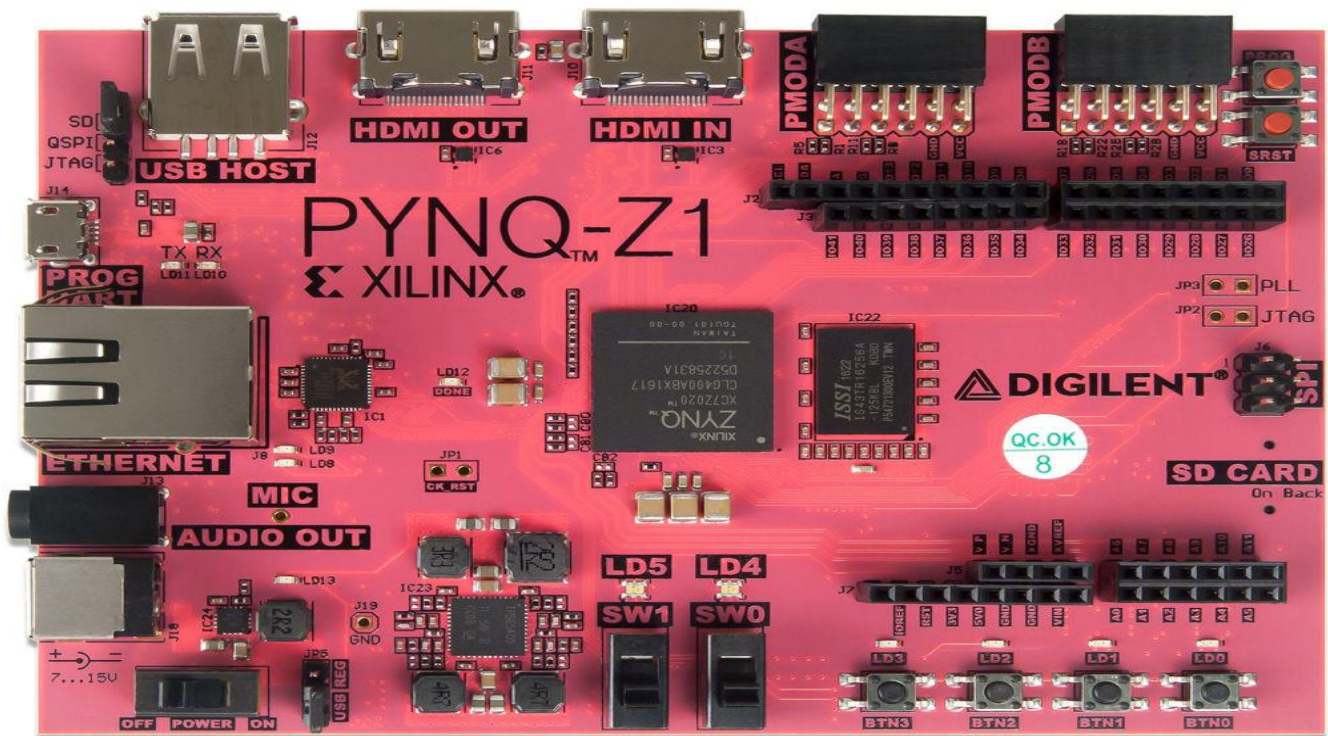


This manual applies to PYNQ Z1 Board image 2.6 By Heba Saleh



Contents

1.	Introduction	1
2.	Getting Started	2
2.1	MicroSD Card Set-up Considerations	2
2.2	Set-up the PYNQ Board	3
2.3	Connecting PYNQ Board	4
2.4	System Organization & Navigation	6
2.5	Accessing the System	7
3.	Using the System	8
3.1	Boot mode selection	9
3.2	Clock Sources	9
3.3	HDMI	10
3.4	Reset Sources	10
4.	Xilinx Vivado integrated into PYNQ	12
4.1	Create a custom IP	12
4.2	Design overlay in vivado	14
4.3	Test IP in Python	16
4.4	Create a Driver	17
	Appendix : Useful links	21

1. Introduction

PYNQ is an open-source project that aims to work on any computing platform and operating system. This goal is achieved by adopting a web-based architecture, which is also browser agnostic. It incorporates the open source Jupyter notebook infrastructure to run an Interactive Python (IPython) kernel and a web server directly on the ARM processor of the Zynq device. The web server brokers access to the kernel via a suite of browse. The IPython kernel and packages are installed with base hardware library and API for the FPGA. This manual details the setup instruction for the board as well as explains some of the board overlay. In addition, it covers an example to demonstrate how to integrate Vivado HLS into the PYNQ Board.

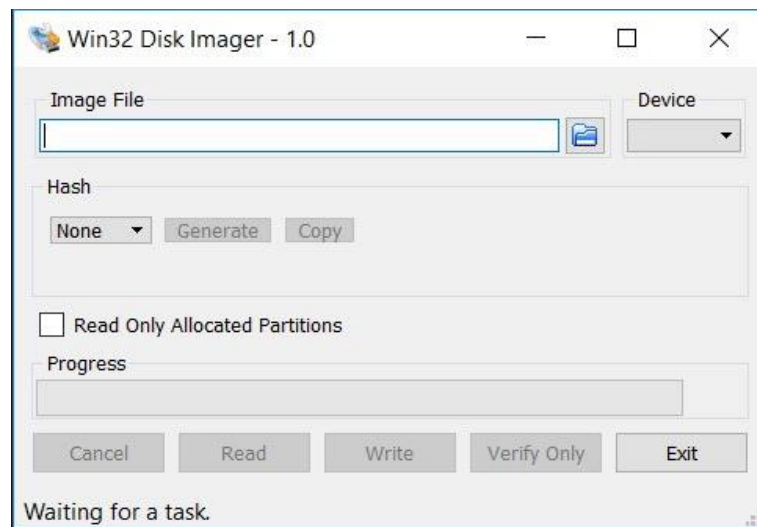
2. Getting Started

This guide details how to set up the PYNQ Board and computer to be ready.

2.1 MicroSD Card Set-up Considerations

Setting up the development board and configuring the user network are required before programming the PYNQ board.

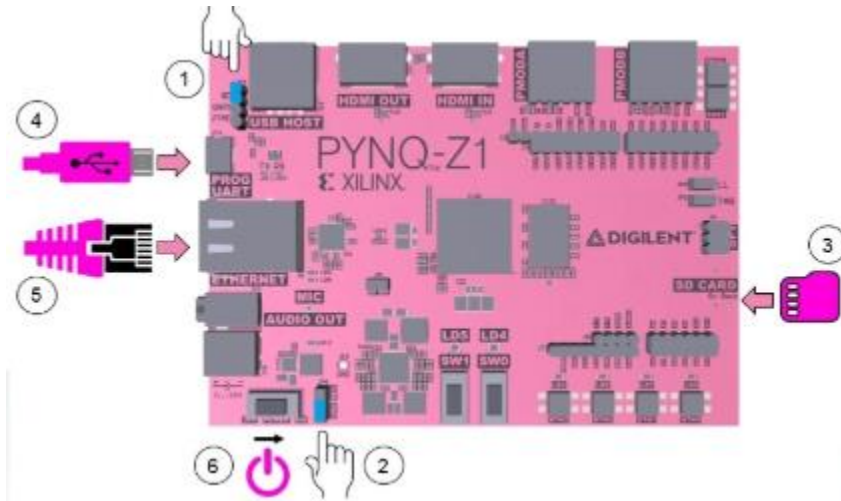
1. You would need a MicroSD card (minimum 8GB recommended) to download the appropriate PYNQ image (suitable for your code) on it.
2. Download the pre – compiled image on your computer. It can be found in the link (<http://www.pynq.io/board.html>). The image includes the board files, and overlays examples and Jupyter notebooks.
3. Unzip the image and write the image to a blank Micro SD card using **Win32 Disk Imager** as show in the figure below. Upload the image file and choose the right device (the SD card) then click write.



Due to the instability of the PYNQ Board, you might have to rewrite the MicroSD whenever the board freezes. Thus, always backup your work.

2.2 Set-up the PYNQ Board

The initial board setup steps below are taken directly from the [PYNQ documentation](#), we decided to copy them here as we have some small changes in our workflow.



1. Set the **JP4 / Boot** jumper to the SD position by placing the jumper over the top two pins of JP4 as shown in the image. (This sets the board to boot from the microSD card)
2. To power the PYNQ-Z1 from the Micro-USB cable, set the **JP5 / Power** jumper to the **USB** position. (You can also power the board from an external 12V power regulator by setting the jumper to **REG**.)
3. Insert the microSD card loaded with the PYNQ image into the **SD CARD** slot underneath the board.
4. Connect the USB cable to your computer, and to the **PROG - UART / J14** Micro-USB port on the board.
5. Turn on the PYNQ-Z1 and check the boot sequence

The red **LD13** LED will come on immediately to confirm that the board has power. And if the MicroSD Card is set up correctly then the boot sequence will continue as following. After a few seconds, the yellow/green **LD12 / Done** LED will light up to show that the Zynq® device is operational.

After a minute you should see two blue **LD4 & LD5** LEDs and four yellow/green **LD0-LD3** LEDs flash simultaneously. The blue **LD4-LD5** LEDs will then turn on and off while the yellow/green **LD0-LD3** LEDs remain on. The system is now booted and ready for use.

2.3 Connecting PYNQ Board

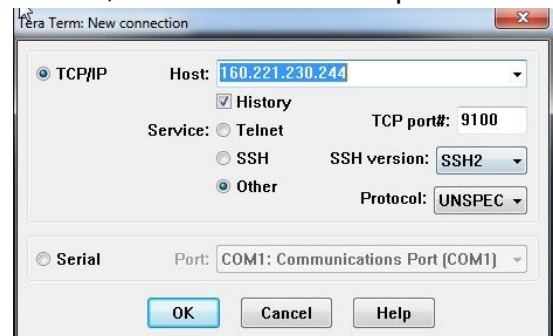
Connecting the board after setting up the board could be achieved in two ways, either via UART using the USB cable or via SSH over an Ethernet cable.

2.3.1 UART

UART allows a serial communication protocol between the board and the host computer over the USB. It is used to have a direct access to the PYNQ hardware boot menu options, and to the operating system installed on the microSD card. Thus, the following steps should be followed.

1. Tera Term is used as a terminal emulator.
2. Determine the COM port the board is using. That could be done from the **Device Manager** expand the **Ports** group. There, several USB serial ports are listed showing which ports are in use.
3. press the **serial** option and select the **COM port** associated with your board, then click **ok**
4. Then setup the serial communication from the **setup** option in the menu bar. The settings are as following:

- 115200 BAUD
- 8 data bits
- 1 stop bit
- no parity
- no flow control (neither hardware nor software)



Upon a successful connection to the board, you will be presented with the OS login. By default, the PYNQ SD image is configured to use the **xilinx** user, with **xilinx** as the password. From there you can access the board Ethernet configuration and board IP.

2.3.2 SSH

Using the Ethernet cable allows the board to access the network configuration. It is important to note that the IP addresses of the computer and the board must be in same subnet to have an internet access. Typically, this means that the first three numbers of the board's IP address must match that of your computer's IP.

1. First, you need to find your computer IP address by opening the **command prompt** (Win+R), type **cmd** then run the "ipconfig" command.
2. Once you find your computer IP, it is always better to check if the board Ethernet IP matched to the computer IP. Therefore, access the board hardware from the UART. Run the "ifconfig" command in the **terminal** for

the board interface configuration, and check the **eth0** in the **inet addr** (board's IP).

```
eth0      Link encap:Ethernet  HWaddr 00:0C:29:17:1B:27
          inet addr:192.168.208.133  Bcast:192.168.208.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe17:1b27/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:195 errors:0 dropped:0 overruns:0 frame:0
          TX packets:189 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:21313 (20.8 Kb)  TX bytes:16778 (16.3 Kb)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:18 errors:0 dropped:0 overruns:0 frame:0
          TX packets:18 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1060 (1.0 Kb)  TX bytes:1060 (1.0 Kb)
```

3. However, if by any chance they did not match you can assign the Board a Static IP, by following the steps below:
 - a. Open the board's UART terminal and edit the eth0 file within the interfaces.d folder using this command
(`sudo vi /etc/network/interfaces.d/eth0`)
 - b. Inside the eth0 file you will find a similar entry as shown below with the default board static IP.

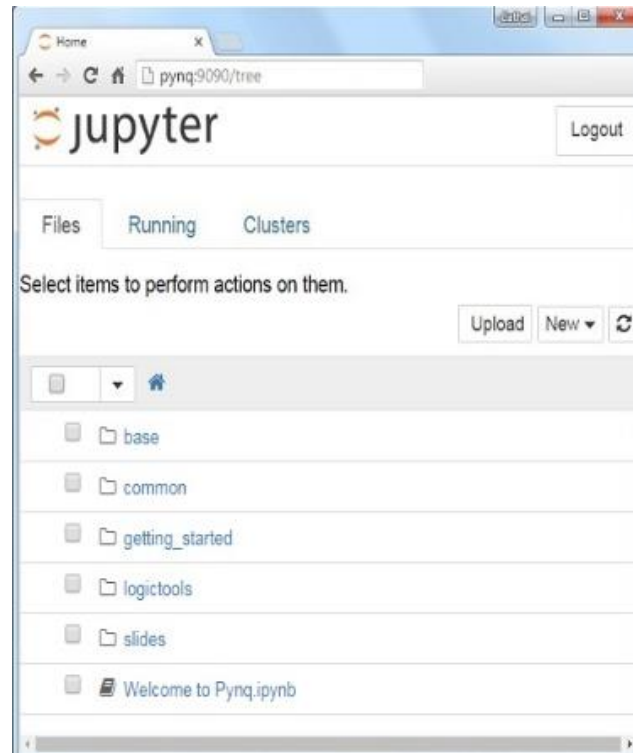
```
auto eth0:0
        iface eth0:0 inet static
        address 10.162.177.99
        netmask 255.255.255.0
```
 - c. Change the address entry to match your computer's IP subnet.
 - d. Connect the board to the Ethernet cable
 - e. Restart the interface with the command (`sudo ifdown eth0:0`) followed by (`sudo ifup eth0:0`)
 - f. Confirm this worked by running `ifconfig | grep "inet addr"`.
4. After having both IPs' subnet matching, give your computer a Static IP. Do the following:
 - a. Right click on your **internet access** and click on **Open Network & Internet settings**.
 - b. Under the change your network settings, click **Change adapter settings**. This should present you with the list of network interfaces available to the Windows operating system.
 - c. Look for the network interface that you use to connect to the board (e.g, Ethernet). Right-click on that interface and select **Properties**.
 - d. In the **Properties** window, click on the **Networking** tab, select the **Internet Protocol Version 4 (TCP/IPv4)** entry. Ensure that the checkbox is ticked, then click **Properties**.
 - e. Select **Use the following IP address** and enter IP address as **10.162.177.10** (or what matches with your board IP however the last digit should be different). The subnet mask should be **255.255.255.0**.

- f. Tick the **Validate settings upon exit** box and click **OK**, and **OK** once again for the network interface's **Properties** window.

Upon a successful connection to the internet, test your connection by opening the Jupyter Notebook in a web browser and navigate to

<http://10.162.177.99:9090>

(Whatever you set your board IP to + :9090). if your board is configured correctly you will be presented with a login screen. The username is **xilinx** and the password is also **xilinx**. After logging in, you will see the following screen:



In order to test the internet connection, try to [ping 8.8.8.8](#), or [ping www.google.com](#) in the Jupyter terminal. You will see the [ping](#) command report a non-zero number of bytes being received from the board's IP.

2.4 System Organization & Navigation

PYNQ uses the Jupyter Notebook platform, making it easier for embedded systems designers to build a more capable and high performance applications. As it allows the users to use Python language and libraries, thus, a strong candidate for machine learning applications as well.

The PYNQ image includes the following folders that will be displayed in the Jupyter Notebook:

Base includes examples related to the base overlay

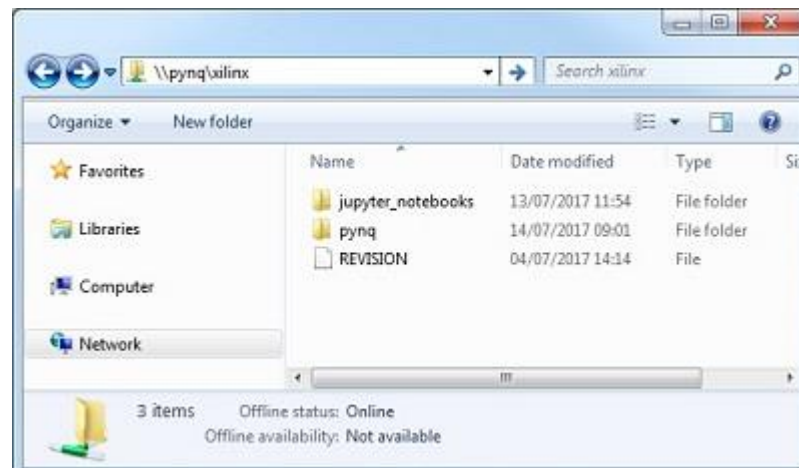
Common includes examples that are not overlay specific
Getting_started includes some introductory Jupyter notebooks

2.5 Accessing the System

Samba is used in order to transfer a large sized folder without facing any trouble. Samba is a file sharing service that is running on the PYNQ board. It allows to access the PYNQ home area in windows Explorer by typing the board IP address in the navigation bar.

[\\192.168.2.99\xilinx](http://192.168.2.99/xilinx)

When prompted, the username is **xilinx** and the password is **xilinx**. The following screen should appear:



3. Using the System

PYNQ Board hardware libraries (Overlays) are programmable FPGA designs to expand the functionality and application from the Zynq processing system to the programmable logic. Overlays can be used to customize the hardware platform for a particular application. The base overlay is included with the PYNQ-Z1 image in the python pynq package. The python package consists of four main sub packages: board, iop, drivers and bitstream.

The **board sub package** contains libraries for peripherals on the PYNQ board, such as buttons, switches, regled and led. The following code is an example to turn on the LEDs:

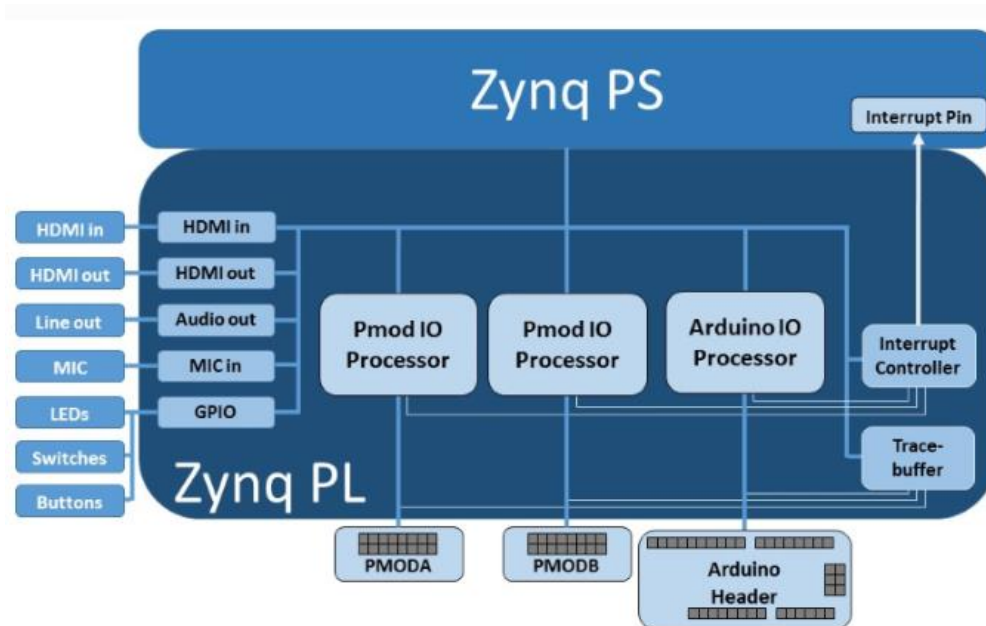
```
from pynq.board import LED
led = LED(0)
led.on()
```

The **iop sub package** contains libraries for Pmod devices and Arduino interface on the PYNQ board. The following code will instantiate and write to the Pmod_OLED attached to PMODA:

```
from pynq.iop import Pmod_OLED
from pynq.iop.iop_const import PMODA

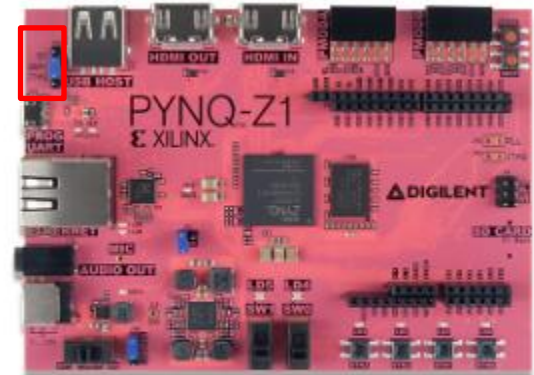
pmod_oled = Pmod_OLED(PMODA)
pmod_oled.clear()
pmod_oled.write('Welcome to the\nPynq-Z1 board!')
```

The **bitstream sub package** contains the base.bit (precompiled overlay) and base.tcl (hardware information it is built from). The **driver sub package** contains classes to support audio, video, DMA and Trace Buffer. The peripherals are detailed in the pynq digilent reference manual (<https://reference.digilentinc.com/reference/programmable-logic/pynq-z1/reference-manual>), in addition, to the pynq readthedocs (<https://pynq.readthedocs.io>). This chapter covers a quick view over some of the PYNQ systems.



3.1 Boot mode selection

The PYNQ-Z1 supports MicroSD and 16MB Quad SPI Flash onboard both loaded by the PYNQ image. It also supports JTAG boot modes, where the processor will wait until software is loaded by a host computer using the Xilinx tools. After software has been loaded, it is possible to either let the software begin executing, or step through it line by line using Xilinx SDK. The boot mode is selected using the Mode jumper that is move to the appropriate position as indicated by the label on the board.

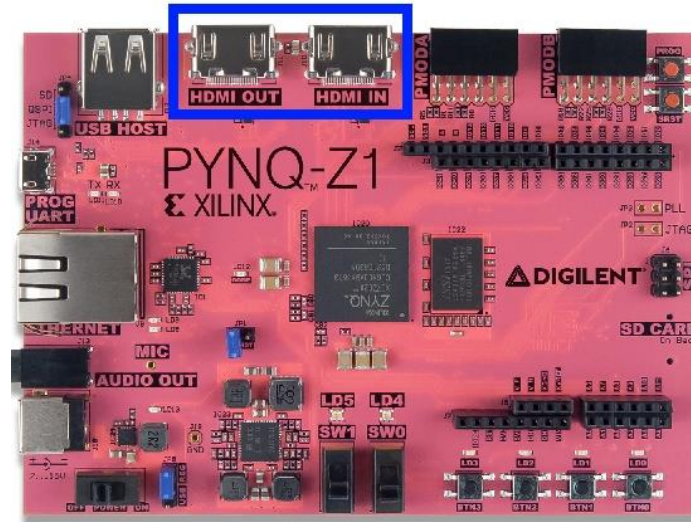


3.2 Clock Sources

The PYNQ-Z1 runs a 50 MHz clock to the Zynq PS_CLK input, which is used to generate the clocks for each of the PS subsystems. Additionally, the PYNQ-Z1 provides an external 125 MHz reference clock directly to pin H16 of the PL. The external reference clock allows the PL to be used completely independently of the PS, which can be useful for simple applications that do not require the processor.

3.3 HDMI

The PYNQ-Z1 contains two unbuffered HDMI ports: one source port J11 (output), and one sink port J10 (input). Both ports use HDMI type-A receptacles with the data and clock signals terminated and connected directly to the Zynq PL.



The base overlay contains a HDMI input controller, and a HDMI Output controller, both connected to their corresponding HDMI ports. A frame can be captured from the HDMI input, and streamed into DDR memory. The frames in DDR memory, can be accessed from Python.

A framebuffer can be shared between HDMI in and HDMI out to enable streaming. Both interfaces can be controlled independently.

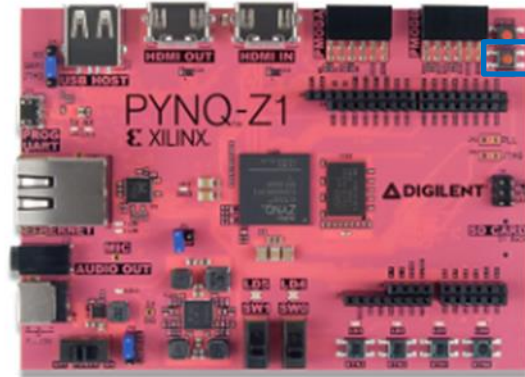
3.4 Reset Sources

3.4.1 Power-on Reset

The external power-on reset is the master reset of the entire chip. This signal resets every register in the device capable of being reset.

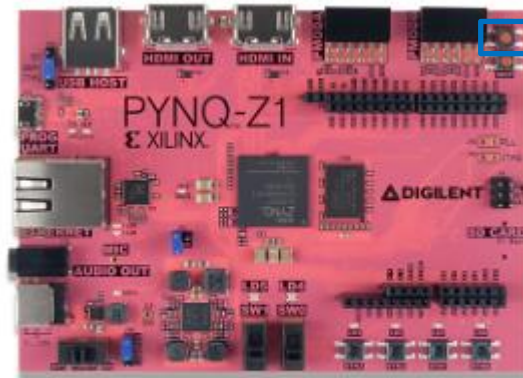
3.4.2 Program Push Button Switch

A PROG push switch, labeled PROG, resets the PL and causes DONE to be de-asserted. The PL will remain unconfigured until it is reprogrammed by the processor.



3.4.3 Processor Subsystem Reset

The external system reset, labeled SRST, reset erases all memory content within the PS, including the OCM. The PL is also cleared during a system reset.



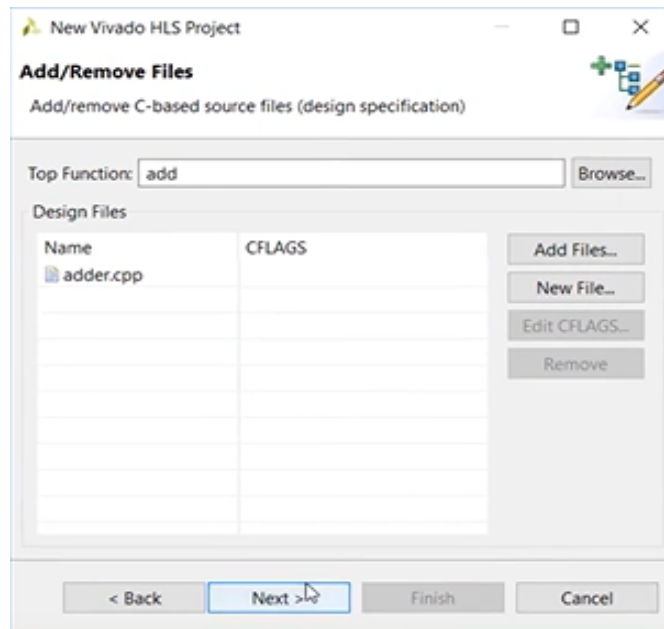
4. Xilinx Vivado integrated into PYNQ

Vivado software is used to create the Programmable Logic design. By providing the bitstream (.bit file) that is used to program the Zynq Board. This allow the user to create highly optimized design for specific task that could be integrated into PYNQ and used in the python interface like any other python package. This section covers an example on **creating an overlay using Vivado HLS**. The example is based on the example in the online PYNQ documentary.

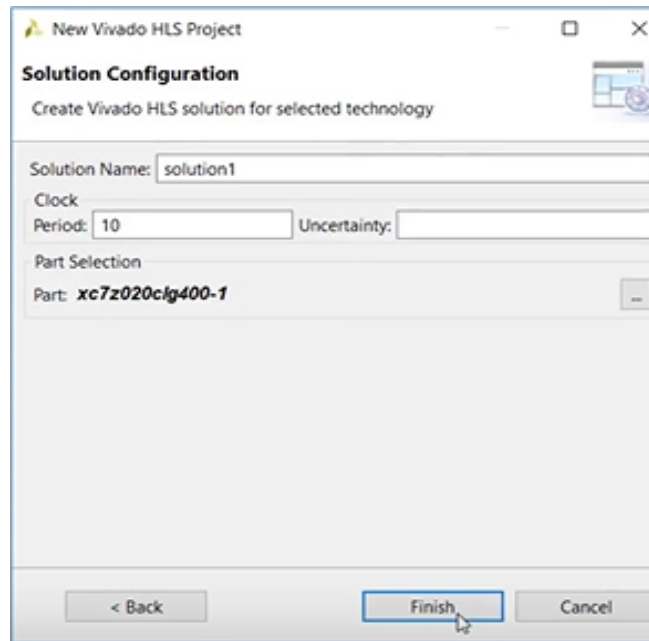
4.1 Create a custom IP

First, we create custom IP using Vivado HLS:

1. make sure you have downloaded the pynq z1 board files [https://github.com/cathalmccabe/pynq-z1_board_files/raw/master/pynq-z1.zip] and copy the file into the vivado instillation. (usually located in c:\Xilinx\Vivado\version\data\boards\board_files).
2. **Create new HLS project** with the **project name** addder the following window will come up. Click **Next**




3. For your **Top Function** use the name add cause that is the actual function name of our code. And **Add** a new file for our code (the c code that will implement the IP) and name it “adder.cpp” and then click **Next**. For this example we don't have a test bench so we can skip and click **Next**
4. Finally, in the following window you will have to the same part as on the PYNQ Board “xc7z020clg400-1” and Click **Finish**



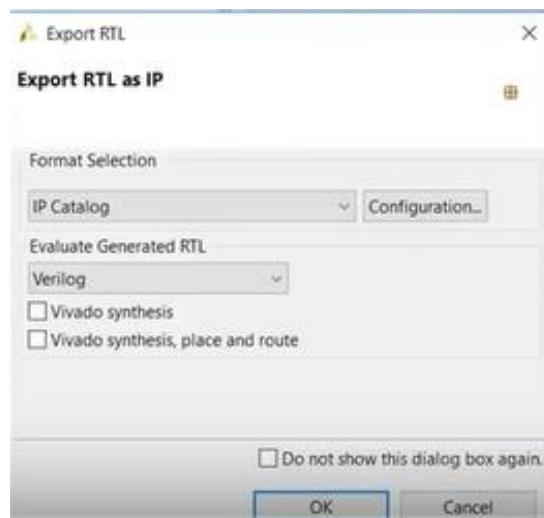
5. That will create the HLS Project. Now copy the following code in the adder.cpp file. The adder IP has a s_axilite interface with 2 inputs (a & b) and 1 output (c)

```
void add(int a, int b, int& c) {
    #pragma HLS INTERFACE ap_ctrl_none port=return
    #pragma HLS INTERFACE s_axilite port=a
    #pragma HLS INTERFACE s_axilite port=b
    #pragma HLS INTERFACE s_axilite port=c

    c = a + b;
}
```

6. Click on the **run synthesis button** . It is going to convert the c code into a hardware definition language [Verilog or vhd].

7. Export RTL



- Note down the register offsets of a, b, and c in the file called **xadd_hw.h**, that contains all the offsets of your used registers accessible on the axilite interface.

solution1 > misc > drivers > add_v1_0 > src > xadd_hw.h

Input a: 0x10

Input b: 0x18

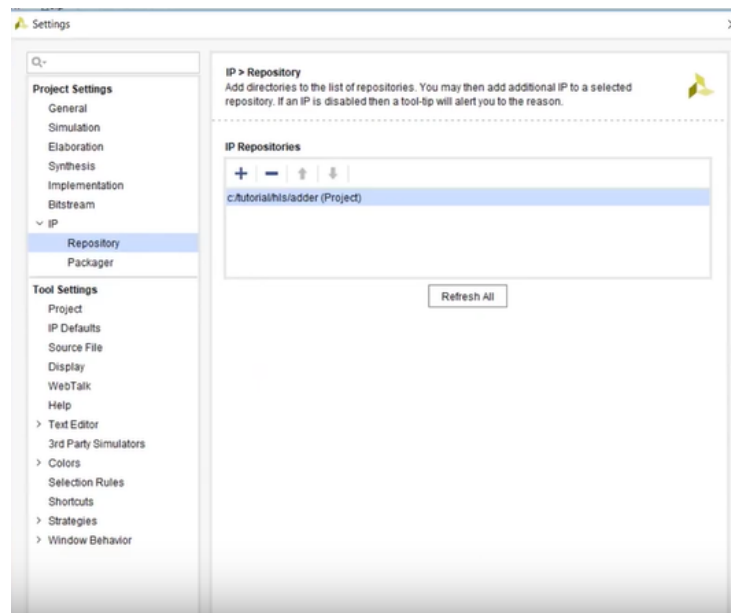
Output c: 0x20

4.2 Design overlay in vivado

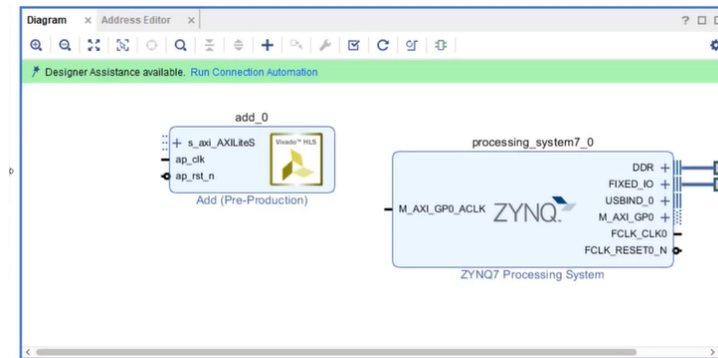
The Hardware Overlay is design using the Xilinx Vivado by exporting the Block Design containing the IPs.

- Open Vivado and **Create a new Vivado project**, give it a name and click **Next**. Then click on **Boards** and select **PYNQ-Z1** board, click finish.
- Click on **Create Block Design**, press ok
- Start adding the Zynq Processing System IP to the Block design by pressing on the + sign. **"ZYNQ7 Processing System"**
- Run the block automation** feature to set up the zynq according to the PYNQ z1 hardware with respect to the board files.
- In order to import the custom IP we created, we have to change the IP repository from the settings.

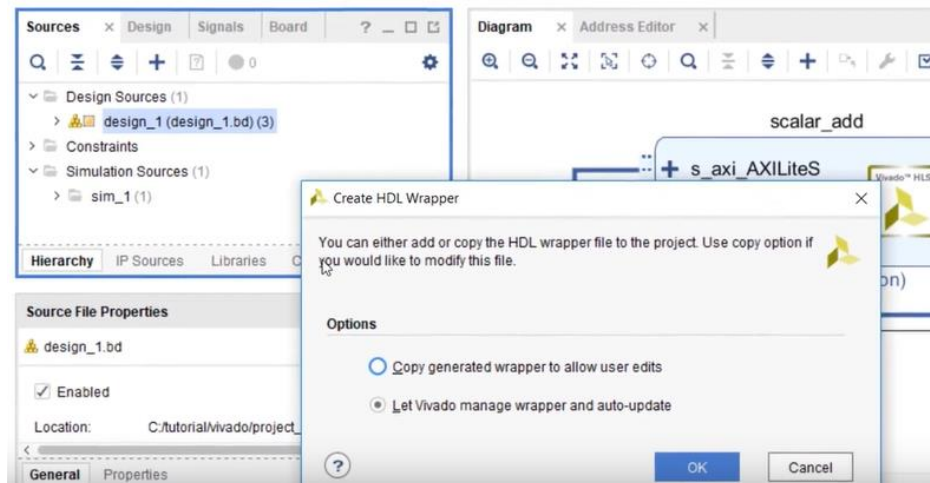
Tools > Settings > IP > Repository > add + > point to the HLS project folder we created earlier.



6. Add the custom IP by clicking into + in the Block diagram, search for add (Top function name)



7. **Run Connection automation** to connect the IP with the ZYNQ Processor. The name of the IP will be used is going to be used in the python code, thus, change the name of the block from add_0 to scalar_add to match with the code in the PYNQ document.
8. Save the block design
9. Create a wrapper by right clicking on the design_1, select **Create HDL Wrapper**. Keep the setting as shown below. Press Ok

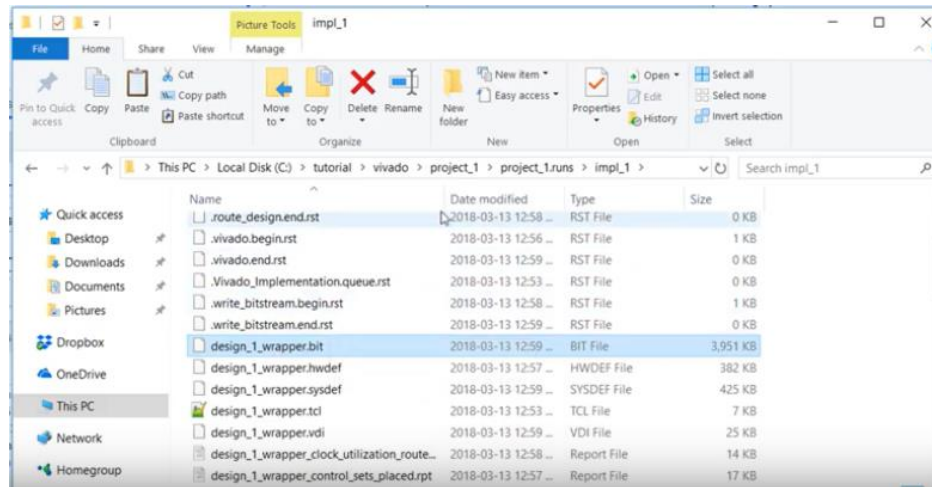


10. Once the wrapper is complete, you need to generate the bitstream by clicking **Generate Bitstream** under **Program and Debug**.
11. Export the block design to a tcl file that would be essential for the PYNQ libraries to interoperate the hardware design.

File > Export > Export Block Design

NOTE: PYNQ Libraries require to have the same name for the tcl and bitstream

The bit stream file is located in the **project run folder** in **impl_1**



4.3 Test IP in Python

Here the Block overlay exported design integrated to python interface and used within the Jupyter Platform.

1. The tcl and bitstream files created in part 4.2 should be copied to the SD card of the PYNQ Z1 board
2. Set the Board
3. Once you are ready, access the board by opening the file explorer and type [\\pynq](#) or your board IP address as shown in chapter 2.
4. Type in your Login: **Xilinx** and Password: **Xilinx**
5. Create an new file (named adder) in the overlays file in your pynq folder, copy the tcl and bitstream. Now the PYNQ has access to the overlay.
6. Now open the Jupyter web application, type " pynq:9090 " enter your password **xilinx**
7. Go to **New > Python3** to create a new notebook
8. Copy the following to the first code block. Which will load the overlay and program the FPGA with the bitstream.

```
from pynq import Overlay

overlay = Overlay('/home/xilinx/pynq/overlays/adder/adder.bit') \\ Location of the bitstream
file in the board
```

9. The code below will read and write function. It will write numbers to the input registers and read the output registers

```
add_ip = overlay.scalar_add
add_ip.write(0x10, 4)
add_ip.write(0x18, 5)
add_ip.read(0x20)
```

[6]: 9

10. It will show you an output of 9. or alternatively you can interact with the IP using the register map directly. The output is as shown below

```
add_ip = overlay.scalar_add
add_ip.register_map.a = 3
add_ip.register_map.b = 4
add_ip.register_map.c
```

[5]: Register(c=7)

4.4 Create a Driver

For a more user friendly call, a driver is created that contains that code and give you a handy function that is simpler to use.

1. The code below creates a class attribute consists of the IP types the driver should bind to. The constructor of the class should take a single *description* parameter and pass it through to the super class `__init__`. The description is a dictionary containing the address map and any interrupts and GPIO pins connected to the IP.

```
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)
        return self.read(0x20)
```

2. Reload the overlay to associate the customer driver with the IP.

```
overlay = Overlay('/home/xilinx/pynq/overlays/adder/adder.bit')
overlay?
```

And if we typed “ overlay? “ we will see that our new driver has been associated with the scalar_add

```
Type: Overlay
String form: <pynq.overlay.Overlay object at 0x2f31fdf0>
File: /opt/python3.6/lib/python3.6/site-packages/pynq/overlay.py
Docstring:
Default documentation for overlay /home/xilinx/pynq/overlays/adder/adder.bit. The following
attributes are available on this overlay:

IP Blocks
-----
scalar_add      : __main__.AddDriver
Hierarchies
```

3. Finally, to test out the add function we put in the driver. Giving us an output of 35

```
overlay.scalar_add.add(15,20)
```

```
[9]: 35
```

Appendix A: Useful links

Pynq-Z1 XDC constraints file

https://reference.digilentinc.com/_media/reference/programmable-logic/pynq-z1/pynq-z1_c.zip

PYNQ schematics for details on shared pins

http://www.tul.com.tw/download/TUL_PYNQ%20Schematic_R12.pdf

Vivado board files contain the configuration for a board that is required when creating a new project in Vivado.

https://github.com/cathalmccabe/pynq-z1_board_files/raw/master/pynq-z1.zip

The PYNQ peripherals details

<https://reference.digilentinc.com/reference/programmable-logic/pynq-z1/reference-manual>

The pynq readthedocs

<https://pynq.readthedocs.io>