

# Tutorial 10: PYNQ Overlay Tutorial (Part 2)

---

## (Part 2: Test your custom IP)

---

You can find more details about this tutorial in [https://pynq.readthedocs.io/en/latest/overlay\\_design\\_methodology/overlay\\_tutorial.html](https://pynq.readthedocs.io/en/latest/overlay_design_methodology/overlay_tutorial.html)

### Objectives

---

After completing this tutorial, you will be able to:

- Build the SoC system in Vivado;
- Apply the custom IP built by Vitis HLS in tutorial 9 to the above SoC system;
- Write driver to configure registers;
- Test the custom IP on Pynq board.

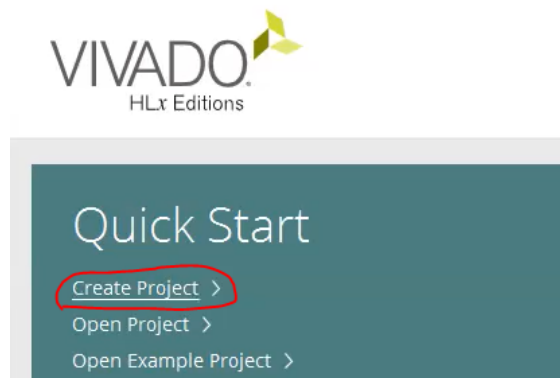
### Steps

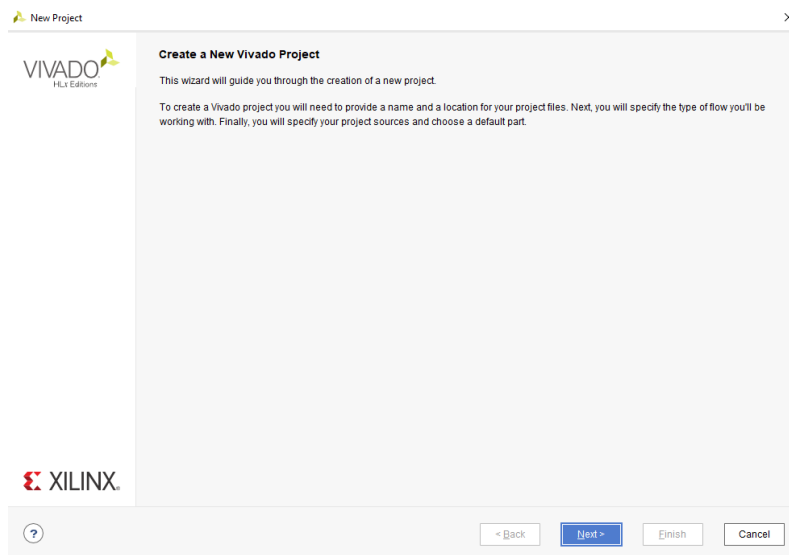
---

#### 1. Build SoC System in Vivado

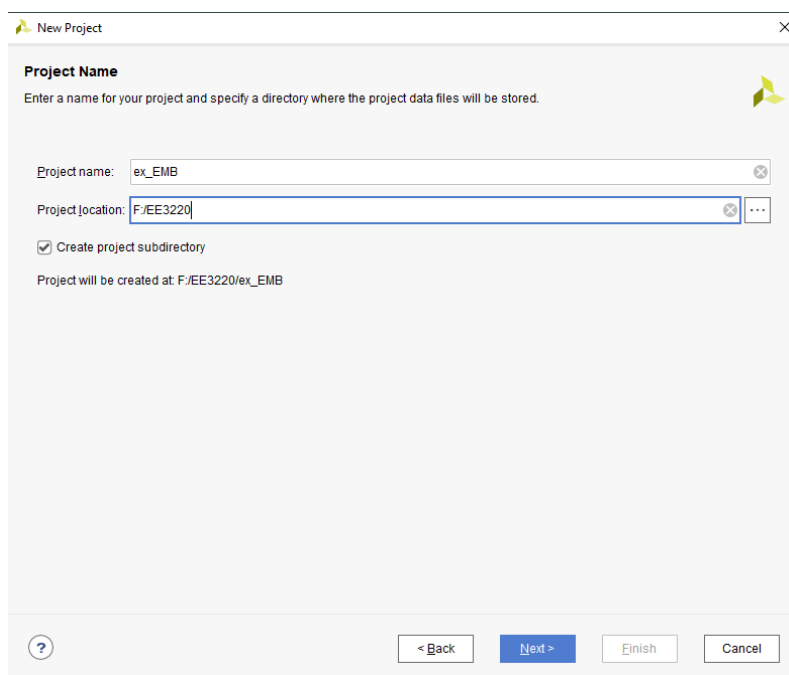
1. Create a new project named **ex\_EMB** in Vivado.

Open Vivado 2020.1. And click **Create Project**. Then click **Next**.

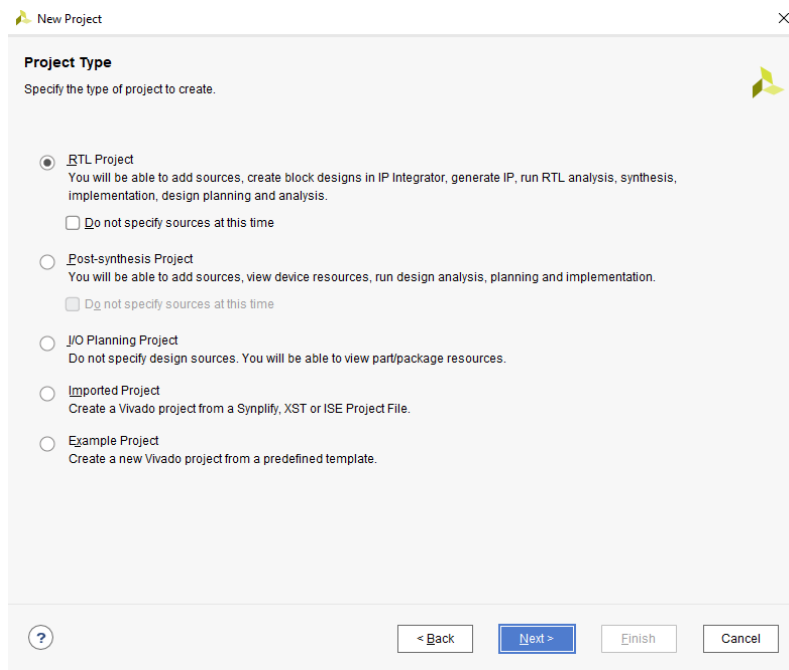




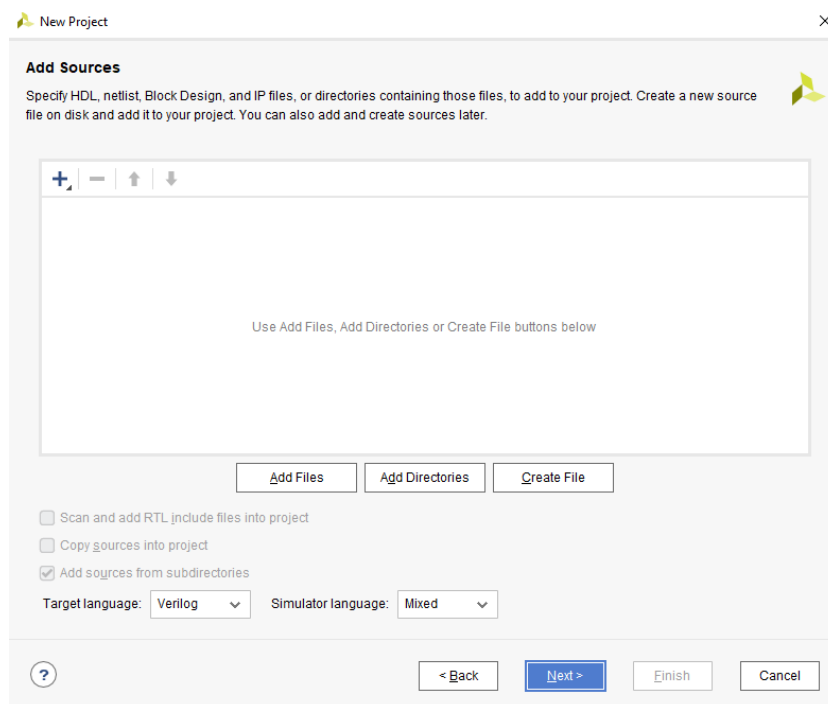
Choose a **location** for your project and specify the project **name**, then click **Next**.



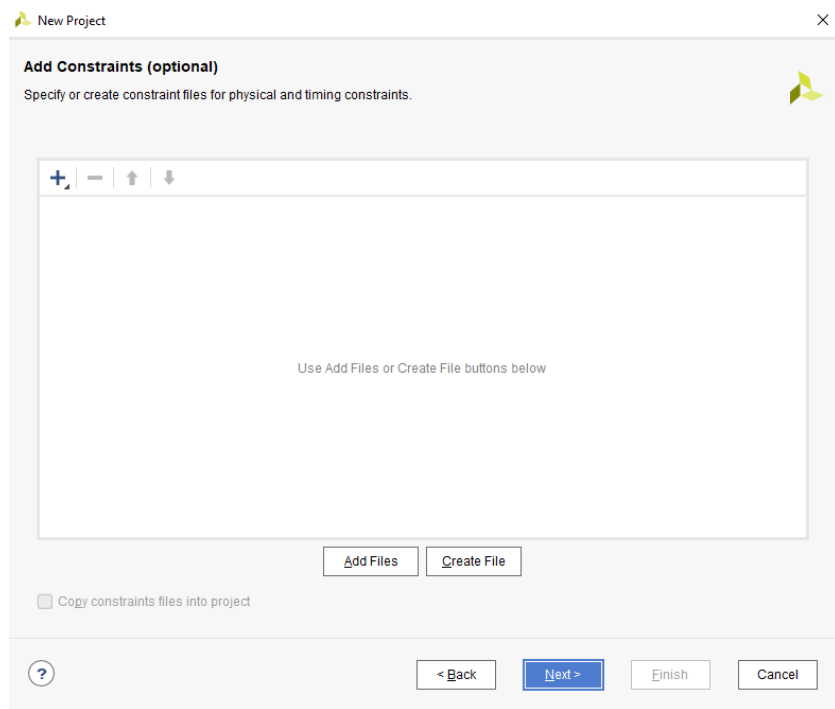
Choose **RTL Project**, then click **Next**.



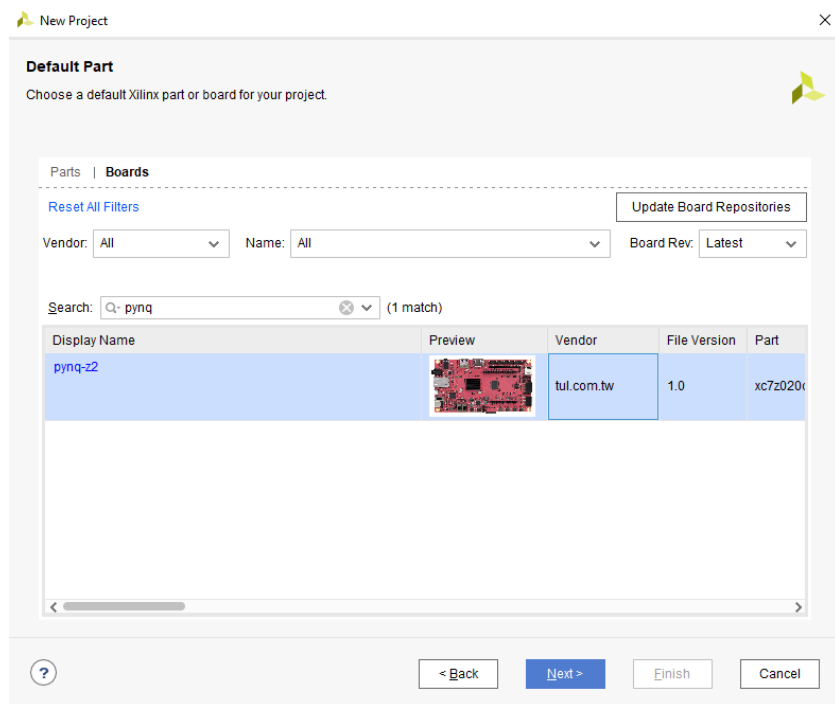
Click **Next** to create an empty project with no **source code**.



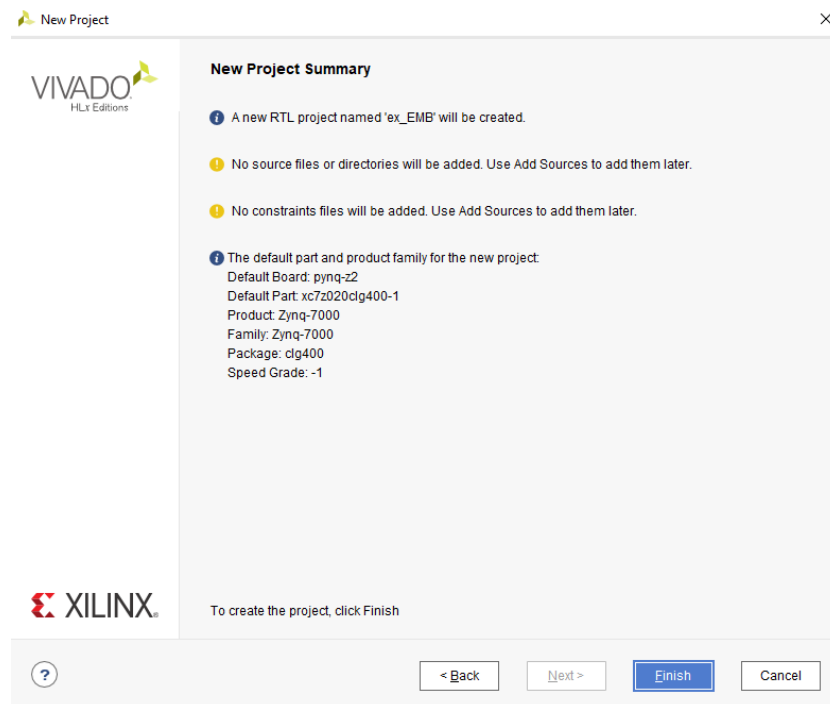
Click **Next** to continue without adding any **constraint**.



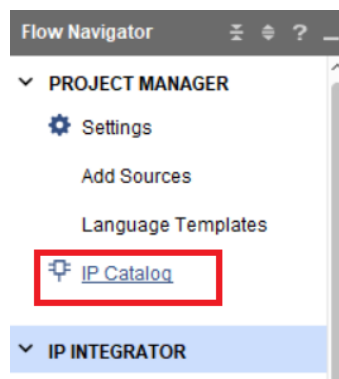
Choose **Boards** tab, and search pynq. Then choose **PYNQ-Z2** and click **Next**.

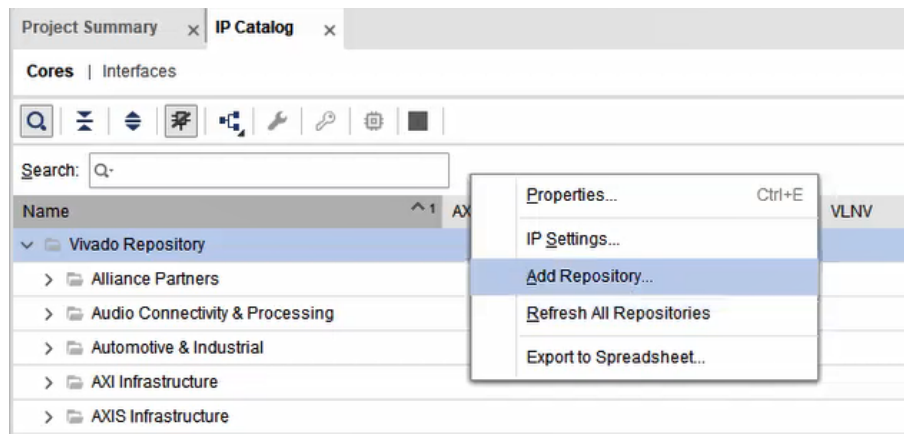


Cross check the summary and click **Finish** to create the project.

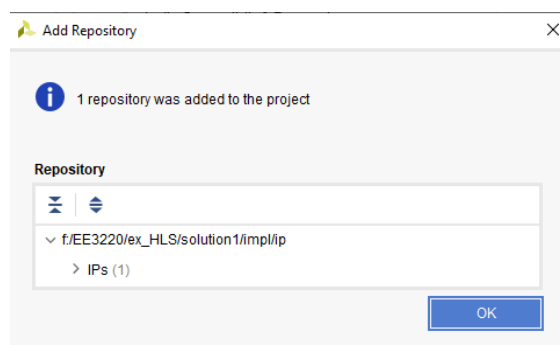
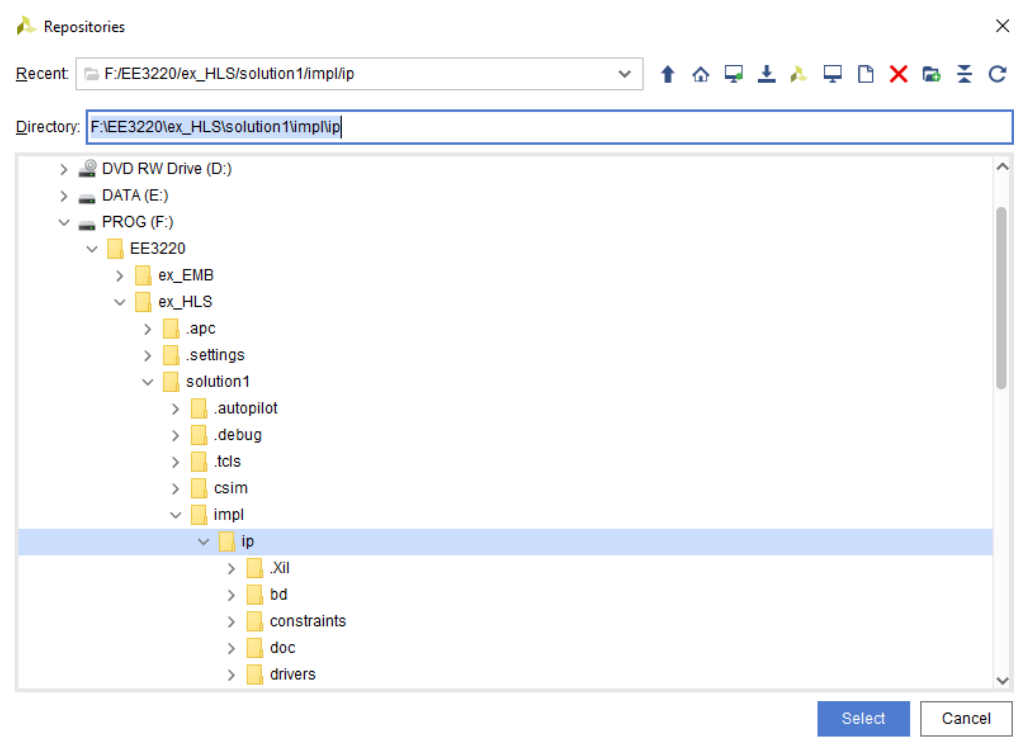


2. Find and click **IP catalog**. In the window of **IP catalog**, **right click** and open **Add Repository....**

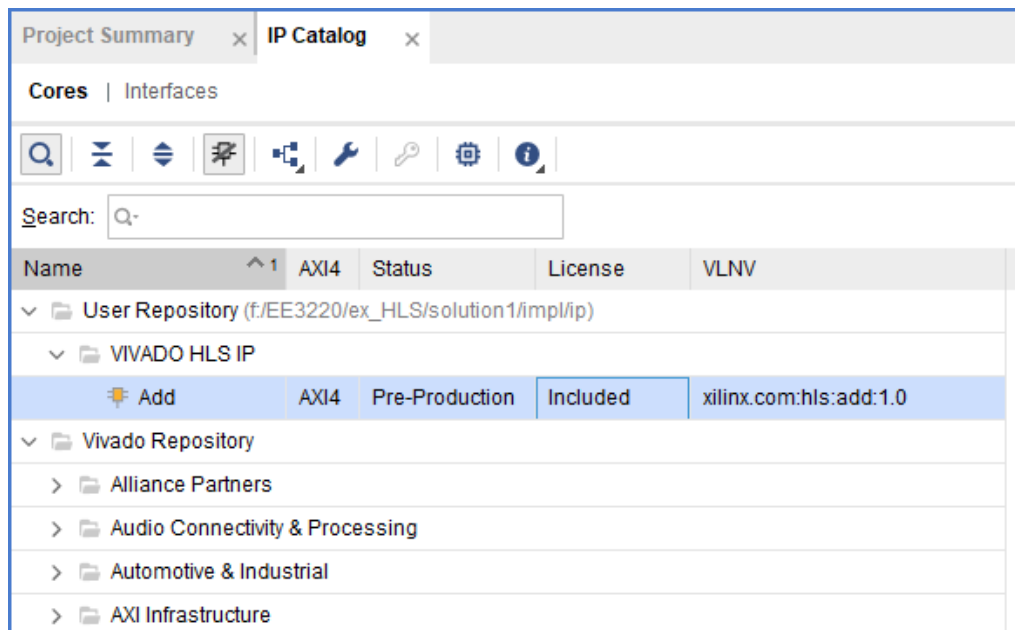




3. In the previous tutorial, we exported RTL design in Vivado HLS. The exported **Add IP** is in **HLS Project->Solution->impl->ip** folder. Select this folder and add it to the IP repository. Then click **OK**.

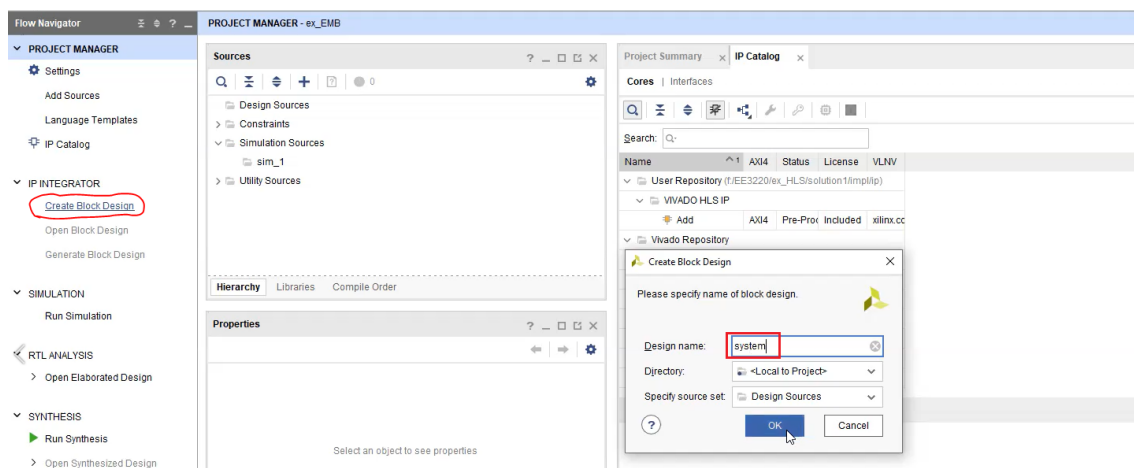


Now the **Add** IP is in the repository.

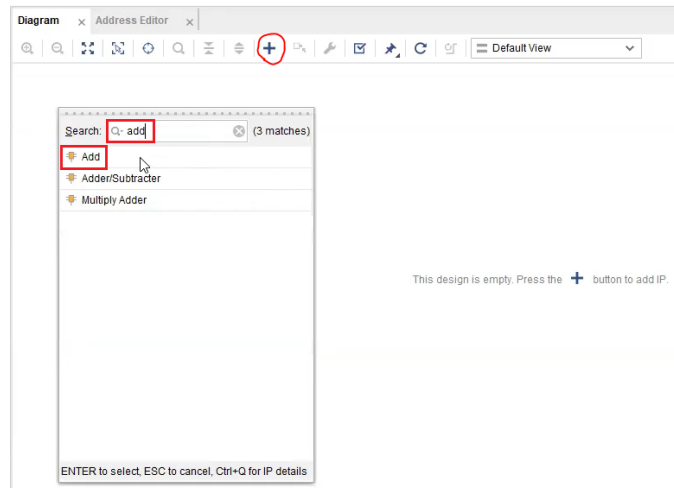


#### 4. Create a new block design named **system**.

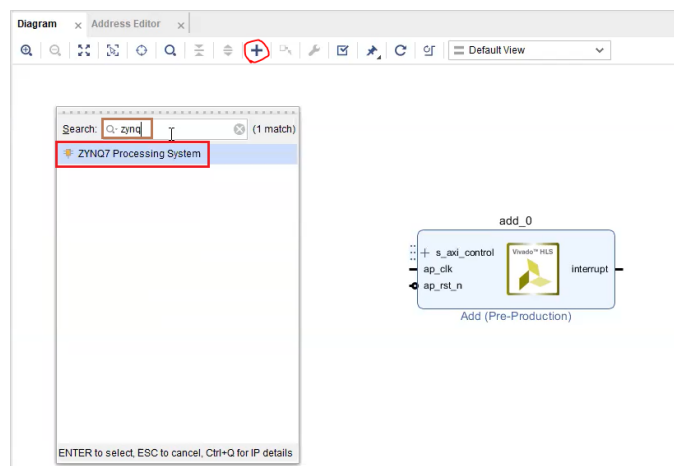
In **Flow Navigator**, click **create block design**. In the **Create Block Design** window, set Design name as **system**, then click **OK**.



Click **+**, then search add in the catalog, and **double click** the **Add** IP. The **Add** IP will be added to the design.

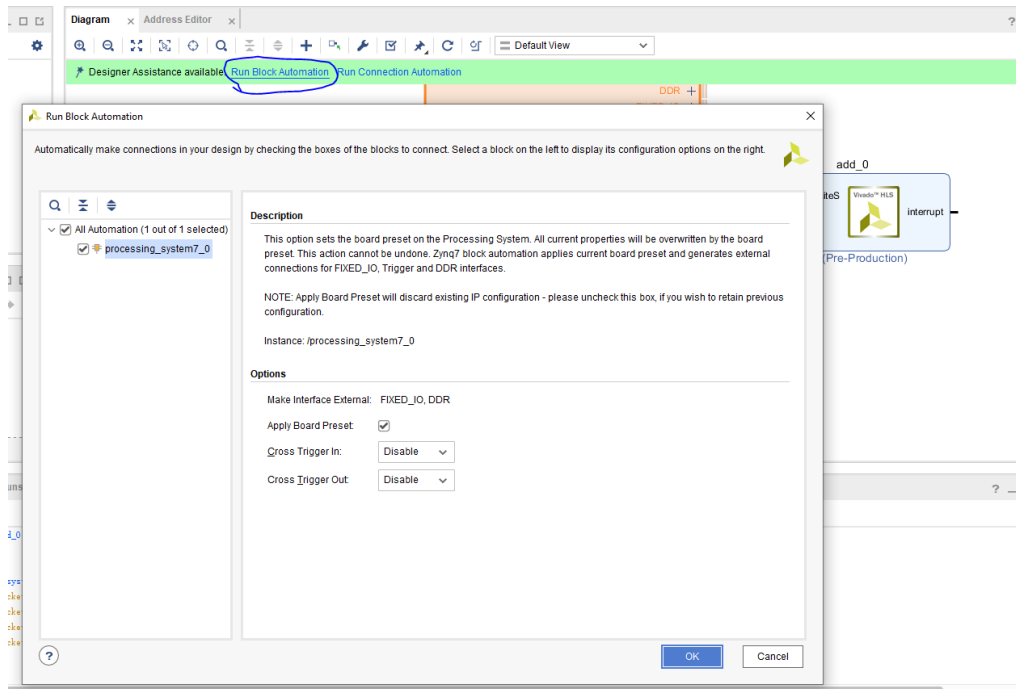


Click +, then search zynq in the catalog, and **double click** the **ZYNQ Processing System** IP. The **ZYNQ Processing System** IP will be added to the design.

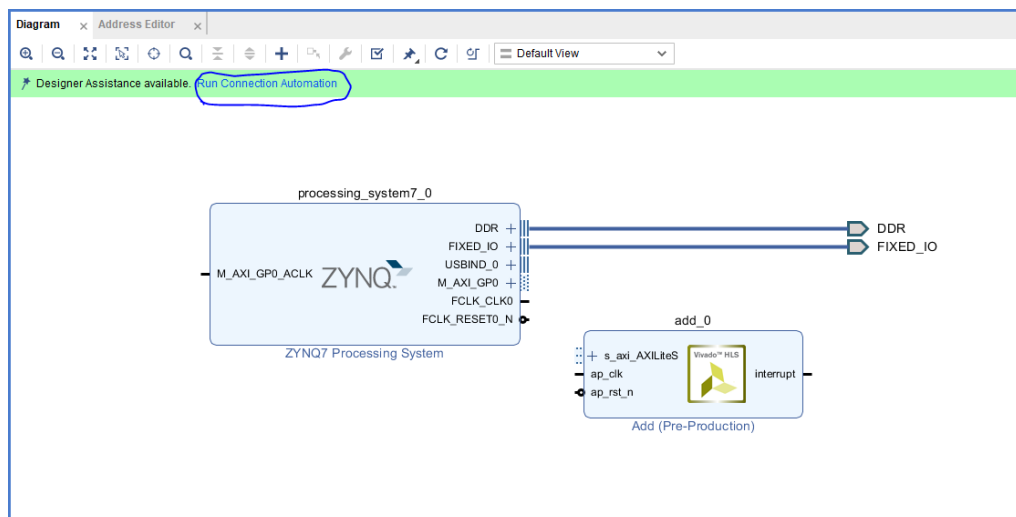


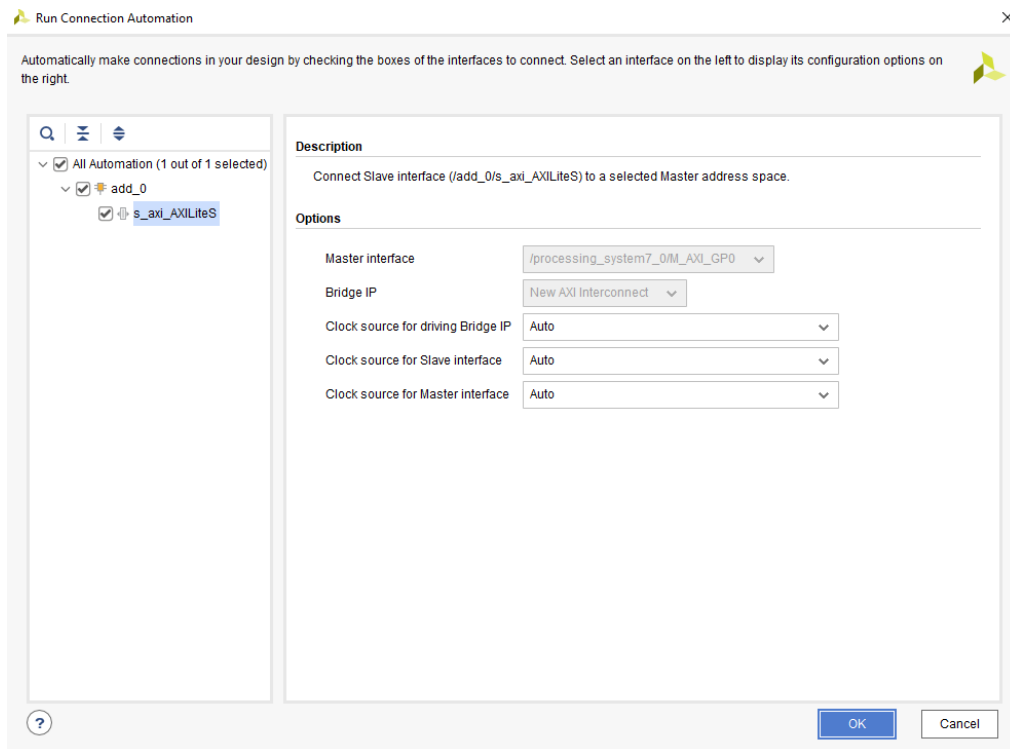
Click **Run Block Automation**, and choose the following configuration then click **OK**.



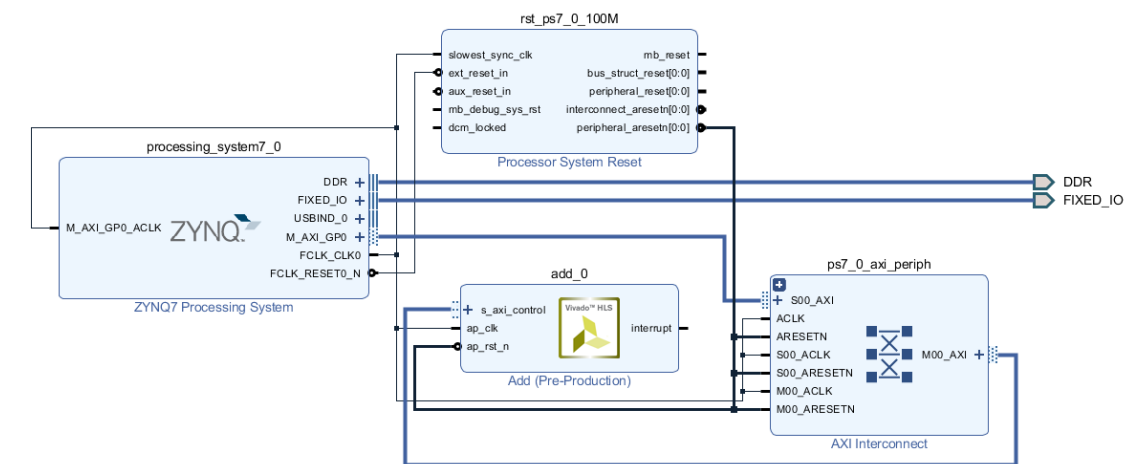


Click **Run Connection Automation**, and choose the following configuration then click **OK**.





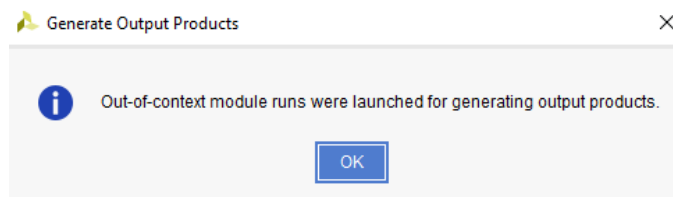
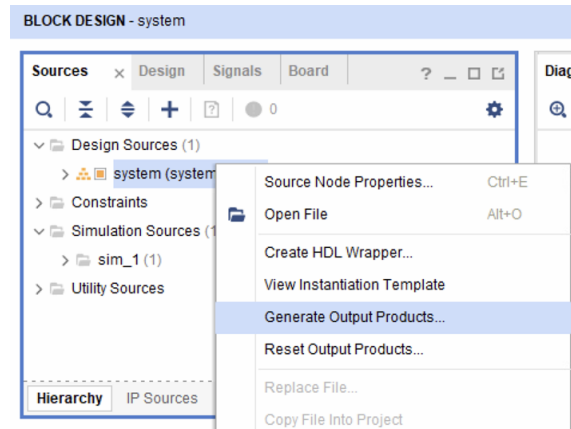
You are expected to build a SoC system in the following figure, with Zynq PS, Processor System Reset, the Add IP built by Vitis HLS in tutorial 9, and AXI Interconnect.



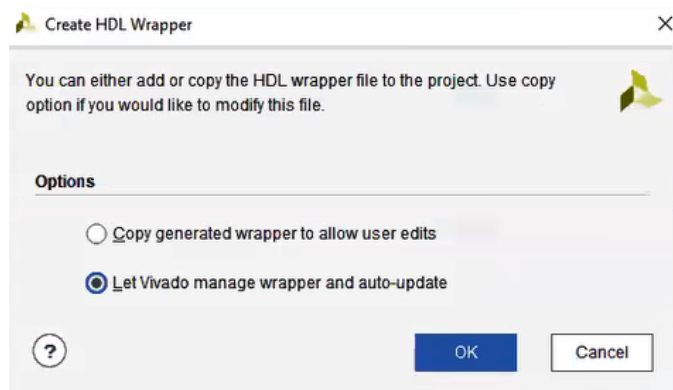
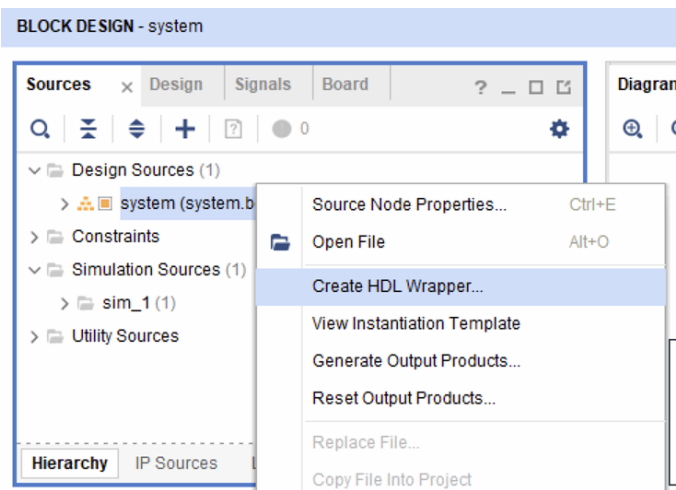
Click on the **Regenerate Layout** button. Then click on the **Validate Design** button and make sure that there are no errors. You can refer to **Tutorial 8, Section 3** for details.

5. Create a top-level module for block design and generate bitstream file. There is no need to add constraint in this tutorial.

In the **Sources** panel, **right click** on **system.bd**, and select **Generate Output Products...** and click **Generate** to generate the Implementation, Simulation and Synthesis files for the design. Then click **OK**.

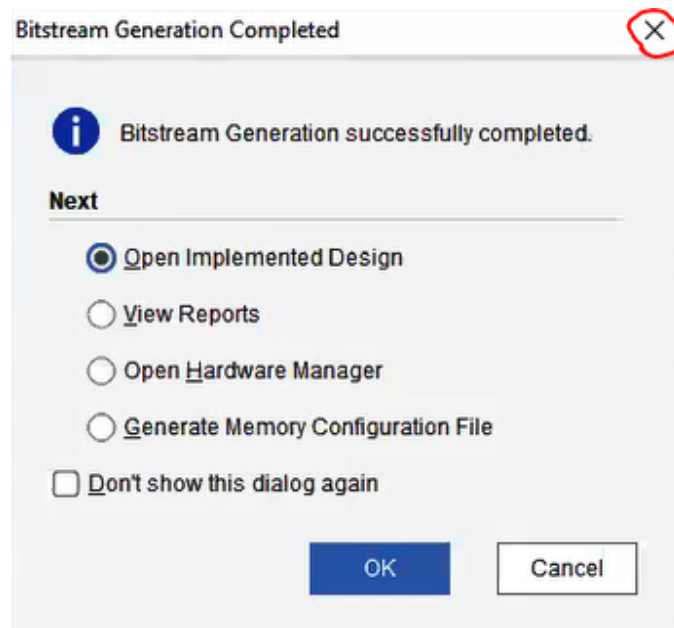


**Right click** again on **system.bd**, and select **Create HDL Wrapper...** to generate the top-level Verilog/VHDL model. Leave the **Let Vivado manager wrapper and auto-update** option selected, and click **OK**. The `system_wrapper.v` will be created and added to the project.

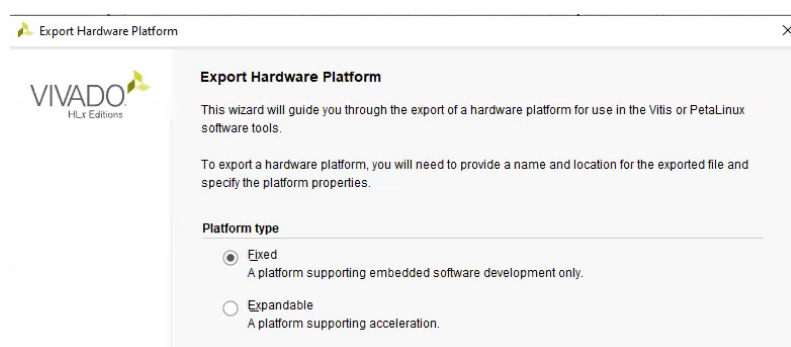


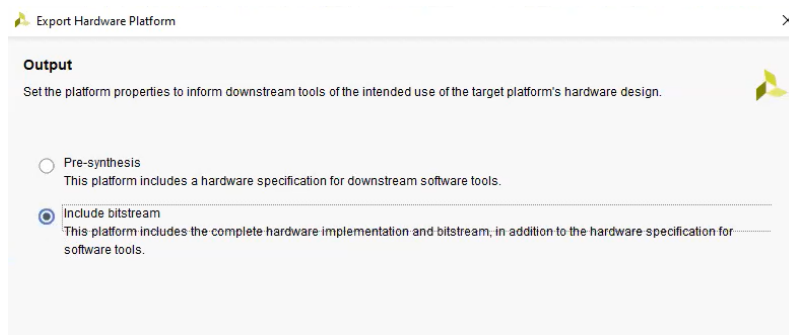
Click the **Generate Bitstream** in the Flow Navigator panel to generate the bitstream. Click **Yes**. Then leave all the settings unchanged, and click **OK**. Pls wait until a **Bitstream Generation Completed** dialog box is displayed. Then **close** the dialog box of **Bitstream Generation Completed**.

- ▼ IMPLEMENTATION
  - ▶ Run Implementation
  - > Open Implemented Design
- ▼ PROGRAM AND DEBUG
  - Generate Bitstream
  - > Open Hardware Manager



6. Select **File > Export > Export hardware**, select **Include bitstream** and Leave all the settings unchanged.





The dialog box is titled "Export Hardware Platform" with a close button (X) in the top right corner. Below the title bar, there is a section labeled "Output" with a small green icon to its right. The text below "Output" reads: "Set the platform properties to inform downstream tools of the intended use of the target platform's hardware design." There are two radio button options: "Pre-synthesis" (unselected) and "Include bitstream" (selected). The "Pre-synthesis" option has a description: "This platform includes a hardware specification for downstream software tools." The "Include bitstream" option has a description: "This platform includes the complete hardware implementation and bitstream, in addition to the hardware specification for software tools."

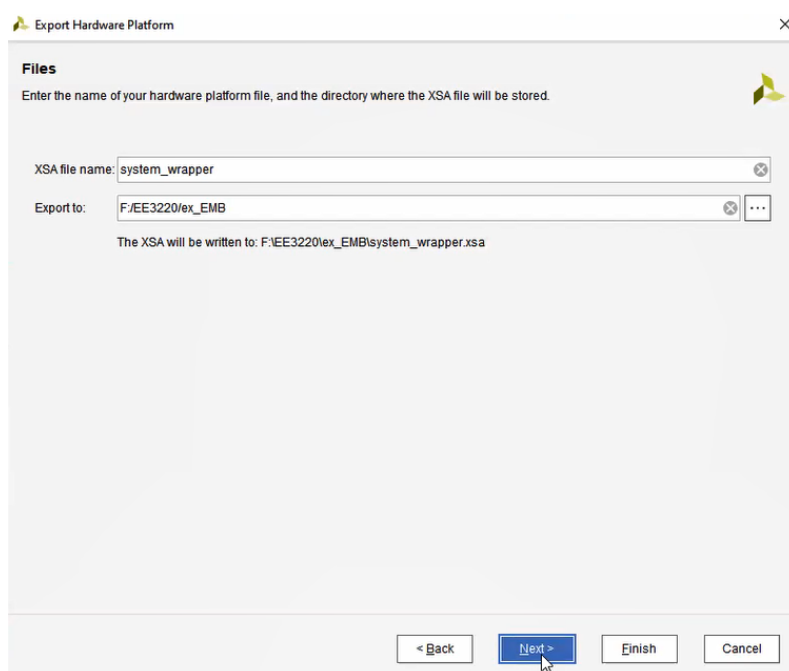
**Output**

Set the platform properties to inform downstream tools of the intended use of the target platform's hardware design.

☐ Pre-synthesis  
This platform includes a hardware specification for downstream software tools.

☒ Include bitstream  
This platform includes the complete hardware implementation and bitstream, in addition to the hardware specification for software tools.

Then click **Next**.



The dialog box is titled "Export Hardware Platform" with a close button (X) in the top right corner. Below the title bar, there is a section labeled "Files" with a small green icon to its right. The text below "Files" reads: "Enter the name of your hardware platform file, and the directory where the XSA file will be stored." There are two text input fields: "XSA file name:" with the value "system\_wrapper" and "Export to:" with the value "F:/EE3220/lex\_EMB". Both fields have a small 'x' icon to their right. Below the "Export to:" field, there is a text label: "The XSA will be written to: F:/EE3220/lex\_EMB/system\_wrapper.xsa". At the bottom of the dialog, there are four buttons: "< Back", "Next >" (highlighted with a mouse cursor), "Finish", and "Cancel".

**Files**

Enter the name of your hardware platform file, and the directory where the XSA file will be stored.

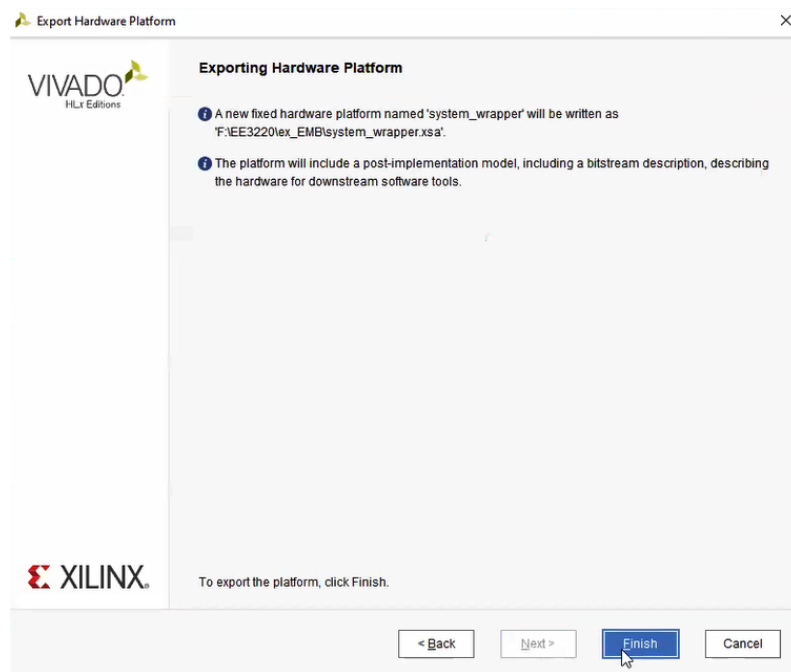
XSA file name:

Export to:

The XSA will be written to: F:/EE3220/lex\_EMB/system\_wrapper.xsa

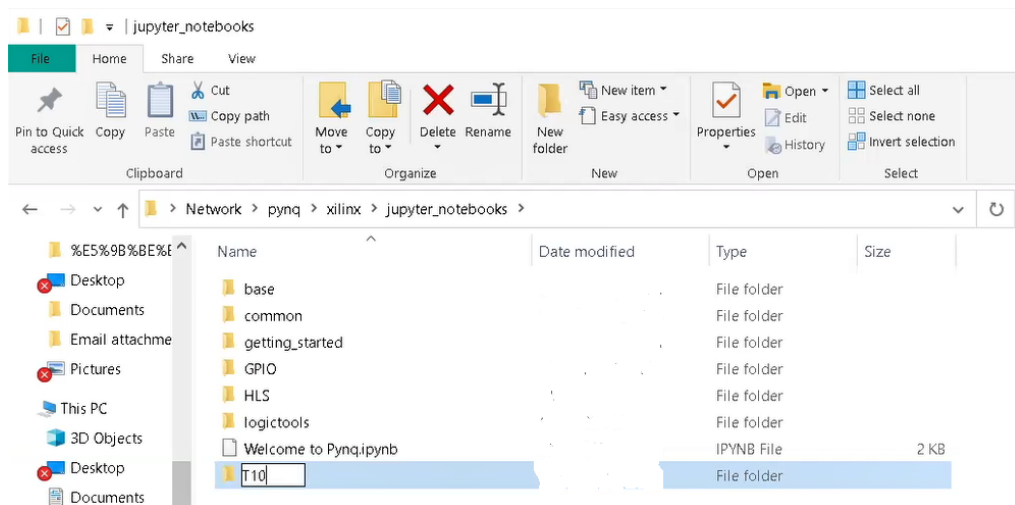
< Back   **Next >**   Finish   Cancel

Click **Finish**.

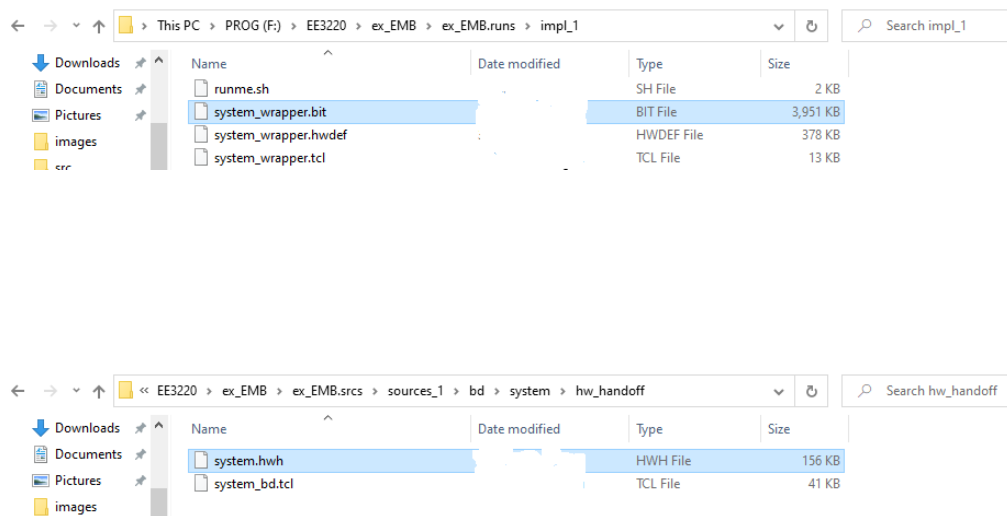


## B. Test your hardware system on Pynq board

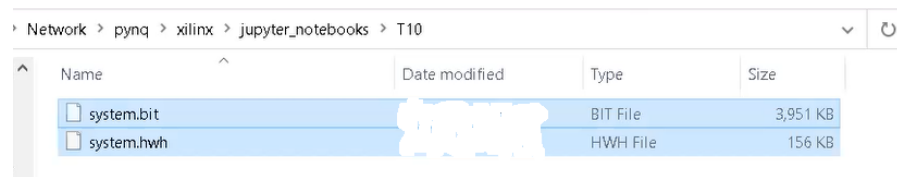
1. Connect the board and your computer. Connect **Ethernet cable** and **USB cable** to you computer, insert **SD card** into the board, set static IP of the computer, and power on the board. You can refer to **Tutorial 8, Section 5** for details.
2. Create a new folder named **T10** under **jupyter\_notebooks** folder, we will put overlay files and software code under this folder.



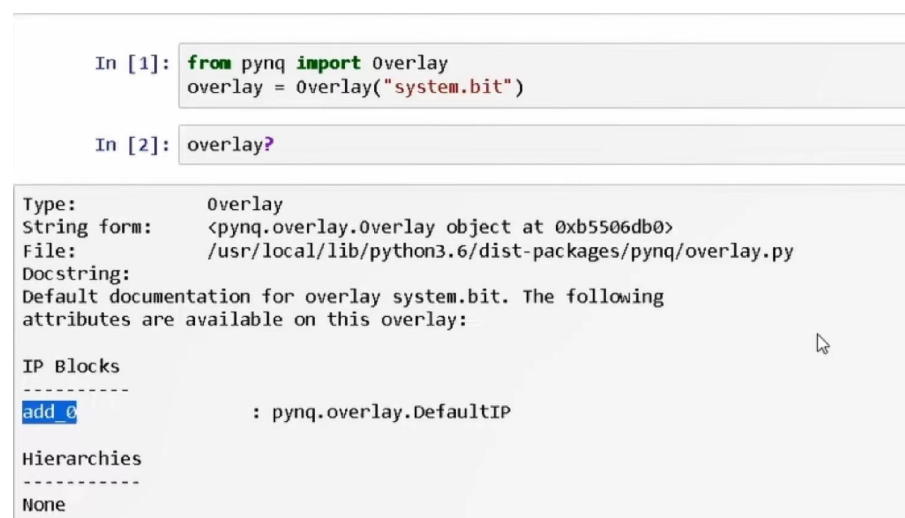
3. Search the bitstream file **system\_wrapper.bit** and another **system.hwh** file. For example, our file locations of **system\_wrapper.bit** file and **system.hwh** file are shown in the following figures:



4. Rename **system\_wrapper.bit** to be **system.bit**. Then copy your design **system.hwh** and **system.bit** into the **T10** folder.



5. The default static IP of PYNQ-Z2 is 192.168.2.99. Open browser and input <http://pynq:9090>, password is xilinx, you will open the Jupyter notebook of the board. Then create a new Python3 notebook under **T10** folder. We configure the FPGA with Overlay first. We can find an **add\_0** block in our design.



6. We can expose the **register map** associated with the IP. Find the input **a** and **b**, the output **c**, and the control signal **CTRL**.



```
In [3]: add_ip = overlay.add_0
```

```
In [4]: add_ip.register_map
```

```
Out[4]: RegisterMap {
  CTRL = Register(AP_START=0, AP_DONE=0, AP_IDLE=1, AP_READY=0, RESERVED_1=0, AUTO_RESTART=0, RESERVED_2=0),
  GIER = Register(Enable=0, RESERVED=0),
  IP_IER = Register(CHAN0_INT_EN=0, CHAN1_INT_EN=0, RESERVED=0),
  IP_ISR = Register(CHAN0_INT_ST=0, CHAN1_INT_ST=0, RESERVED=0),
  a = Register(a=0),
  b = Register(b=0),
  c = Register(c=0),
  c_ctrl = Register(c_ap_vld=0, RESERVED=0)
}
```

7. We can **interact** with the IP using the **register map** directly.

```
In [5]: add_ip.register_map.a = 4
        add_ip.register_map.b = 3
        add_ip.register_map.CTRL.AP_START = 1
```

```
In [6]: add_ip.register_map.c
```

```
Out[6]: Register(c=7)
```

8. Alternatively by reading the **driver source code** generated by HLS, we can determine **offsets** that we need write and read. In my project, the **offset** of **a**, **b**, and **c** are **0x10**, **0x18**, and **0x20**, respectively.

```
add_ip.write(0x10, 6)
add_ip.write(0x18, 3)
add_ip.write(0x00, 1)
add_ip.read(0x20)
```

9

9. **(Optional)** Furthermore, we can create a more user-friendly driver for other users to easily develop software programs with your hardware system. This step requires that you are more familiar with Python, so it is an optional step.

```
In [11]: from pynq import DefaultIP

class AddDriver(DefaultIP):
    def __init__(self, description):
        super().__init__(description=description)

    bindto = ['xilinx.com:hls:add:1.0']

    def add(self, a, b):
        self.write(0x10, a)
        self.write(0x18, b)
        self.write(0x00, 1)
        return self.read(0x20)

overlay = Overlay("system.bit")
```

```
In [12]: scalar_adder = overlay.add_0
        scalar_adder.add(10, 15)
```

```
Out[12]: 25
```

End