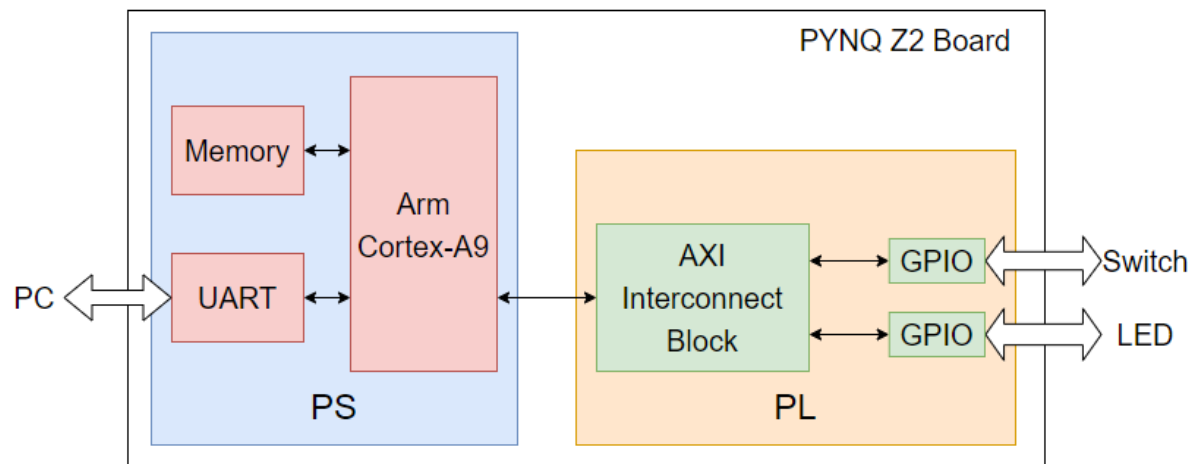


Tutorial 8: Use Vivado to build an System-on-Chip

Please go to <https://github.com/xupgit/Zynq-Design-using-Vivado> for more details.

Introduction

In this tutorial, we would learn how to use PYNQ-Z2 FPGA board to build an SoC. We would focus on two issues: using IP Integrator to create a processing system (or named PS), and extending the processing system by adding two GPIO (General Purpose Input/Output) IPs. The block diagram of hardware design in this tutorial is show in the Figure below.



Objectives

After completing this lab, you will be able to:

- Use the IP Integrator to create a hardware system
- Configure the GP Master port of the PS to connect to IP in the PL
- Boot Linux system on PYNQ-Z2 board
- Configure FPGA with your hardware design and test the hardware design by python

Tools

The Tools required for this tutorial includes:

- Software: Xilinx Vivado
- Hardware: PYNQ-Z2 FPGA board, SD card

Students could complete this tutorial with remotely logging into laboratory. You can also download the Xilinx Vivado <https://www.xilinx.com/support/download.html> on your PC if you are interested.

Steps

1. Create a Vivado Project

1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado**
2. Click **Create New Project** to start the wizard. You will see the Create a New Vivado Project dialog box. Click **Next**.
3. Click the Browse button of the *Project location* field of the **New Project** form, browse to a suitable location, and click **Select**.
4. Enter **ex_ps** in the Project Name field. Make sure that the Create Project Subdirectory box is checked. Click **Next**.
5. In the Project Type form select **RTL Project**, and click Next
6. In the Add Sources form, select **Verilog** as the Target language and **Mixed** as the Simulator language, and click Next
7. Click Next two more times to skip **Add Sources** and **Add Constraints**
8. In the *Default Part* form, use the **Parts** option and various drop-down fields of the **Filter** section. Select the **XC7Z020clg400-1**.

New Project

Default Part
Choose a default Xilinx part or board for your project. This can be changed later.

Parts | Boards

[Reset All Filters](#)

Category: All Package: clg400 Temperature: All Remaining
Family: Zynq-7000 Speed: -1

Search: (8 matches)

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gt
xc7z020clg400-1	400	125	53200	106400	140	0	220	0

[? < Back](#) [Next >](#) [Finish](#) [Cancel](#)

Part Selection for the PYNQ

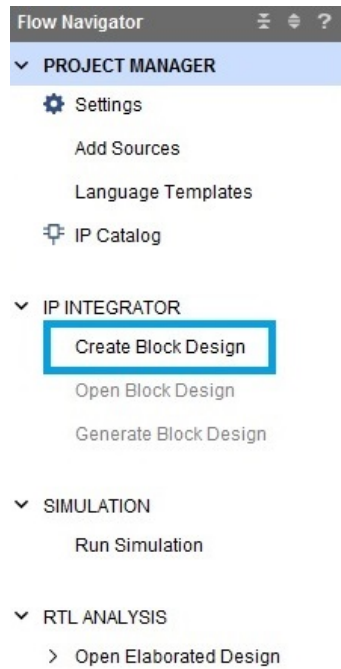
You may also select the **Boards** option, select www.digilentinc.com for the PYNQ-Z1 board, tul.com.tw for the PYNQ-Z2 board under the Vendor filter and select the appropriate board. Notice that PYNQ-Z1 and PYNQ-Z1 may not be listed as they are not in the tools database. If not listed then you can download the board files for the desired boards either from Digilent PYNQ-Z1

webpage or TUL PYNQ-Z2 webpage. Click **Next**.

9. Click **Finish** to create the Vivado project.

2. Creating the System Using the IP Integrator

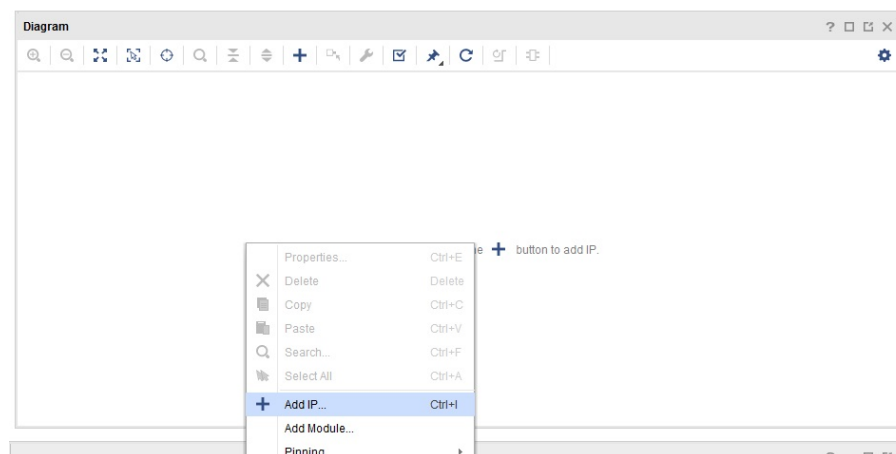
1. In the Flow Navigator, click **Create Block Design** under IP Integrator



Create IP Integrator Block Diagram

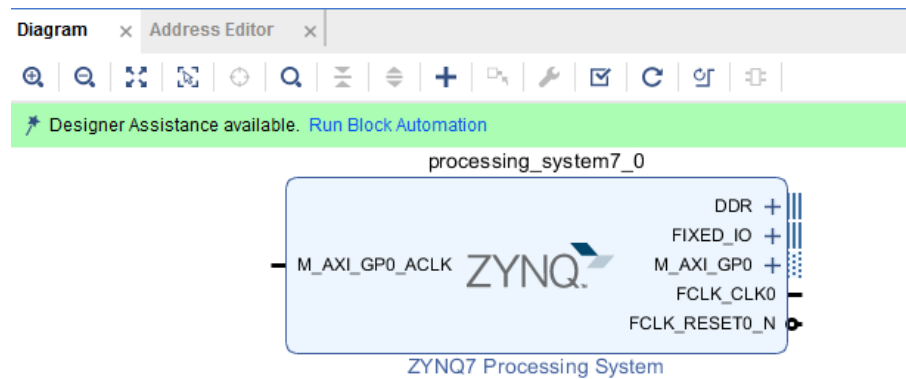
2. Enter **system** for the design name and click OK

3. Right-click anywhere in the Diagram workspace and select **Add IP**.



Add IP to Block Diagram

- Once the **IP Catalog** opens, type “zynq” into the Search bar, find and double click on **ZYNQ7 Processing System** entry, or click on the entry and hit the Enter key to add it to the design.



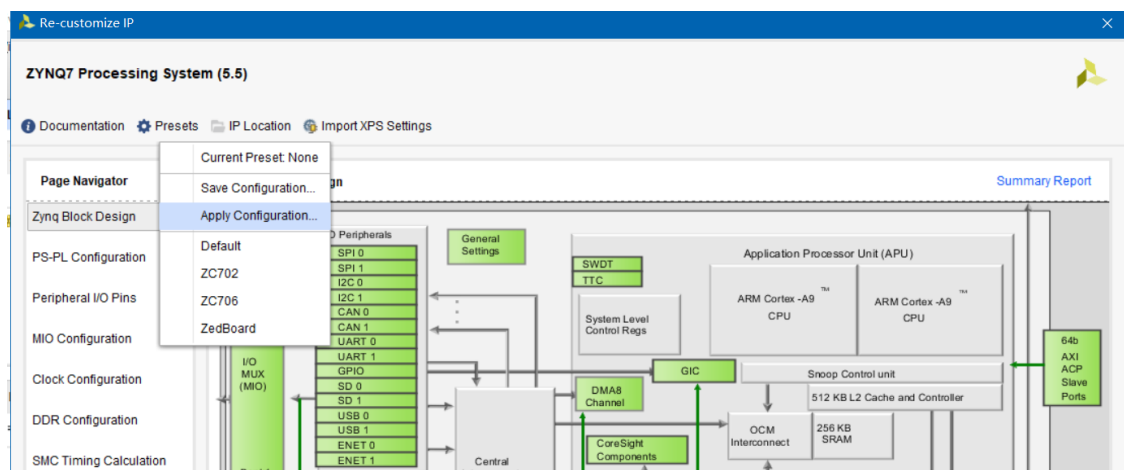
Zynq 7 Processing System

3. Configure the processing block

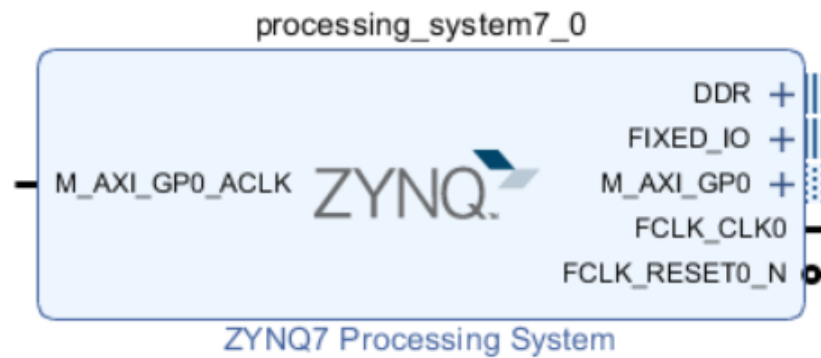
- Double-click on the added block to open its **Customization** window. Click **Preset** and **Apply Configuration**, select **pynq_z2.tcl** file and click OK.

The **Tool Command Language (Tcl)** is the scripting language integrated in the Vivado tool environment. Tcl lets you perform interactive queries to design tools in addition to executing automated scripts. Tcl is a standard language in the semiconductor industry for application programming interfaces. In this tutorial, the IP configuration is saved as Tcl file, and we could use the Tcl file to re-store our configuration efficiently.

Notice now the Customization window shows selected peripherals (with tick marks). You can configure the processing block later with more experience.

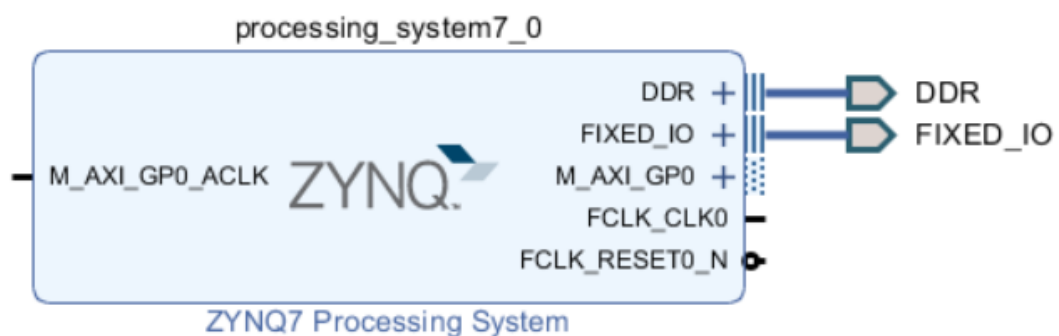


Processing System Customization



Customized Zynq Block

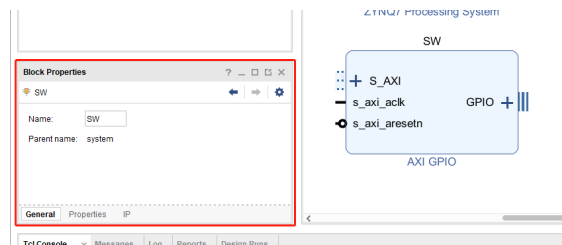
2. Notice the message at the top of the Diagram window in a green label saying that Designer Assistance available. Click **Run Block Automation**.
3. A new window pops up called the Run Block Automation window. In it, select **processing_system7_0**, leave the default settings and click OK
4. Once Block Automation has been complete, notice that ports have been automatically added for the **DDR** and **Fixed IO**, and some additional ports are now visible. The imported configuration for the Zynq related to the board has been applied which will now be modified. The block should finally look like this:



Zynq Block with DDR and Fixed IO ports

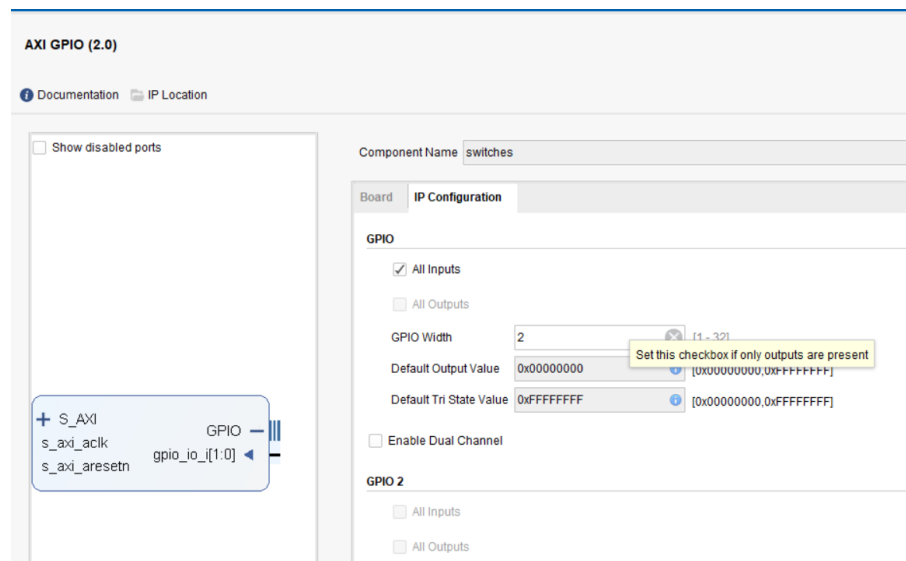
3. Add Two Instances of GPIO

1. Next add an IP by **right clicking on the Diagram window> Add IP** and search for AXI GPIO in the catalog
2. Double-click the *AXI GPIO* to add the core to the design. The core will be added to the design and the block diagram will be updated.
3. Click on the AXI GPIO block to select it, and in the properties tab, change the name to **SW**



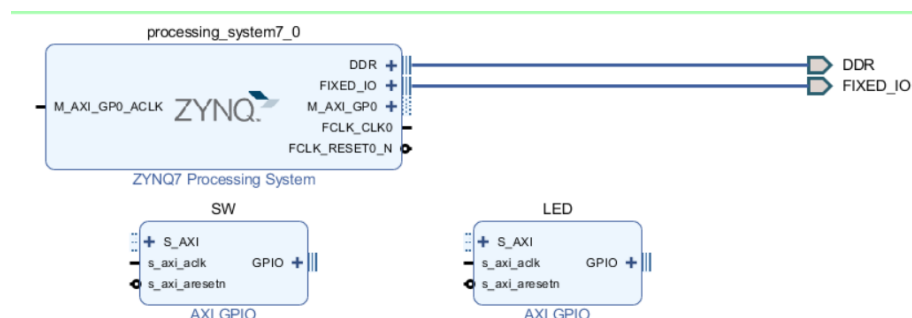
Change AXI GPIO default name

- Double click on the AXI GPIO block to open the customization window. Select **2-bit input** shown as below



Configure 2-bit input GPIO as switches

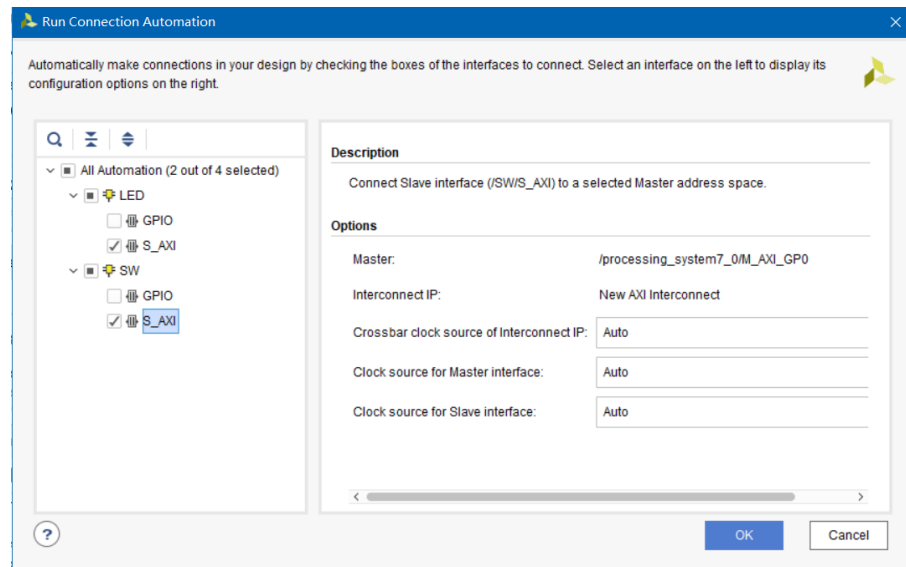
- Similarly, add another GPIO and change the name to **LED**, configure the IP as **4-bit output**



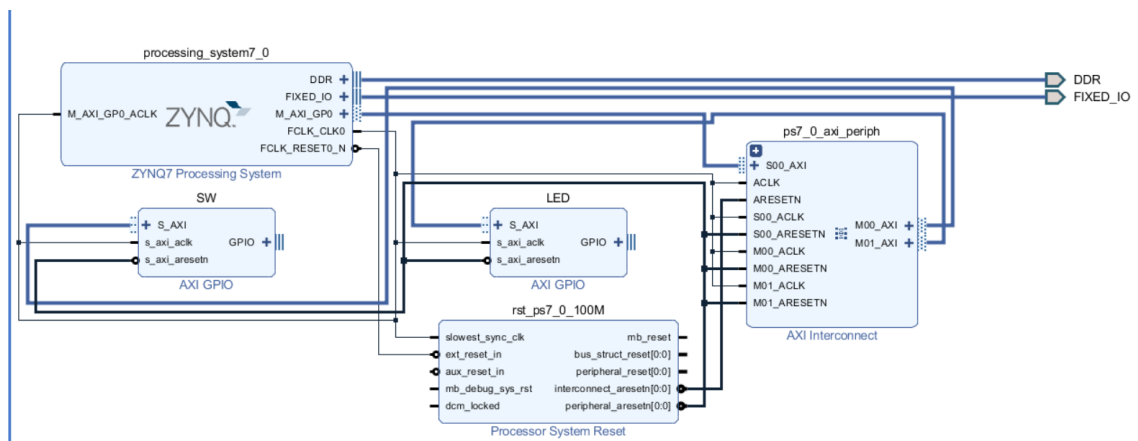
Add another GPIO IP named LED

3. Build the system

1. You can find a **Run connection automation**, click it. In the new window, select **S_AXI** and then click **OK**

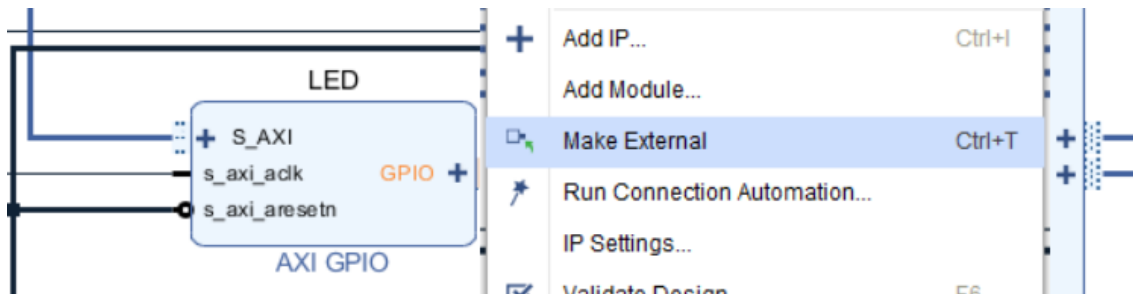


Run connection automation



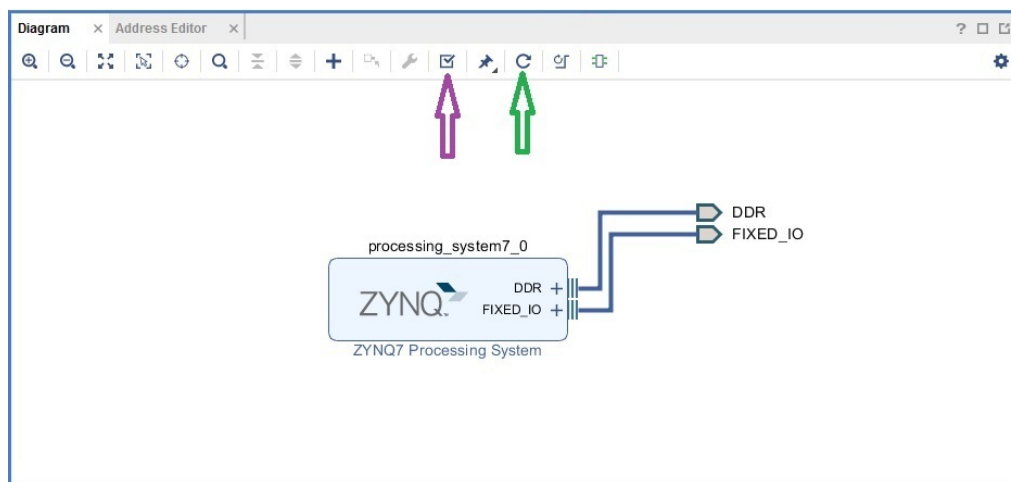
You will find the similar block design like this

2. Select GPIO port in SW, right click and click **Make External**, do the same thing for GPIO port in the LED block



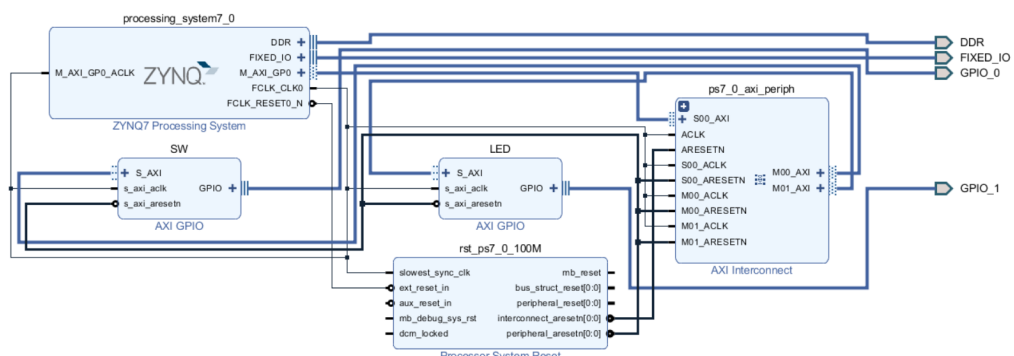
Make External for two GPIO ports

- Click on the Regenerate Layout button (green arrow) shown below, then **ctrl+S** to save the block design.



Regenerating and Validating Design

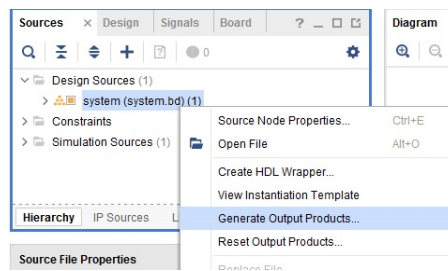
- Click on the Validate Design button (purple arrow) and make sure that there are no errors.



You may get the block design like this

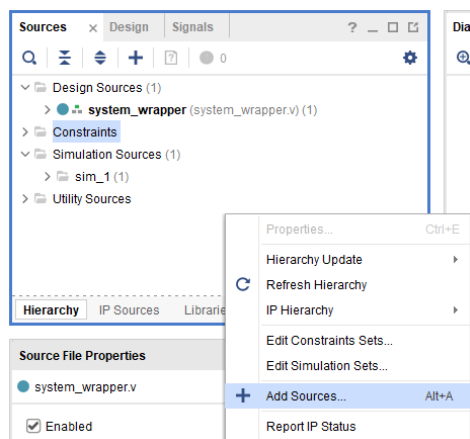
4. Generate Top-Level and bitstream

1. 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 (You can also click on **Generate Block Design** in the Flow Navigator pane to do the same)



Generating output products

2. 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/system_wrapper.vhd file will be created and added to the project. Double-click on the file to see the content in the Auxiliary pane.
3. Right-click on the Source panel, click **Add Sources...** . Select **add or create the constraints**. Add constraint file **pynq_z2.xdc**, in case your port name is not the same as the xdc file, you may do minor modifications. Click **Finish**.

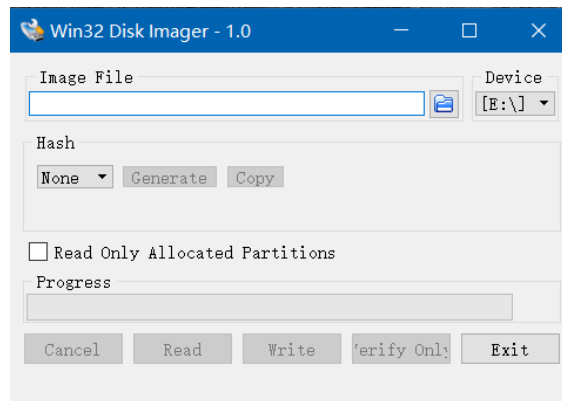


Win32DiskImager

4. Click the **Generate Bitstream** in the Flow Navigator panel to generate the bitstream. Leave all the settings unchanged.
5. Select **File > Export > Export hardware**, select **Include bitstream** and click **OK**. (Save the project if prompted)

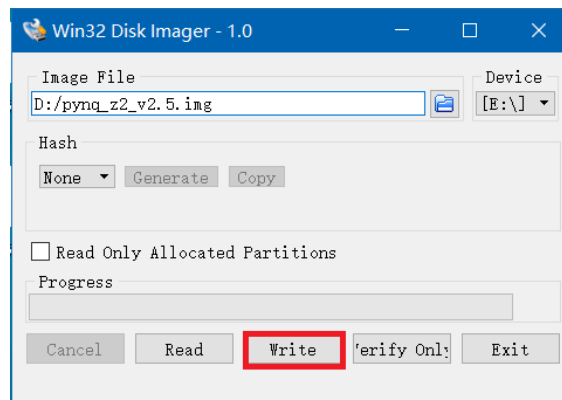
5. Load System Image to an SD Card and Boot Linux

1. In case you only have a bland card, the first step is to load the image to the Micro-SD card.
Download Win32DiskImager (for Windows): <https://sourceforge.net/projects/win32diskimage/>.





Win32DiskImager

2. Select pynq system image and card device, click **write** to load image to the SD card.



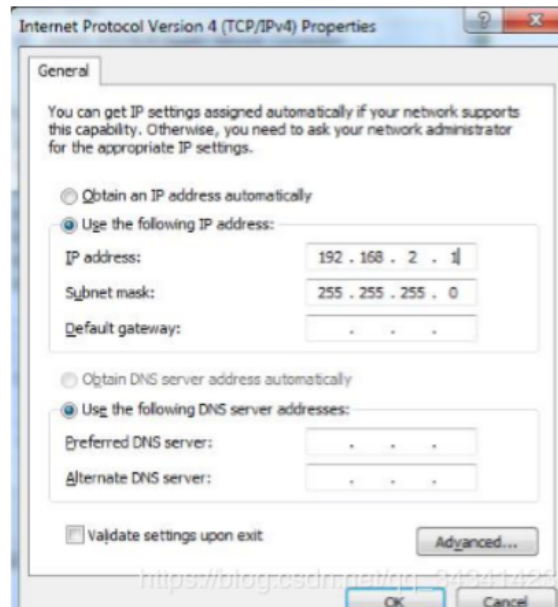
Load system image

3. The SD card has two partitions, one boot partition including **BOOT.bin** and **image.ub**, another is the linux file system.

 BOOT.BIN	2019/10/3 17:35	BIN 文件	642 KB
 image.ub	2019/10/3 17:35	UB 文件	4,660 KB

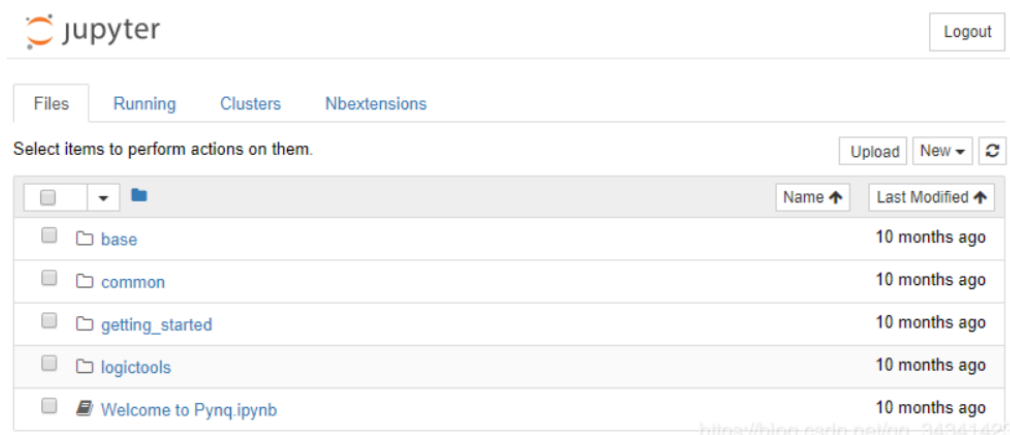
/boot partition

4. Insert SD card into the board, connect USB cable and Ethernet cable to your computer. Set **static IP** of the computer as follows



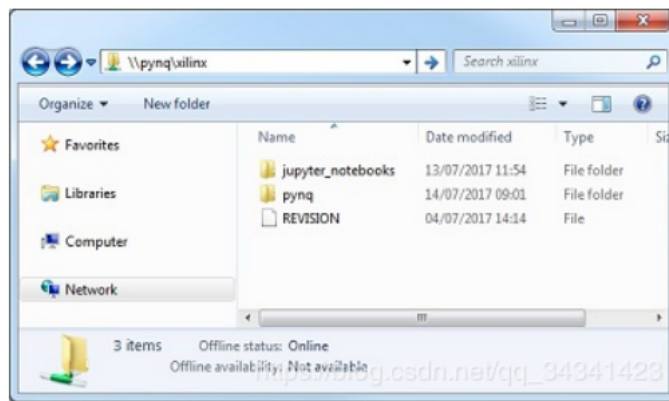
Set static IP

5. Power on the board, 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. Please google Jupyter for more information



Open Jupyter Notebook

6. You can transfer files between board and your computer with samba protocol, open file manager, input `\\pynq\xilinx` to mount disk, the username and password are both **xilinx**



Mount disk in the file manager

7. Alternatively, you can open UART connection with PuTTY or MobaXterm.

```
COM6 - PuTTY
xilinx@pyng:~$ ifconfig
eth0: flags=4096<UP,BROADCAST,MULTICAST> mtu 1500
    inet6 fe80::205:6bff:fe00:c787 prefixlen 64 scopeid 0x20<link>
    ether 00:05:6b:00:c7:87 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10 bytes 1465 (1.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 29 base 0xb000

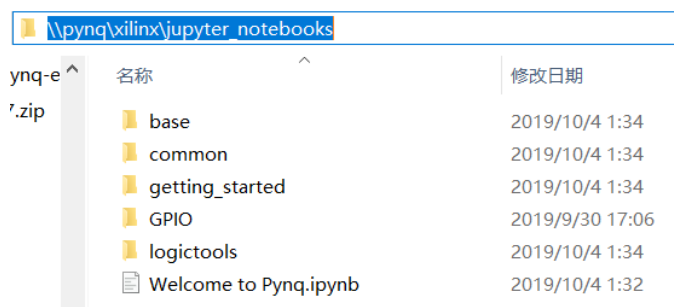
eth0:1: flags=4096<UP,BROADCAST,MULTICAST> mtu 1500
    inet 192.168.2.99 netmask 255.255.255.0 broadcast 192.168.2.255
    ether 00:05:6b:00:c7:87 txqueuelen 1000 (Ethernet)
    device interrupt 29 base 0xb000
```

Mount disk in the file manager

6. Test hardware design

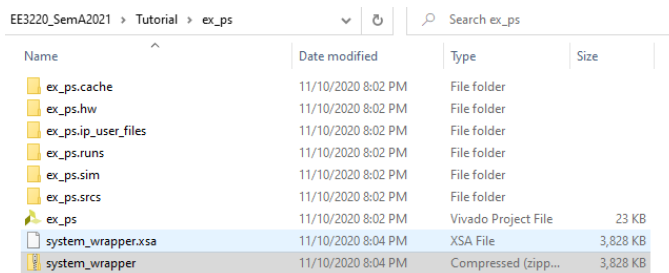
In the last exercise we design a simple embedded system with two GPIO IPs, now we are going to write some software code to test the hardware design. The software can read current switch status and write values to turn on and turn off the led lights.

1. In the file manager, create a new folder named **GPIO** under **jupyter_notebooks**, we will put overlay files and software code under this folder.



Create a new folder

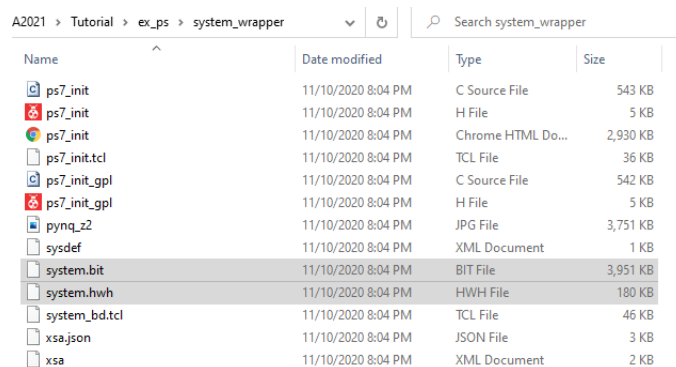
2. As we have exported hardware design in the previous exercise, find the hardware design **.xsa** file in the project **ex_ps\system_wrapper.xsa**, rename the **.xsa** file to **system_wrapper.zip** and unzip it.



Name	Date modified	Type	Size
ex_ps.cache	11/10/2020 8:02 PM	File folder	
ex_ps.hw	11/10/2020 8:02 PM	File folder	
ex_ps.ip_user_files	11/10/2020 8:02 PM	File folder	
ex_ps.runs	11/10/2020 8:02 PM	File folder	
ex_ps.sim	11/10/2020 8:02 PM	File folder	
ex_ps.srcs	11/10/2020 8:02 PM	File folder	
ex_ps	11/10/2020 8:02 PM	Vivado Project File	23 KB
system_wrapper.xsa	11/10/2020 8:04 PM	XSA File	3,828 KB
system_wrapper	11/10/2020 8:04 PM	Compressed (zipp...	3,828 KB

Unzip the hardware design file

3. In **system_wrapper** folder, you will find a bitstream file **system_wrapper.bit** and another **system.hwh** file, rename **system_wrapper.bit** to be **system.bit** (make sure the name is the same as **system.hwh**).



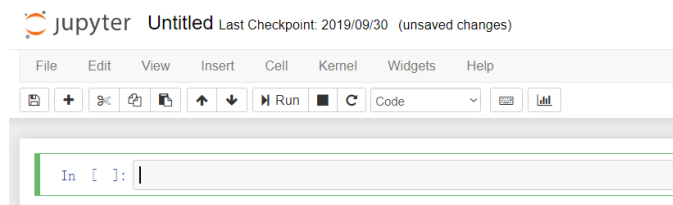
Name	Date modified	Type	Size
ps7_init	11/10/2020 8:04 PM	C Source File	543 KB
ps7_init	11/10/2020 8:04 PM	H File	5 KB
ps7_init	11/10/2020 8:04 PM	Chrome HTML Do...	2,930 KB
ps7_init.tcl	11/10/2020 8:04 PM	TCL File	36 KB
ps7_init_gpl	11/10/2020 8:04 PM	C Source File	542 KB
ps7_init_gpl	11/10/2020 8:04 PM	H File	5 KB
pyng_z2	11/10/2020 8:04 PM	JPG File	3,751 KB
sysdef	11/10/2020 8:04 PM	XML Document	1 KB
system.bit	11/10/2020 8:04 PM	BIT File	3,951 KB
system.hwh	11/10/2020 8:04 PM	HWH File	180 KB
system_bd.tcl	11/10/2020 8:04 PM	TCL File	46 KB
xsa.json	11/10/2020 8:04 PM	JSON File	3 KB
xsa	11/10/2020 8:04 PM	XML Document	2 KB

Get your hardware design files

4. Copy two files **system.bit** and **system.hwh** to the created **GPIO** folder.
5. Open Jupyter notebook in the browser, open GPIO folder, you will see two files in the folder, create a python3 notebook. It will pop up a new window for coding.



Create a new python3 notebook



Python3 coding environment in Jupyter

1. Write Python code shown in the follows, it will read the current switch values, the value is **11** in binary code.

```
In [1]: from pynq import Overlay
        overlay = Overlay("system.bit")
        sw=overlay.axi_gpio_0
        sw.read()

Out[1]: 3
```

Read switch status

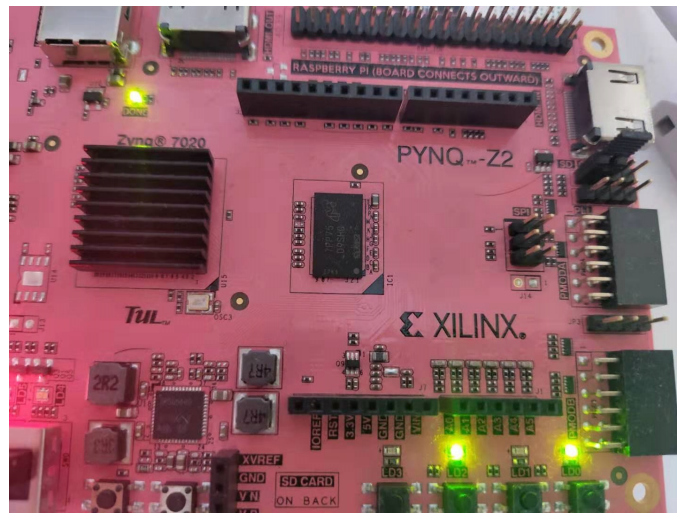
2. Similarly, you can write code to turn on the leds. The following code write **0101** to the leds.

```
In [1]: from pynq import Overlay
        overlay = Overlay("system.bit")
        sw=overlay.axi_gpio_0
        sw.read()

Out[1]: 3

In [2]: led=overlay.axi_gpio_1
        led.write(0, 0x5)
```

Turn on Leds



Turn on Leds

End