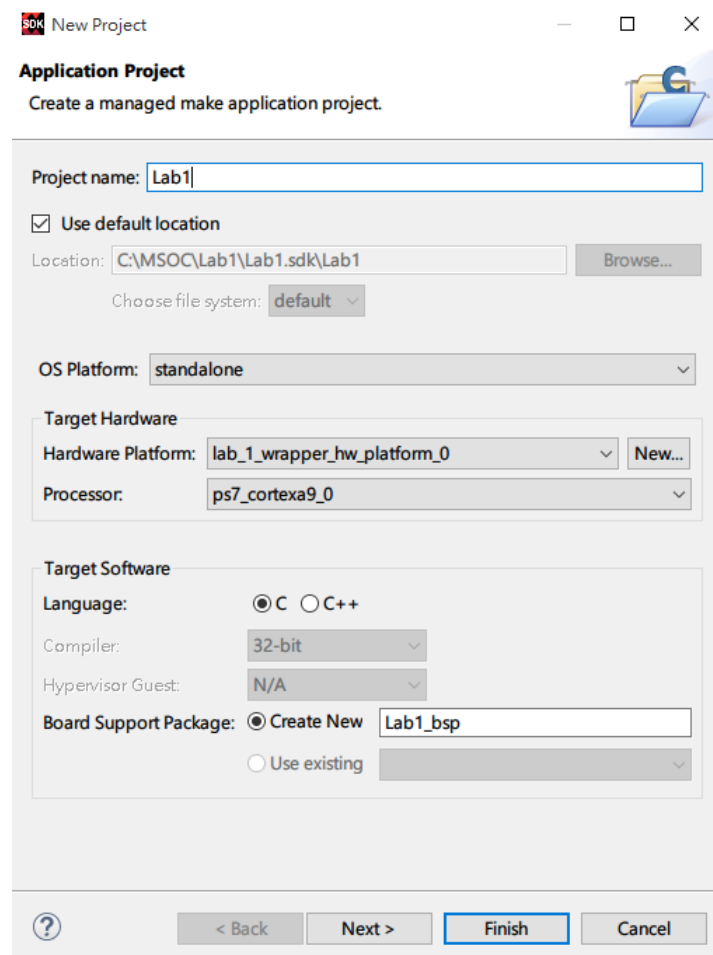


Figure 3.7: Create Application Project

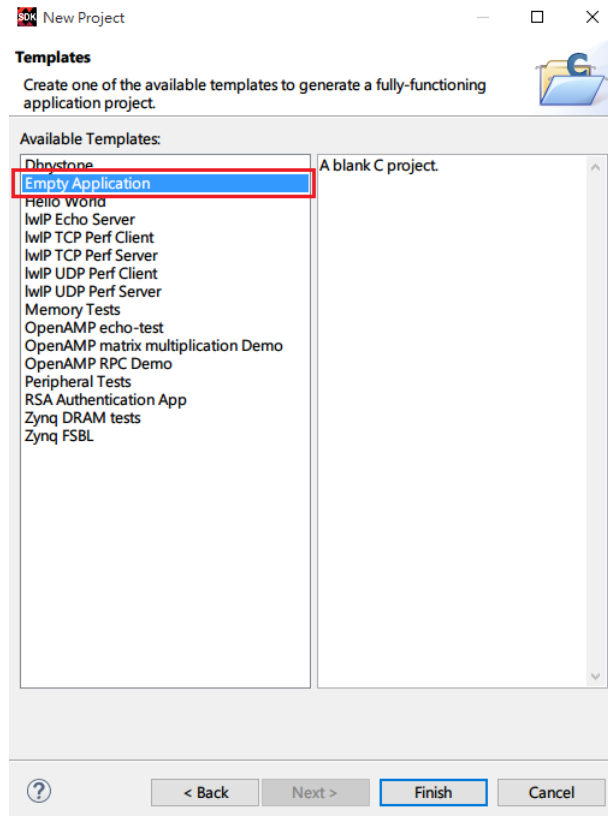


Figure 3.8: Empty Application

Although we get most drivers when SDK create BSP for us, the driver of the IP created by ourself cannot be generated by SDK. So we have to add this first.

Navigate to **Xilinx > Repositories** in *Menu Bar*. In the *Repositories Preferences* windows, click on **New**, as shown in Fig.3.9.

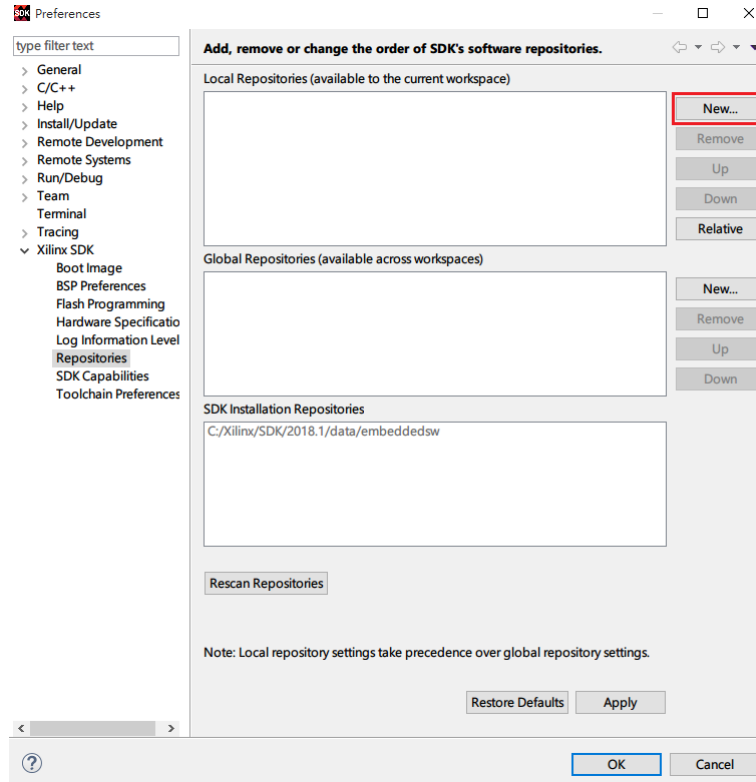


Figure 3.9: SDK Repository Peripherals window

Browse to the directory: **c:/MSOC/ip_repo/led_controller_1.0** as Fig.3.10.



Figure 3.10: led_controller repository selection

The **system.mss** tab should be open in the Workspace. If it is not, open it by expanding **Lab1_bsp** in *Project Explorer* and double-clicking on **system.mss**. At the top left of the **system.mss** tab, click **Modify this BSP's Settings** Fig.3.11.

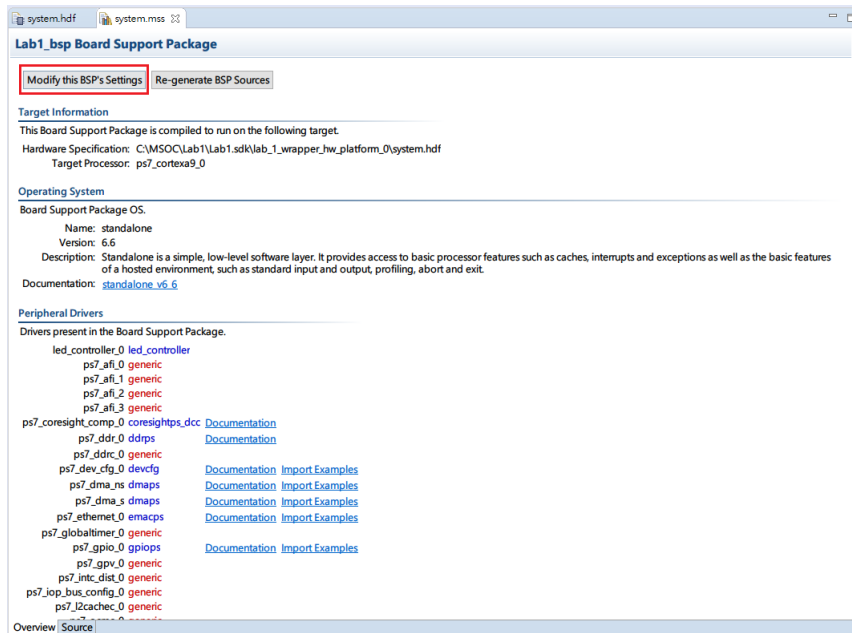


Figure 3.11

The Board Support Package Settings window will open. Select **driver** from the left-hand menu. From the list of components in the *Drivers* pane, identify **led_controller_0** and ensure **led_controller** is selected from the drop-down menu in the **Driver** column, as shown in Fig.3.12.

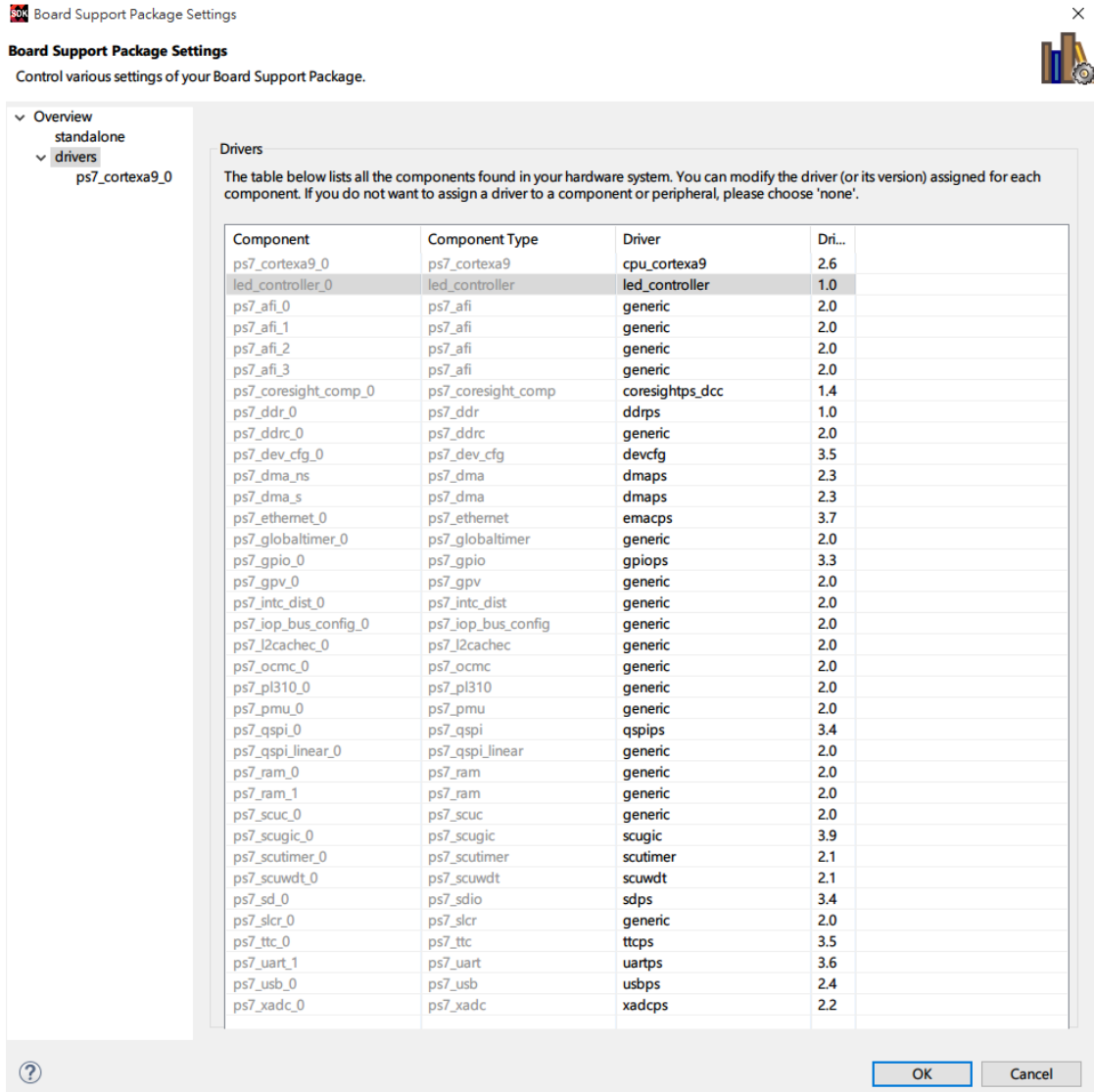


Figure 3.12: led_controller repository selection

Click **OK**.

Now we prepare all the necessary drivers for our system. The last step is to prepare our software to be run on CPU.

In fact, this software is already written for you. In *Project Explorer*, right click on **Lab1** and select **import**. An *Import* window will be opened, expand **General** and double-click on **File System** as Fig.3.13.

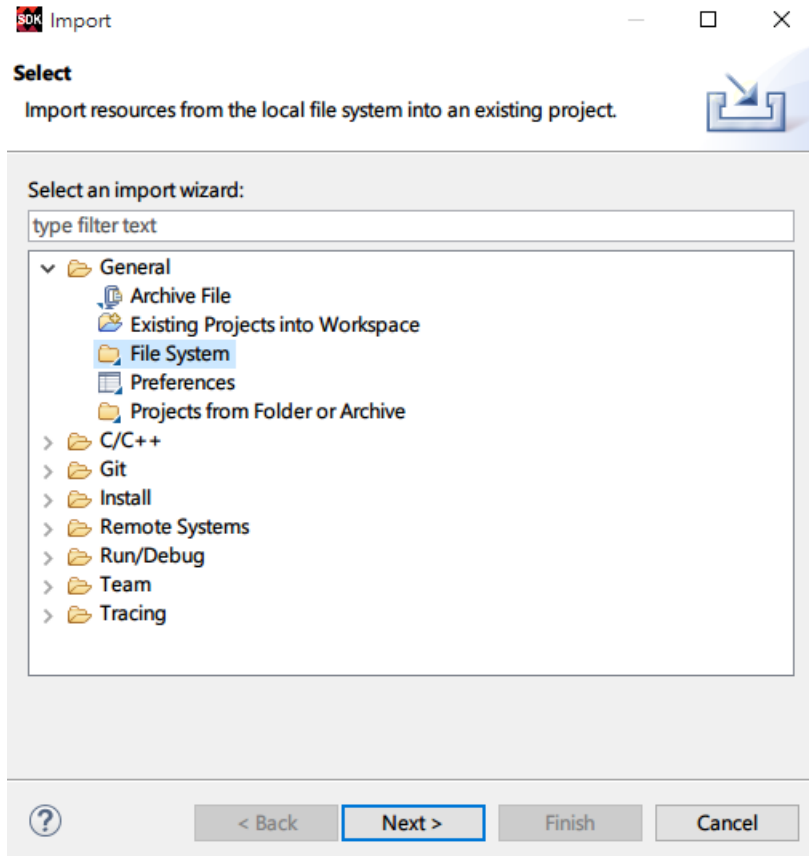


Figure 3.13: Import file

Click **Next**. Type `c:/MSOC/Lab1/source/` in *From directory* box or click **Browse...** to find this route. Select `led_controller.c` as in Fig.3.14.

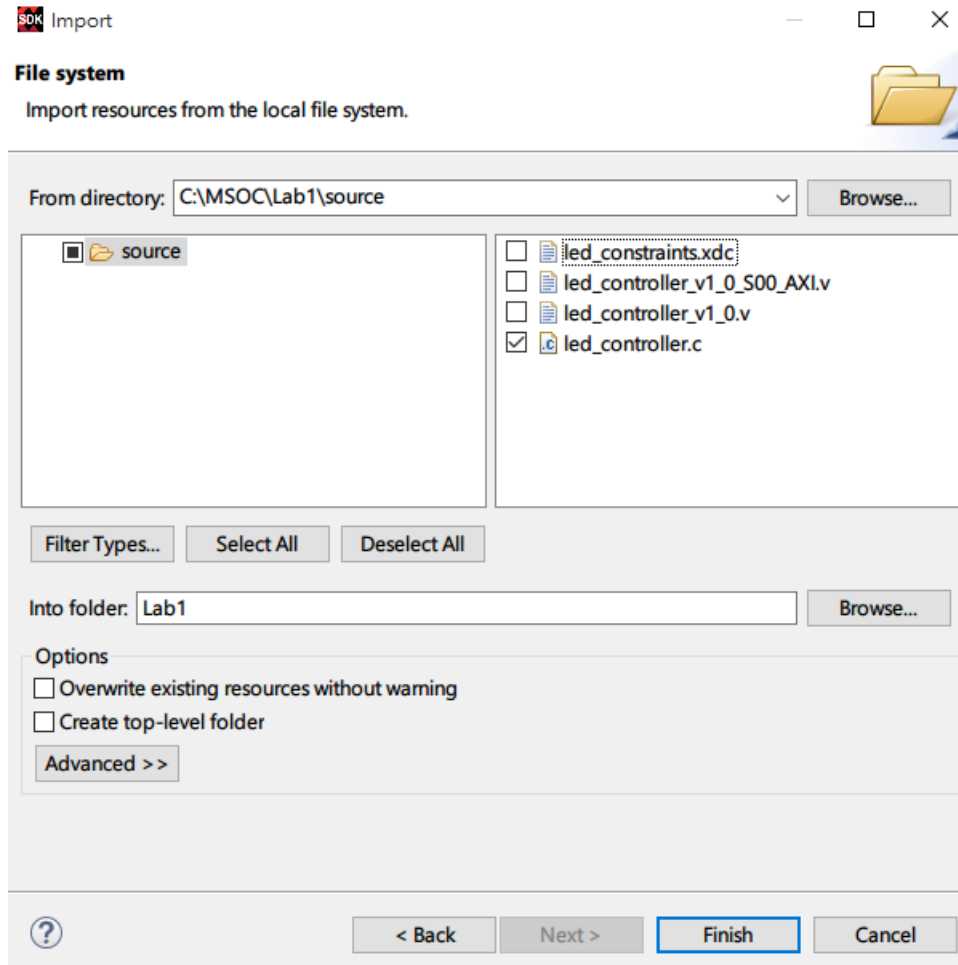


Figure 3.14: led_controller.c

Click Finish.

Before we execute our software, have our ZedBoard ready to connect to PC. Plug the power cable, connect **J17** and **J14** ports on the ZedBoard to PC with mirco-USB cables. The **J14** port is used to communicate between PC and ZedBoard. Turn on the ZedBoard power now. If your PC asks you for driver installation, select **Cypress CY7C64225 chipset**. If there is any problem in installation of the driver, please refer to [Cypress Chipset](#).

Now you need to open the terminal in SDK. Type **terminal** in the *Quick Access* box at the top-right as Fig.3.15 to add a terminal tab.

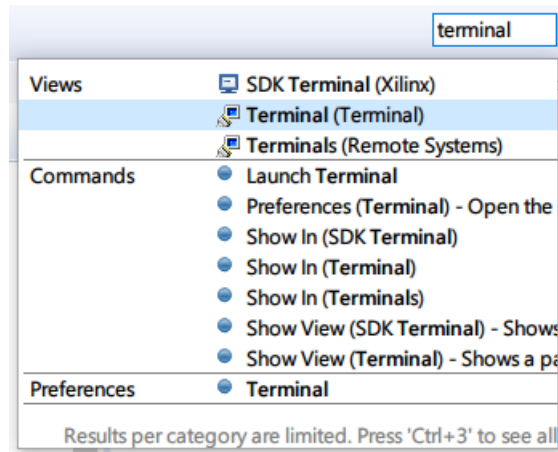



Figure 3.15: Quick Access

Find the **Terminal** tab as shown in Fig.3.16, it may appear at the bottom, bottom-right or top-right of SDK window. Click the  to connect the PC and the board through the UART. The **Terminal Settings** dialogue will appear, select your UART port (default COM3) and set the other options as Fig.3.17

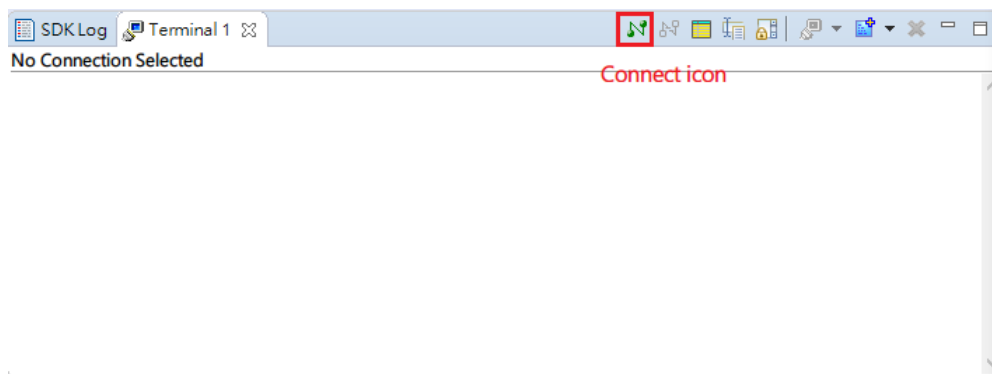


Figure 3.16: Terminal

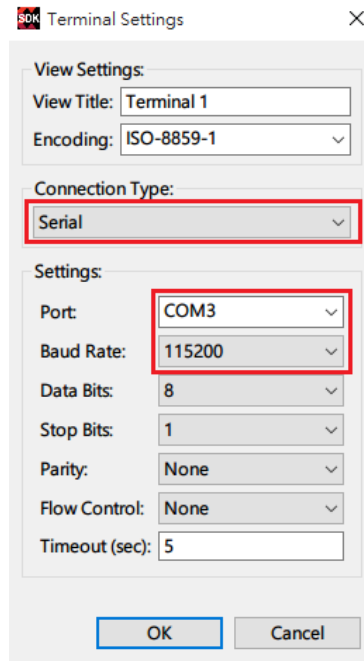


Figure 3.17: Terminal Settings

Click **OK** to initiate the new Terminal connection.

Before we execute our application, we have one last thing to do: **Program FPGA** by the bitstream generated by Vivado. Select **Xilinx > Program FPGA** from the *Menu Bar*. Click **Program**.

After the bitstream being loaded into the FPGA, the blue LED **LD12** on the board will be turned on. We can execute our last step now.

Right click on the **Lab1** in the **Project Explorer** in the left-hand pane. Select **Run As > Launch on Hardware(GDB)** as shown in Fig.3.18.

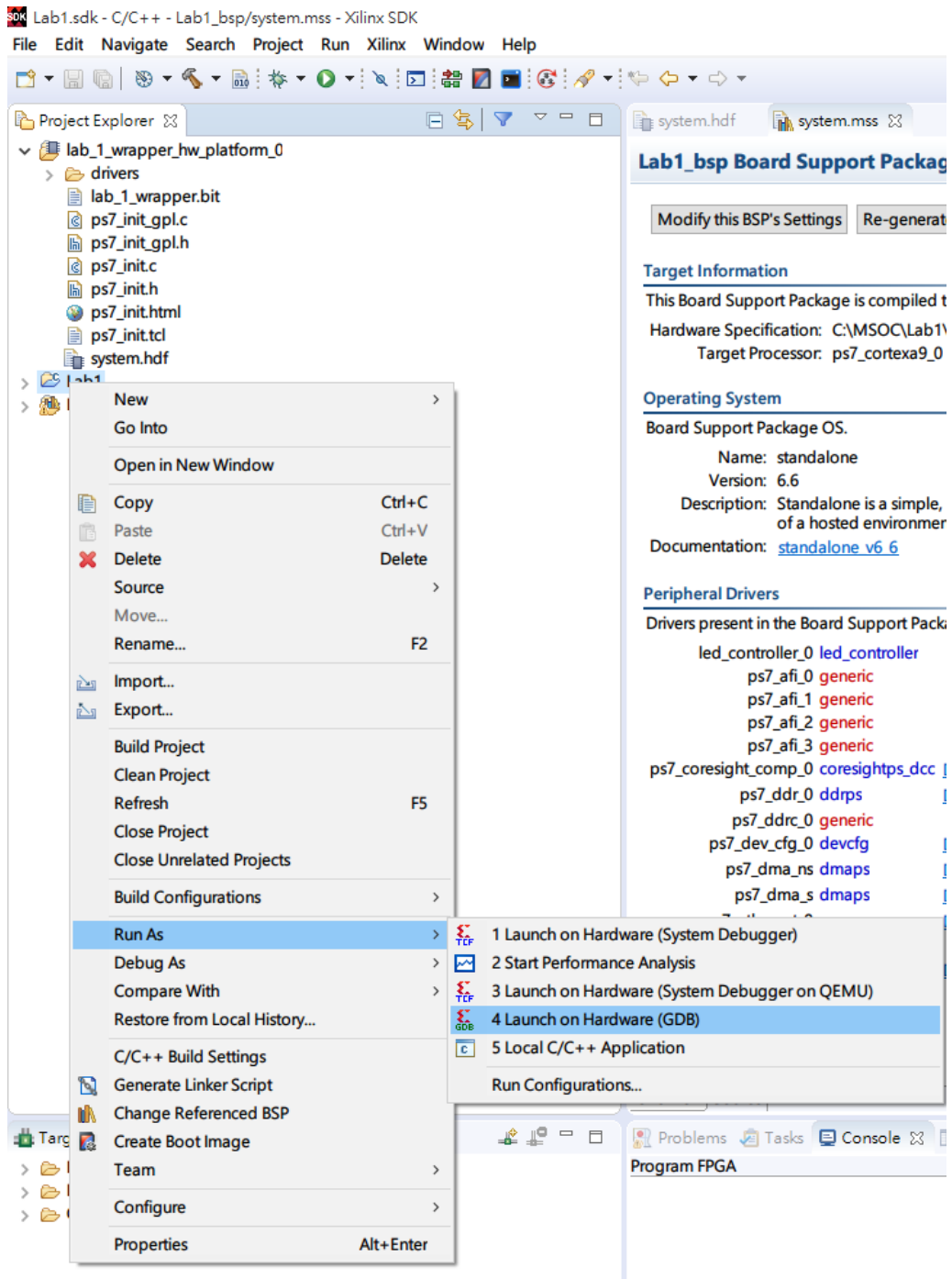


Figure 3.18: Run Application on hardware

This application sends an 8-bits integer value to PL part, and the LEDs on the board show this value. The value will also displayed on the terminal as Fig.3.19. The LEDs turn on or off

depending on the corresponding bit.

```
Serial: (COM3, 115200, 8, 1, None, None - CONNECTED) -
led_controller IP test begin.
-----
LED value: 0
LED value: 1
LED value: 2
LED value: 3
LED value: 4
LED value: 5
LED value: 6
LED value: 7
LED value: 8
LED value: 9
LED value: 10
LED value: 11
LED value: 12
LED value: 13
LED value: 14
LED value: 15
LED value: 16
LED value: 17
LED value: 18
LED value: 19
LED value: 20
LED value: 21
LED value: 22
LED value: 23
LED value: 24
```

Figure 3.19: Run Application on hardware