

Lab 2 - Zynq HLS Design Flow

Pin-Hung Kuo

May 18, 2018

1 INTRODUCTION

In Lab 1, we design a system display a value by LEDs. We will have a similar design in this lab, except for that the IP will be produced by high level synthesis (HLS).

2 CREATING IP IN HLS

Open Vivado HLS. You can find it in the Start menu: **Start > Programs > Xilinx Design Tools > Vivado 201X.X > Vivado HLS > Vivado HLS 201X.X**. Create a new project by click **Create New Project** as shown in Fig.2.1.



Figure 2.1: Create New Project

A New Vivado HLS Project window will appear, type **HLS_led_controller** in the *Project name* and change the *Location* to **c:/MSOC** as Fig.2.2.

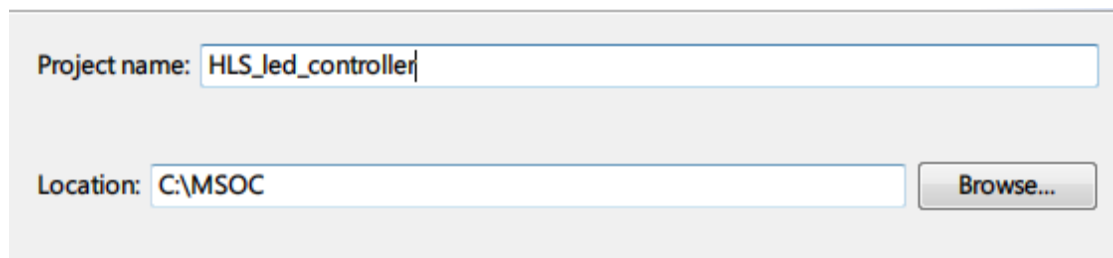


Figure 2.2: Project Name

In *Add/Remove Files* dialogue, type **led_controller** in *Top Function* box. We can add source files here if we have. In this lab, we want to create the source files by our self, so we just click **Next >** here.

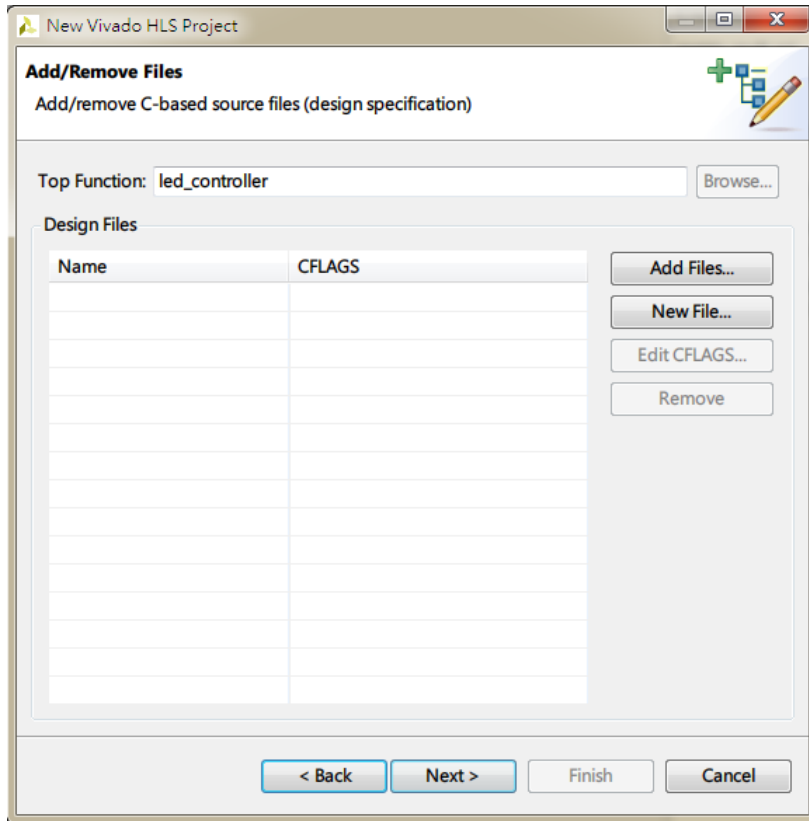


Figure 2.3: Add/Remove Source Files

You should see the dialogue as shown in Fig.2.4. As the hint, we can add testbench files here. We skip this step in this lab, just click **Next >**.

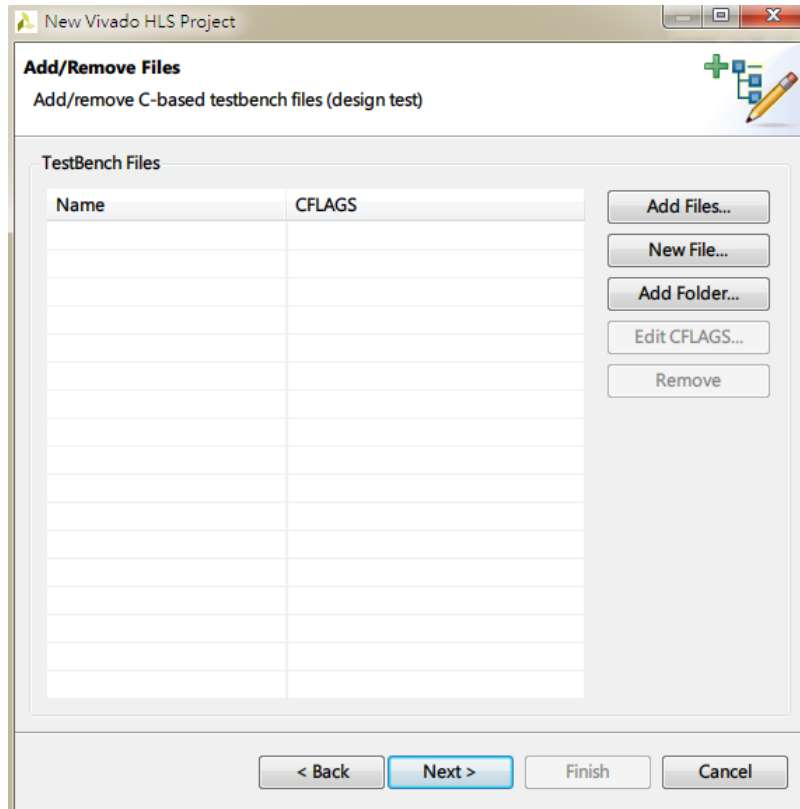


Figure 2.4: Add/Remove Testbench Files

In *Solution Configuration* dialogue, leave *Solution Name* and *Clock* as default. Click ... as shown in Fig.2.5 to select our part.

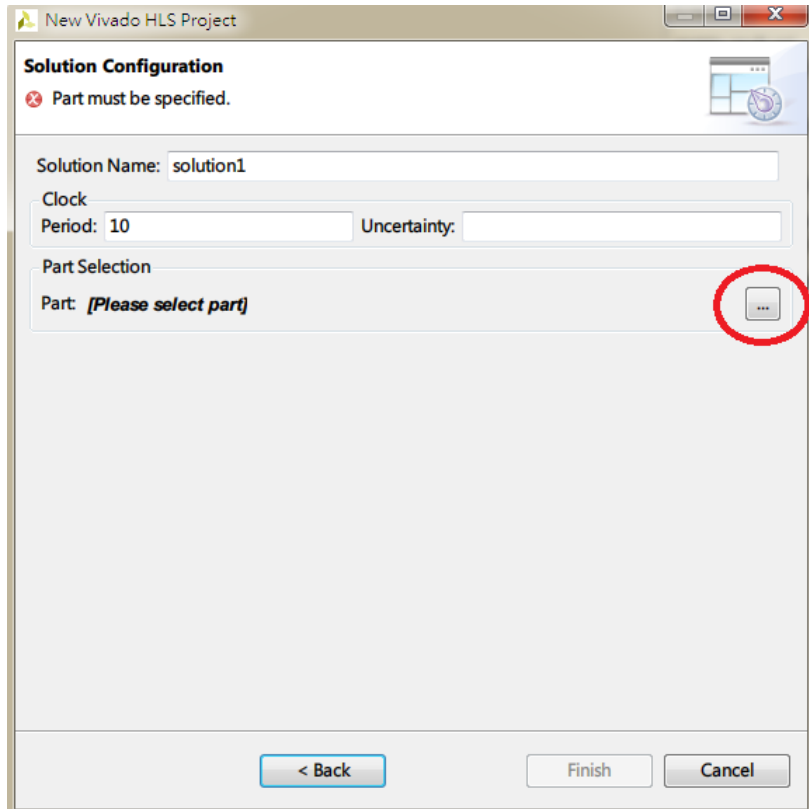


Figure 2.5: Solution Configuration

Similar to the part selection in Lab 1, we choose **ZedBoard Zynq Evaluation and Development Kit** in the *Device Selection Dialogue*.

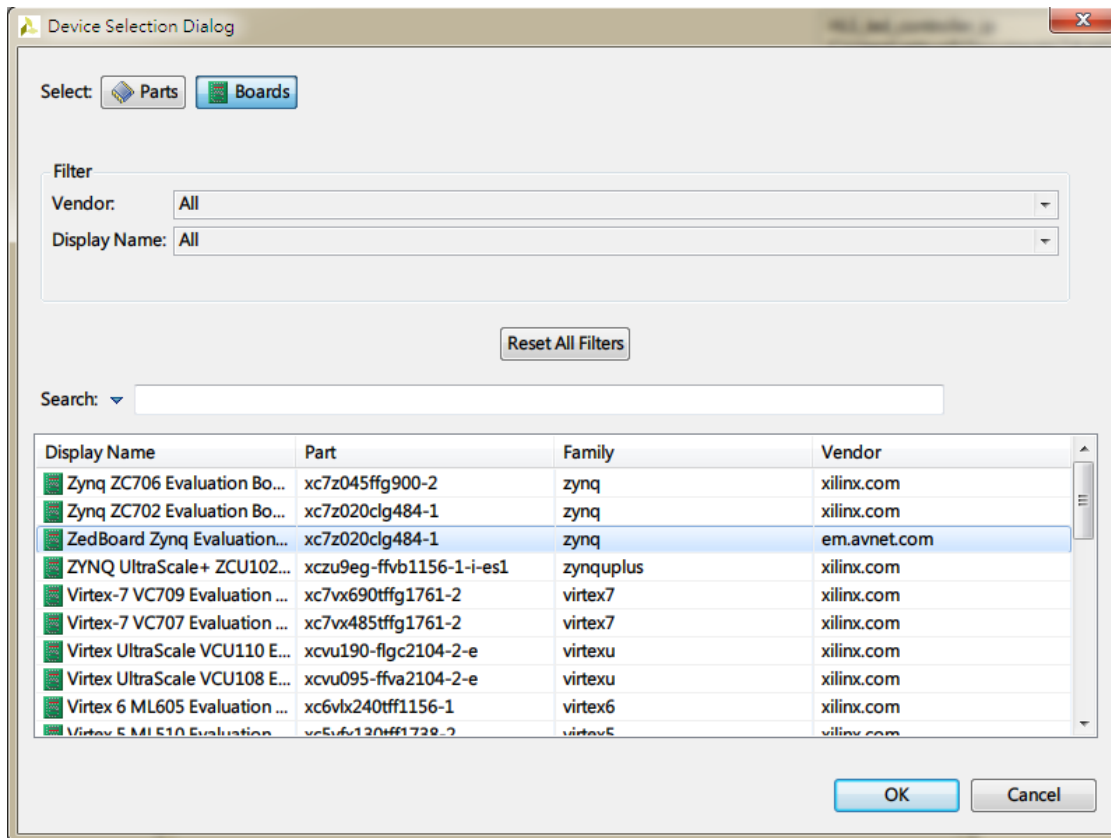


Figure 2.6: Device Selection Dialogue

Click **OK** and **Finish** to enter to our Vivado HLS project.

In Lab 1, our IP is simply pass the value from AXI slave port to output **LEDs_out**. In this lab, we will design a similar IP in C++.

Right-click the **Source** in the left-handed *Explorer* pane, and click **New Files...** to create a new source file as . You should be asked to save a file. Type **led_controller.cpp** as the file name.

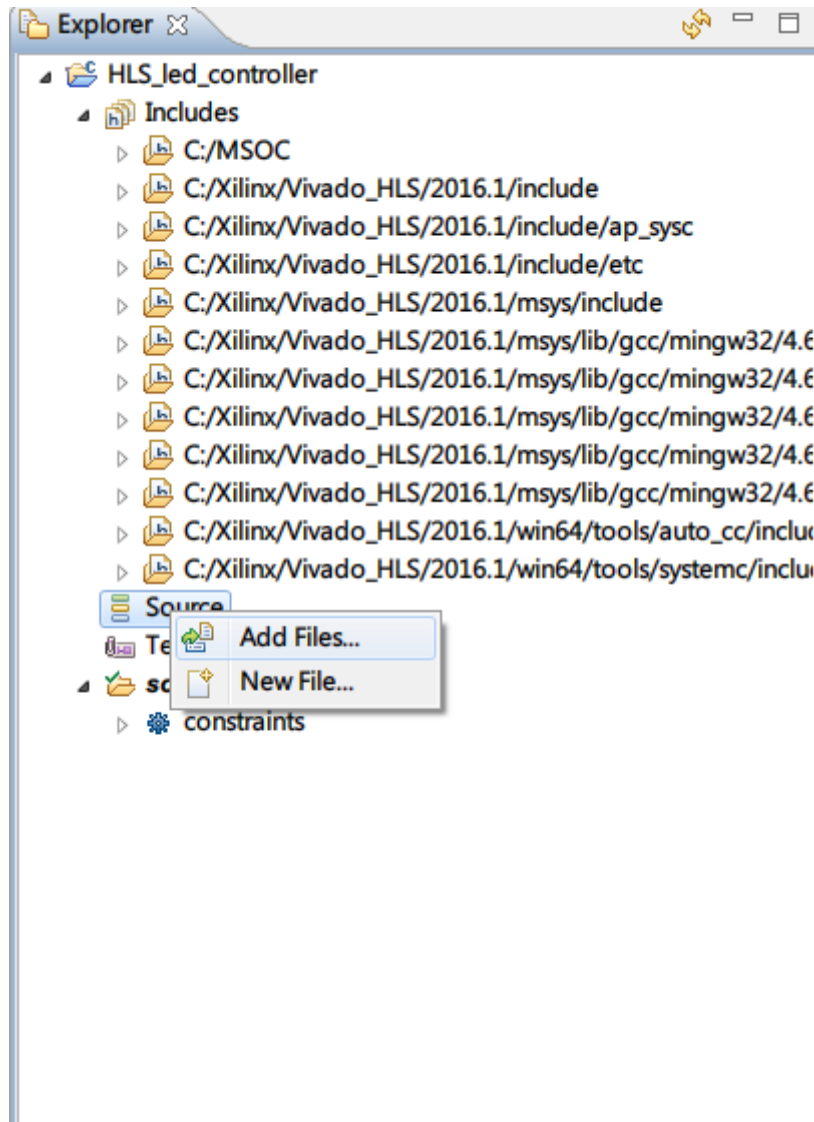


Figure 2.7: New File

An empty cpp file should be opened now in *Workspace*. Type the following code:

```
#include <ap_int.h>

void led_controller(int input, ap_uint<8>* LEDs_out){
*LEDs_out = input;
}
```

or you can find it in the lab file. Save the file by **Ctrl+s**. The data type of **LEDs_out** is **ap_uint<8>**, which is defined by Vivado HLS. This data type allow you to define a arbitrary length data

type. In our case, we just defined an 8-bits unsigned integer. You can realize this data type in detail by reading `ap_uint.h`, which can be opened by holding **Ctrl** and hovering you mouse on the `ap_uint.h`. The `ap_uint.h` will become a link, click on it to open the corresponding file. This operation can also be used in Vivado SDK, which is very useful to find the corresponding declarations, defines and so on.

Now, we have finished our source code. But how can we know if it works well? The testbench is used for this purpose. If you are familiar with HDL, you may be know about testbench. If you are not, it is OK. Testbench is also a code to include our design and test it if it works as our expectation. In HLS, we also have to integrate such testbench to examine our design. Right-click on the **Test Bench** in the **Explorer** pane and select **New File...** to create the testbench. Let the name of testbench be `led_controller_tb.cpp`. You should have an empty cpp file opened in the *Workspace*, type the following code


```
#include <stdio.h>
#include <ap_int.h>

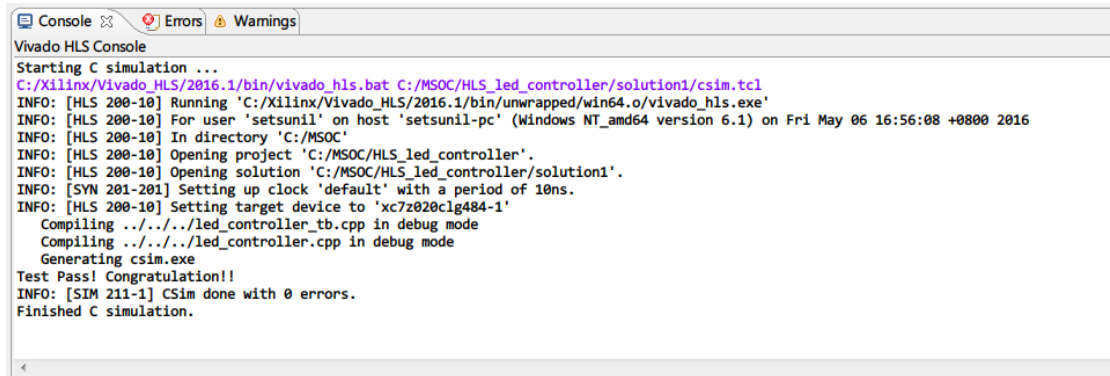
void led_controller(int input, ap_uint<8>* LEDs_out);

int main(){
ap_uint<8> LEDs_out;
int error=0;
for(int i=0;i<256;++i){
led_controller(i,&LEDs_out);
if(LEDs_out!=i)
error++;
}

if(error)
printf("There are %d errors\n",error);
else
printf("Test Pass! Congratulation!!\n");
return 0;
}
```

or find it in lab file. Our IP passes an 8-bits number between 0 and 255, so we test our IP with these 256 values. As you can see, the **main** function in HLS is used for testbench.

Click  and just click **OK** to simulate the source file. You should see the message as Fig.2.8.





```

Vivado HLS Console
Starting C simulation ...
C:/Xilinx/Vivado_HLS/2016.1/bin/vivado_hls.bat C:/MSOC/HLS_led_controller/solution1/csim.tcl
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado_HLS/2016.1/bin/unwrapped/win64.o/vivado_hls.exe'
INFO: [HLS 200-10] For user 'setsunil' on host 'setsunil-pc' (Windows NT_amd64 version 6.1) on Fri May 06 16:56:08 +0800 2016
INFO: [HLS 200-10] In directory 'C:/MSOC'
INFO: [HLS 200-10] Opening project 'C:/MSOC/HLS_led_controller'.
INFO: [HLS 200-10] Opening solution 'C:/MSOC/HLS_led_controller/solution1'.
INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
INFO: [HLS 200-10] Setting target device to 'xc7z020c1g484-1'
Compiling ../../led_controller_tb.cpp in debug mode
Compiling ../../led_controller.cpp in debug mode
Generating csim.exe
Test Pass! Congratulation!!
INFO: [SIM 211-1] CSim done with 0 errors.
Finished C simulation.

```

Figure 2.8: Simulation result

The simulation is to verify if your code is valid to be synthesized. Once we finish the simulation, we can now click the  icon to synthesize the HDL code from our cpp file. Once the synthesis is finished, a synthesis report should be opened in the *Workspace*. You can check the synthesis results here. Expand the **solution1** in the **Explorer** pane, you can find that the *systemc, vhdl and verilog* code of our IP have already been generated (under **syn** and **impl**). In RTL design flow, we usually take advantage of wave form for debugging. Vivado HLS provides this feature too. Click  to run the co-simulation. In co-simulation dialogue, selection your HDL language, where we choose verilog, and set the *Dump Trace* as **all** as shown in Fig.2.9.

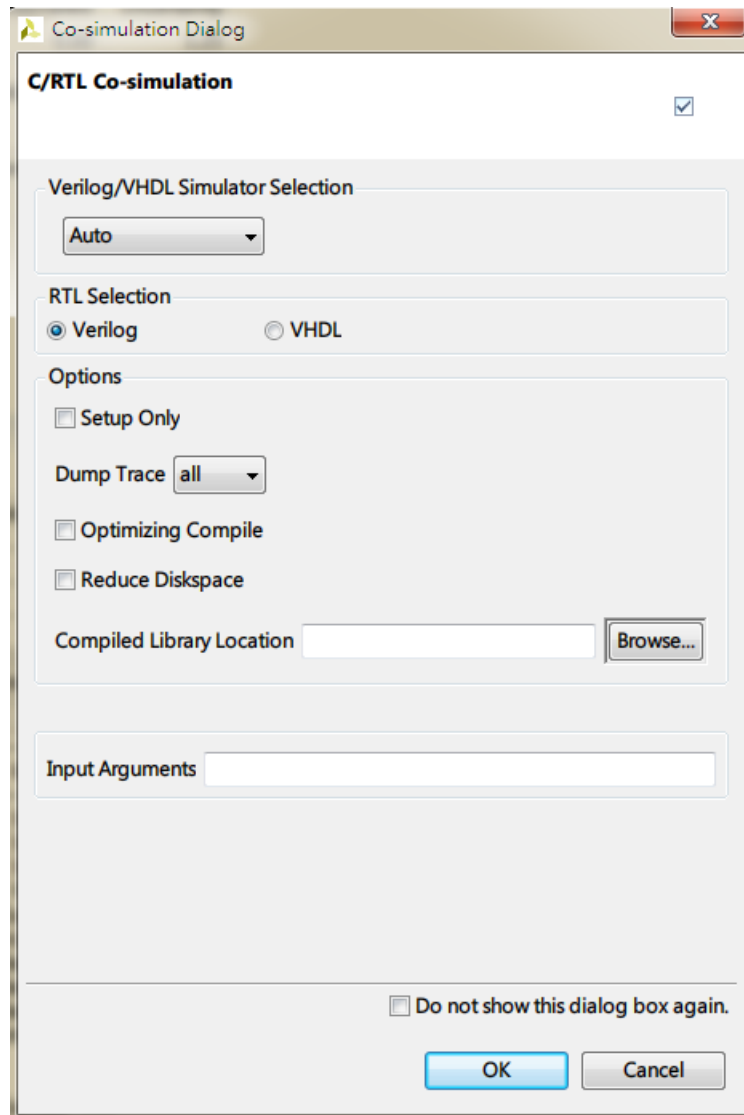



Figure 2.9: Co-simulation dialogue

The co-simulation tests the synthesized HDL code by our testbench cpp. After the co-simulation, you can observe that the  is activated. Click this icon to observe the waveform. A *Wave Viewer* window will be opened. Select the module you want to observed in the *Name* pane at the left-hand, and select the ports you want to observe in *Objects* pane in the middle of the window. Right-click the selected ports and click **Add To Wave Window** to observe the waveform as Fig.2.10. This *Wave Viewer* is very similar to the Synopsis Verdi/nWave, which is commonly used for RTL debugging.

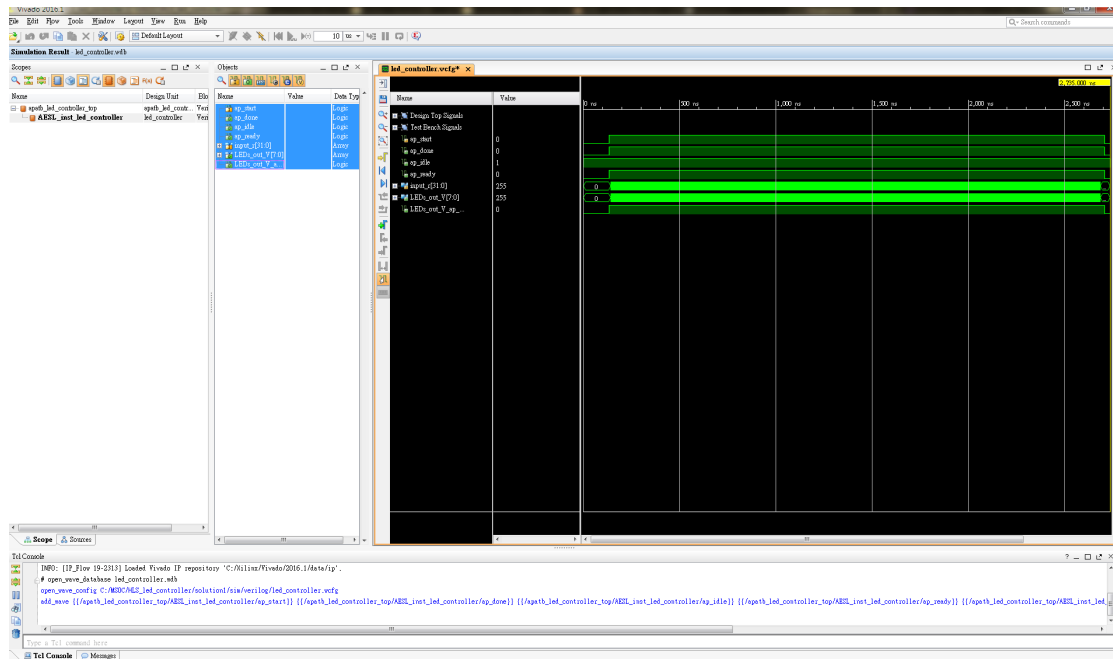


Figure 2.10: Waveform

Close the *Wave Viewer* and discard the waveform. Although we finished the synthesis few minutes ago, we have not added constraints/directives to our design. We are going to back to do this now.

Double click the **led_controller.cpp** in the *Explorer* pane to open the source file. Then select the **Directive** tab at the right-handed pane as shown in Fig.2.11

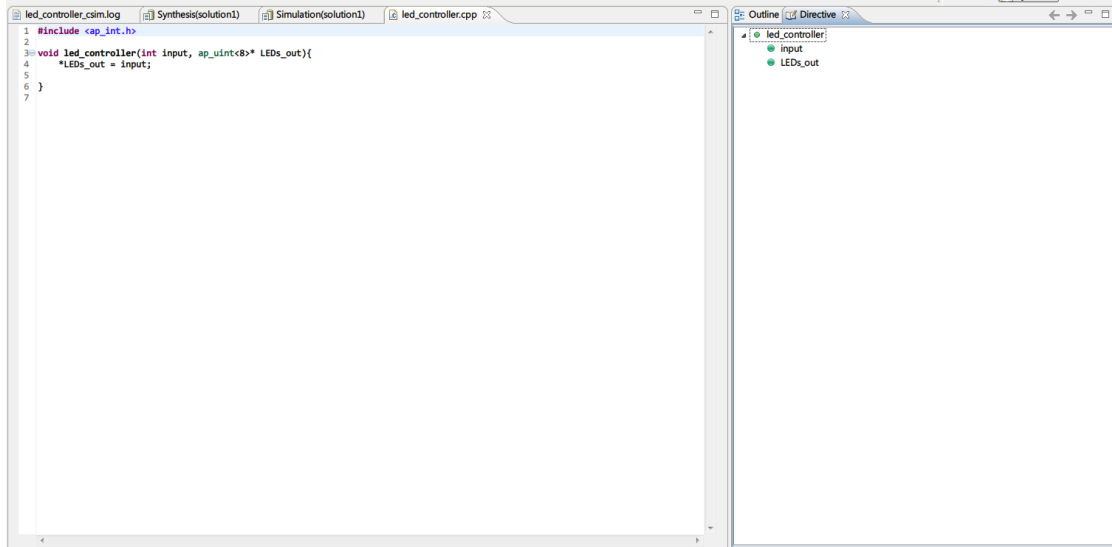


Figure 2.11: Directive

Right click on the **led_controller** in *Directive* pane and select **Insert Directive**. A **Vivado HLS Directive Editor** dialogue will show up. Select *Directive* as **INTERFACE** in the drop-down menu and set **mode(optional)** as **s_axilite**. The directive should be as that of Fig.2.12.

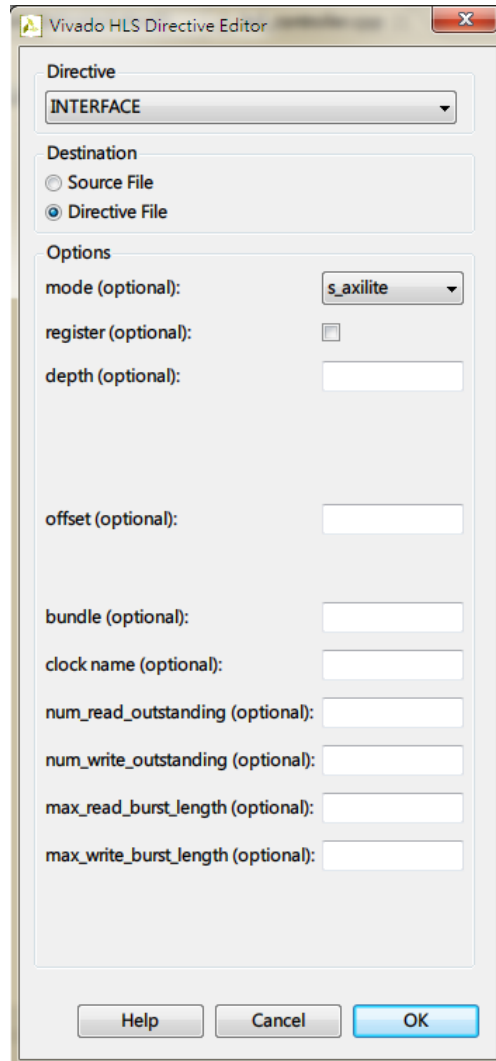


Figure 2.12: HLS Directive Editor

Click **OK**. We have to define our IP as having a **ap_ctrl_none** interface to remove unneeded control signals. Right-click on **led_controller** in *Directive* pane, and select **Insert Directive**. As before, select **INTERFACE** again. This time, we set the mode as **ap_ctrl_none** and click **OK**.

By the similar way, we set the **input** as **s_axilite** and the **LEDs_out** as **ap_none**. Check your *Directive* pane if it is same as Fig.2.13

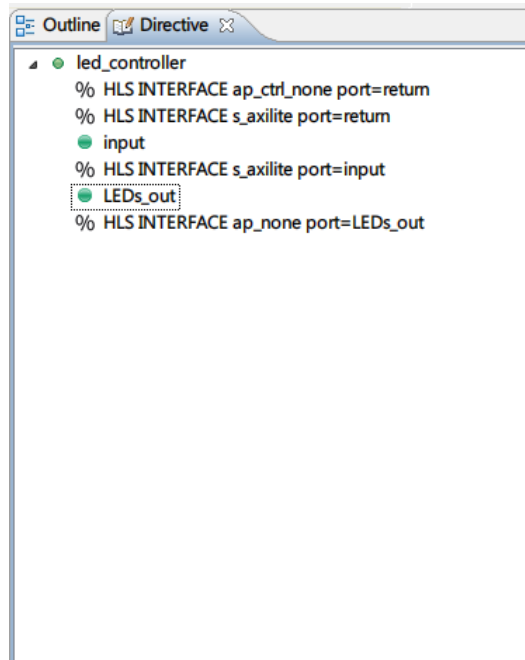




Figure 2.13: All Directives

With directives, we can now generate an IP. Although we did that few minutes ago, that is for co-simulation. This time we will generate the IP can be used in Vivado. It is worth noting that once we add the directives, the co-simulation will fail.

Now we run synthesis again by clicking . After the synthesis, now we can package our IP. Click  to export RTL. A *Export RTL Dialogue* will show up, click **OK** leaving the sets be default.

Now we just finish the HLS IP creation. We will integrate this IP into our system.

3 INTEGRATE HLS IP

Because it is very close to Lab 1, we will roughly go through the steps which we did once in the last lab.

Create a new Vivado project **Lab2. Create Block Design**. Now we have to integrate the HLS IP into our system. Click **Settings** in the tool bar as shown in Fig.3.1



Figure 3.1: IP Settings

In IP defaults, click **+** under IP Catalog to add our IP as Fig. 3.2. Select **c:/MSOC/HLS_led_controller** and click **OK**.

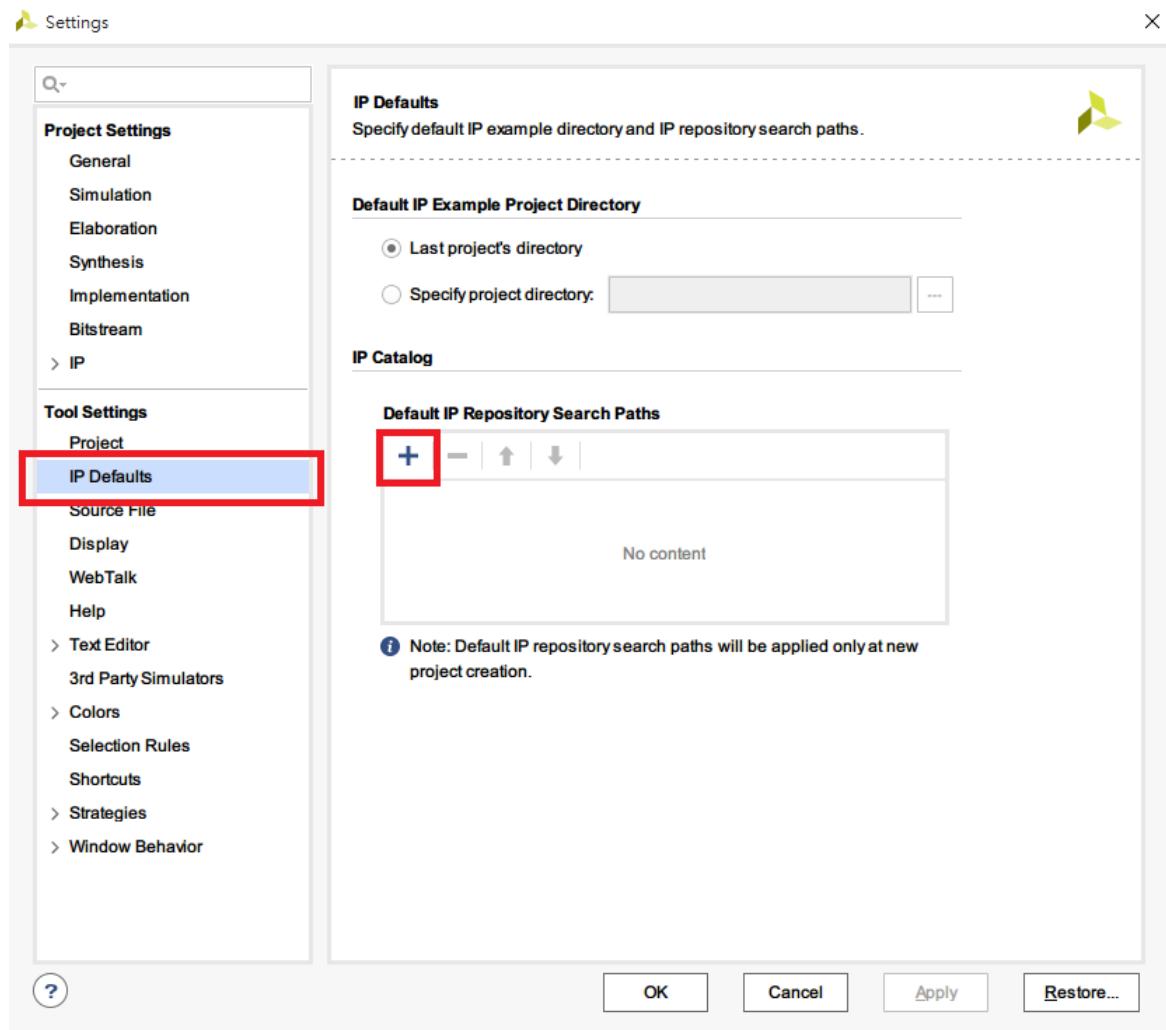


Figure 3.2: IP Repository

Click **OK** to return to our design.

Add our IP by the same method as Lab 1.

Our IP will be added to the diagram as Fig.3.3.

NOTE! Unfortunately, there is a bug in the newest Vivado 2018.1. If you cannot find the **led_controller** in IP Catalog as Fig.3.3, you can fix it by following steps:

1. Close your project.
2. Delete your project folder, e.g. c:/MSOC/Lab2/
3. Create a Lab2 project again
4. Now you can find the IP, Magic!

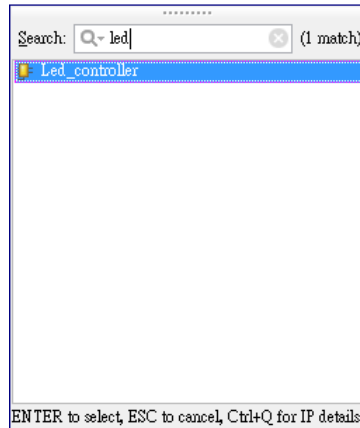


Figure 3.3: HLS_led_controller

Now, follow the steps as Lab 1.

Note that the port name of our HLS IP is a little different from that of Lab 1, therefore you should modify port name from **LEDs_out_0** into **LEDs_out_V_0** in the constraints file. Or you can find it in the lab2 zip file. Observe the difference between wrapper.v files of these two labs.

After the bitsream being generated, launch SDK.

Please create a new project **Lab2**. All files you need can be found in the lab zip file.

Find the difference between the source file of Lab 1 and that of Lab 2.

In the **Xilinx Tools > Repositories** step, please choose **C:/MSOC/HLS_led_controller** folder.

Run the application, and you should have the same result as Lab 1.