

# Embedded Vision Workshop Series: From Zero to Hero with PYNQ

## Session 1

### Getting Started with PYNQ

#### Lab Workbook

*Presented by*  
**element14**  
AN AVNET COMMUNITY

*Sponsored by*  
 **XILINX**

# PYNQ Workshop Series

## About This Workbook

This workbook is intended to be used with **Session 1: Getting Started with PYNQ** of the **Embedded Workshop Series: From Zero to Hero with PYNQ**, presented by element14.

The contents of this workbook are created and owned by Aduvo Engineering and Training, Ltd.

## Your Instructor



**Adam Taylor**  
Founder and  
Principal Consultant



Please email any questions you may have to your instructor at [adam@adiuvoengineering.com](mailto:adam@adiuvoengineering.com).

## Required Hardware and Software

To complete this lab series, you will need the following hardware:

1. PYNQ-Z2 board
2. Micro SD card greater than equal to 16 GB
3. Micro SD card adapter
4. Micro USB cable
5. Ethernet cable
6. Ethernet access to your WIFI network or Ethernet connector on a PC
7. Sports camera e.g. [https://www.amazon.co.uk/Crosstour-Waterproof-Underwater-Wide-angle-Rechargeable/dp/B073WWSYJK?ref=fsclp\\_pl\\_dp\\_1](https://www.amazon.co.uk/Crosstour-Waterproof-Underwater-Wide-angle-Rechargeable/dp/B073WWSYJK?ref=fsclp_pl_dp_1)
8. HDMI to micro HDMI cable <https://www.amazon.co.uk/AmazonBasics-High-speed-latest-standard-meters/dp/B014I8TZXW>

To be able to complete these three labs you will need the following software on your development machine:

1. Vivado Design Suite 2019.1 – used for the development of custom overlays
2. 7-Zip - <https://www.7-zip.org/>
3. Etcher - <https://www.balena.io/etcher/>
4. Tera Term - <https://ttssh2.osdn.jp/index.html.en>
5. Pynq-Z2 board definition files - <https://d2m32eurp10079.cloudfront.net/Download/pynq-z2.zip>
6. WinSCP - <https://winscp.net/eng/index.php>

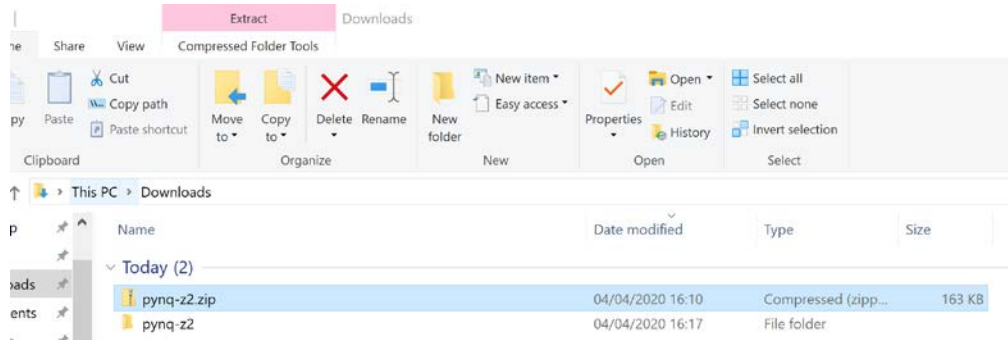
All project files and support files can be found at  
[https://github.com/ATaylorCEngFIET/Element14\\_PYNQ](https://github.com/ATaylorCEngFIET/Element14_PYNQ)

## Pre-Lab One

*This lab should be completed prior to the starting the labs*

### Getting Set Up Vivado

1. Install the webpack version of Vivado Design Suite 2019.1.
2. Once installed, unzip the Pynq-Z2 board definitions.

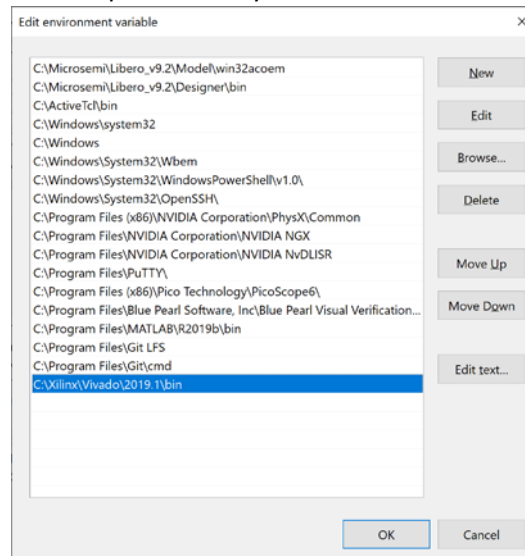


3. Copy the resulting directory into the Vivado boards directory. The path for this will be <install location>\Xilinx\Vivado\2019.1\data\boards\board\_files.

This PC > OS (C:) > Xilinx > Vivado > 2019.1 > data > boards > board\_files

Name	Date modified	Type	Size
ac701	07/12/2019 22:07	File folder	
adm-pcie3-ku3	02/04/2020 21:09	File folder	
adm-pcie-7v3	02/04/2020 21:09	File folder	
kc705	02/04/2020 21:10	File folder	
kcu105	02/04/2020 21:09	File folder	
kcu116	07/12/2019 22:08	File folder	
kcu1500	02/04/2020 21:10	File folder	
li-imx274-mipi	07/12/2019 22:09	File folder	
pico_m505	02/04/2020 21:10	File folder	
pynq-z2	04/04/2020 16:17	File folder	

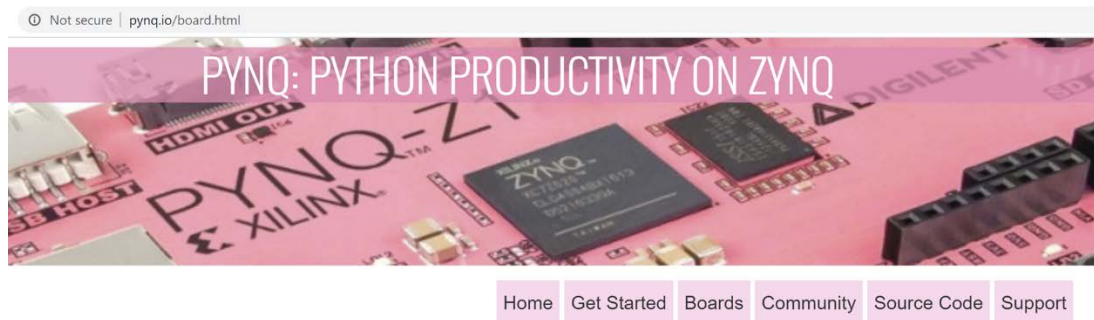
4. Ensure Vivado is declared in the path within your environment variables.



## Getting Up and Running with PYNQ

To be able to get working with the PYNQ-Z2 board, the first thing we need to do is create an SD card containing the PYNQ image.

1. Go to [www.pynq.io/boards](http://www.pynq.io/boards) and download the PYNQ-Z2 V2.5 PYNQ image. To download the image, you will need a Xilinx account. If you do not have one already, the download process will allow you to create one.



## Development Boards

There are currently four Zynq based boards officially supported by PYNQ: **Pynq-Z1 from Digilent**, **Pynq-Z2 from TUL**, **ZCU104 from Xilinx**, and **ZCU111 from Xilinx**. PYNQ also supports the **Xilinx Alveo** accelerator boards and **AWS-F1**. Images for supported Zynq based boards can be downloaded via the links below. For Alveo and AWS-F1, PYNQ can be installed on the host computer or AWS instance.

### Downloadable PYNQ images

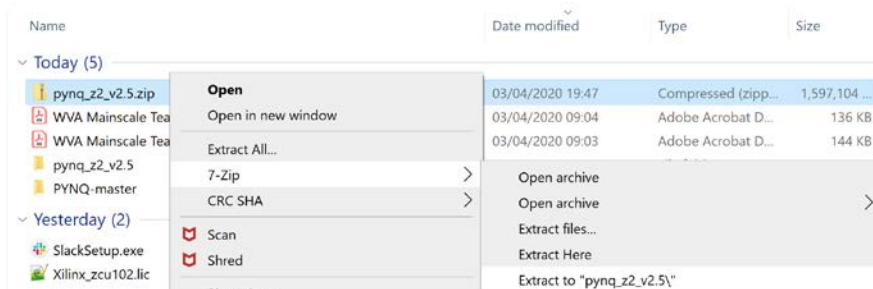
Images for supported Zynq based boards are available via the links below. The image includes board specific example overlays and Jupyter notebooks.

- [PYNQ-Z1 v2.5 PYNQ image](#)
- [PYNQ-Z2 v2.5 PYNQ image](#)
- [ZCU104 v2.5 PYNQ image](#)
- [ZCU111 v2.5 PYNQ image](#)

2. The file which is downloaded will be compressed.



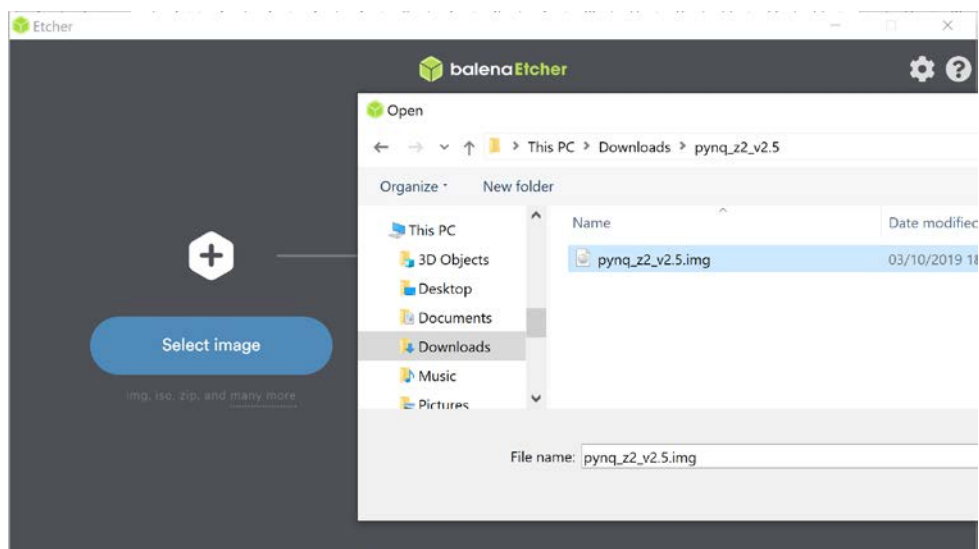
3. Un-compress the image using 7-Zip.



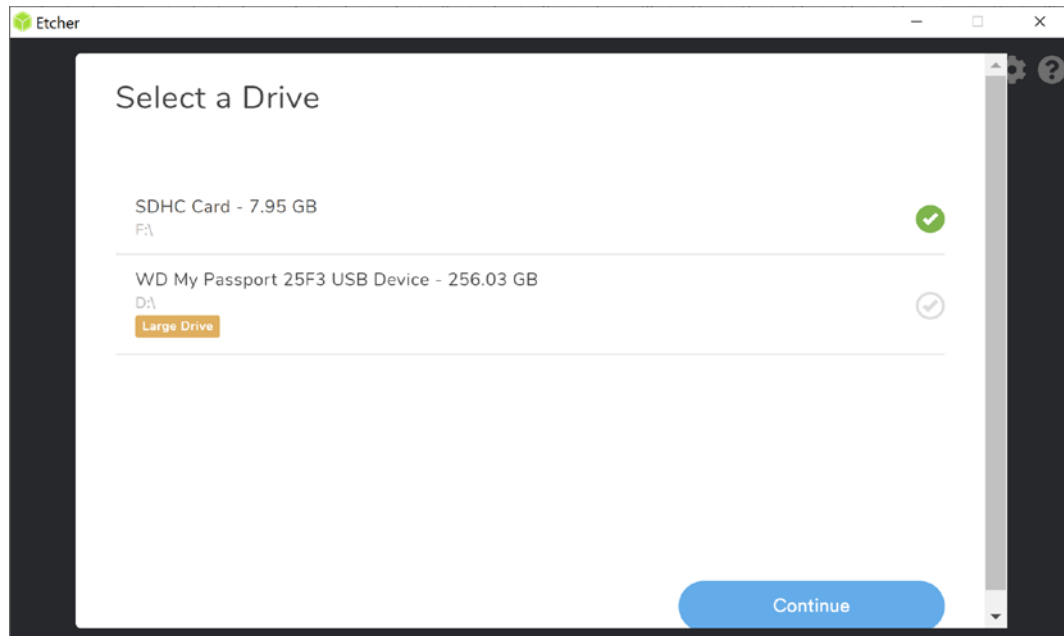
4. Insert the micro SD card into the adaptor and insert into your computer.



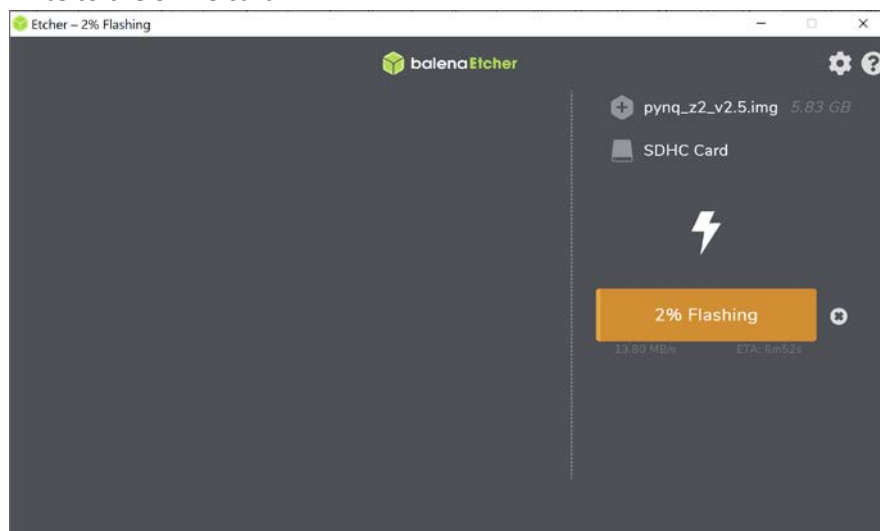
5. Open Etcher and select the PYNQ image.



6. Select the SDHC card in Etcher.

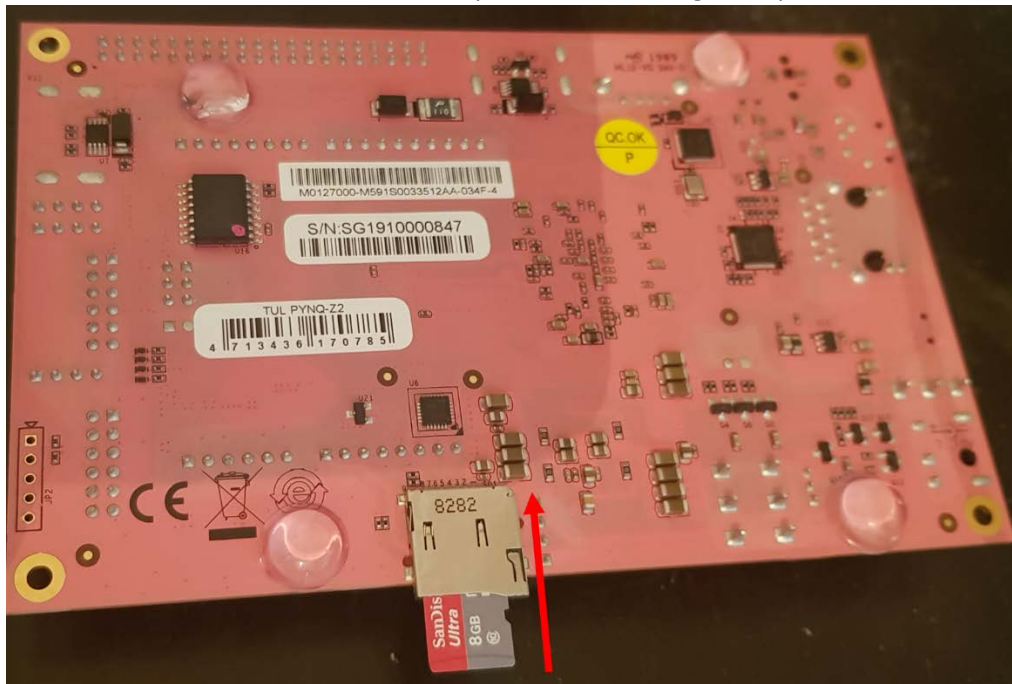


7. Start the write to the SDHC card.

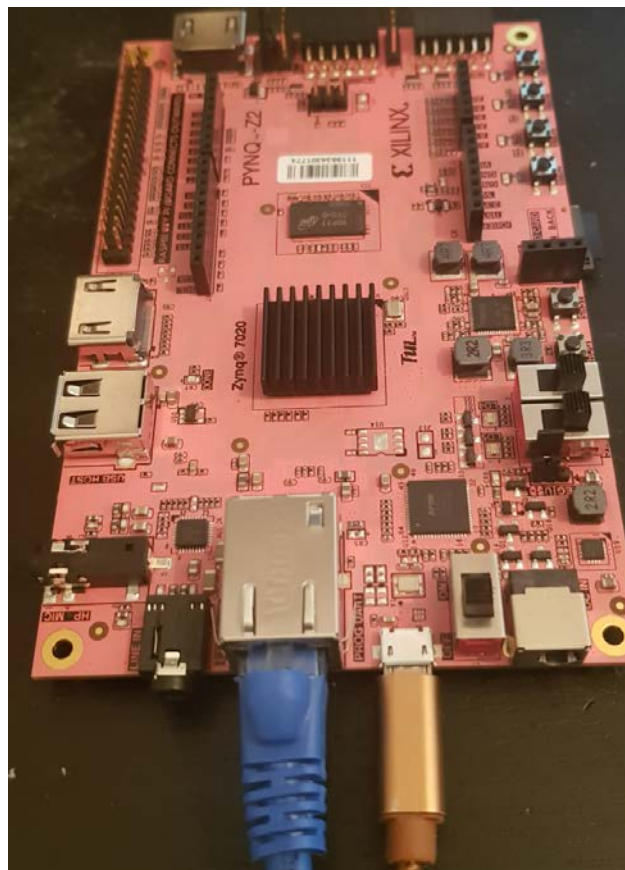




8. Once the write is completed, remove the SD card from your computer and insert it into the PYNQ-Z2 board. Ensure the PYNQ-Z2 is powered off during this operation.

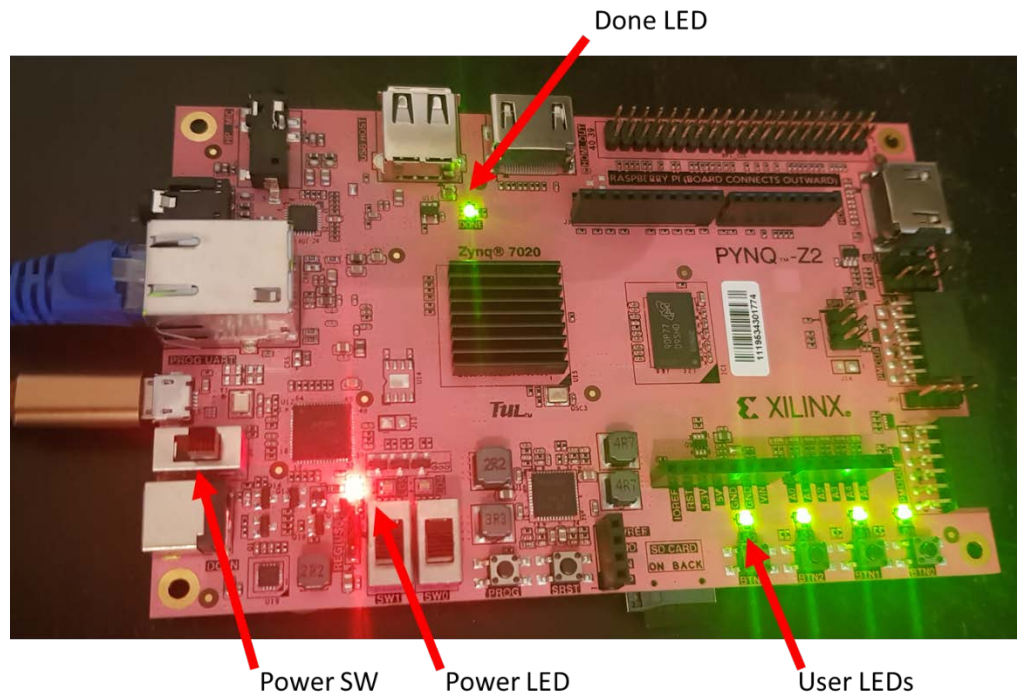


9. Connect the micro USB cable between the PYNQ-Z2 and your laptop. Also connect the Ethernet cable (see Annex A if connecting Ethernet directly to PC).

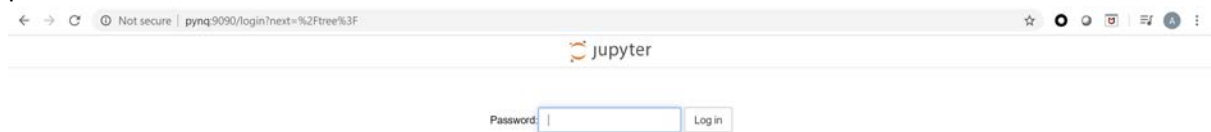




10. Power on the PYNQ-Z2. After a few seconds you will see the LED illuminate and the user LEDs flash and remain lit. This indicates the PYNQ-Z2 board has completed booting PYNQ.



11. The next step is to check that we can access the Jupyter Notebooks. On your computer, open a web browser and navigate to PYNQ:9090. You should see a Jupyter labs log in. The password for which is Xilinx.



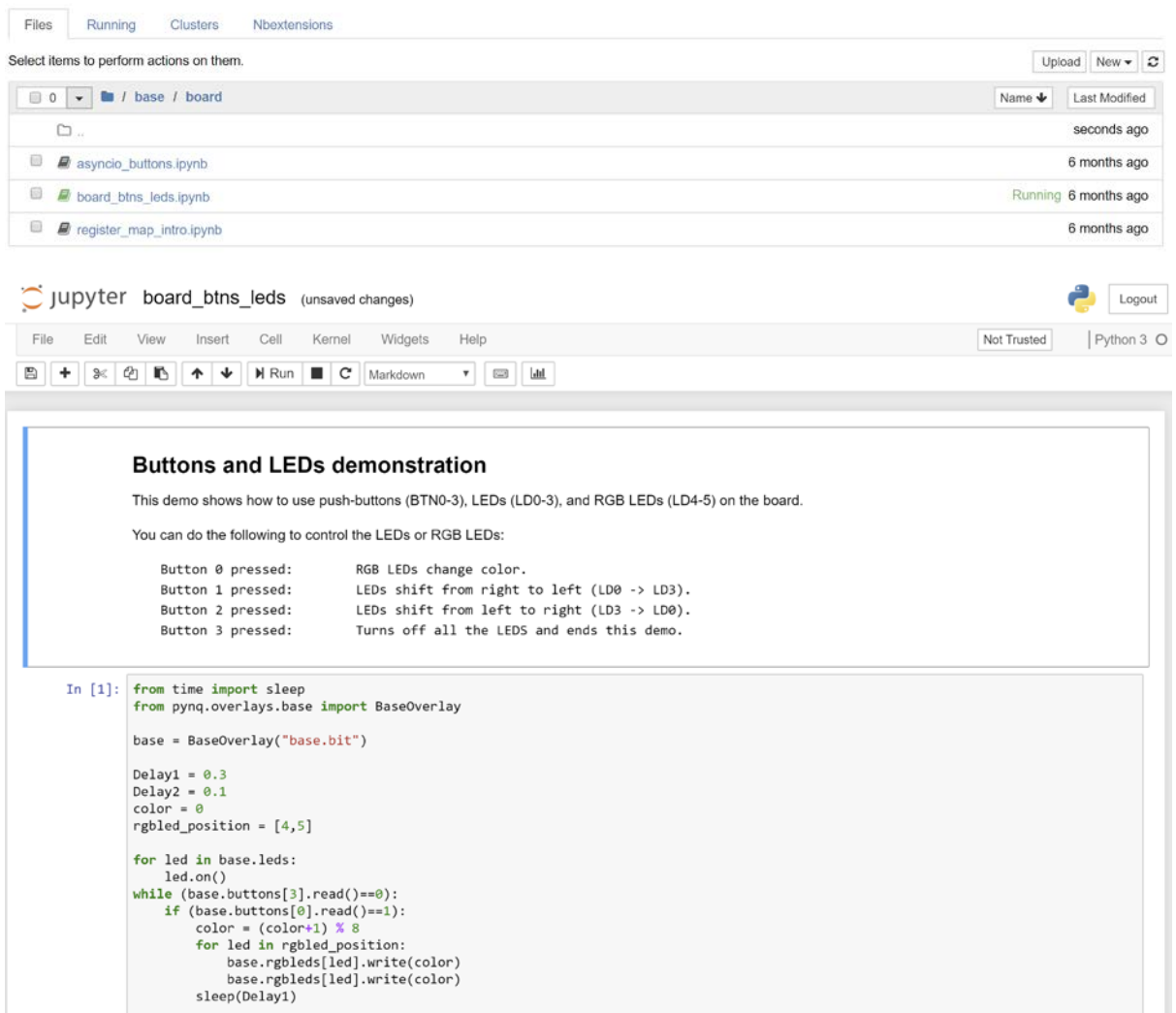
12. Logging in, you will see we are ready to begin our development with PYNQ.



## Lab One

In this lab, we are going to explore the base and logic tools overlay and look at how we can add in overlays from the PYNQ user community.

1. Power up your PYNQ-Z2 and log in to the Jupyter Notebook environment. If this is your first time, see Pre-Lab One above.
2. PYNQ works on a series of Jupyter Notebooks to enable us to work with the PS and the PL of the Zynq device. Notebooks can download programmable logic designs to the PL element of the design and interact with the PL using Python.
3. Let's take a look at one of the base overlay notebooks. In Jupyter labs, click on Base -> Board -> board\_btns\_leds.ipynb. This will open a new browser tab which contains the notebook.



The screenshot displays the Jupyter Notebook environment. At the top, there's a file browser showing the directory structure: 'base / board'. The file 'board\_btns\_leds.ipynb' is highlighted and marked as 'Running'. Below the file browser, the notebook interface is shown with the title 'board\_btns\_leds (unsaved changes)'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The toolbar contains various icons for file operations and execution. The notebook content is as follows:

### Buttons and LEDs demonstration

This demo shows how to use push-buttons (BTN0-3), LEDs (LD0-3), and RGB LEDs (LD4-5) on the board.

You can do the following to control the LEDs or RGB LEDs:

- Button 0 pressed: RGB LEDs change color.
- Button 1 pressed: LEDs shift from right to left (LD0 -> LD3).
- Button 2 pressed: LEDs shift from left to right (LD3 -> LD0).
- Button 3 pressed: Turns off all the LEDs and ends this demo.

```
In [1]: from time import sleep
from pynq.overlays.base import BaseOverlay

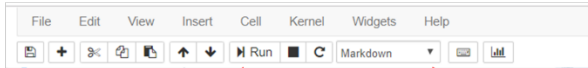
base = BaseOverlay("base.bit")

Delay1 = 0.3
Delay2 = 0.1
color = 0
rgbled_position = [4,5]

for led in base.leds:
    led.on()
while (base.buttons[3].read()==0):
    if (base.buttons[0].read()==1):
        color = (color+1) % 8
        for led in rgbled_position:
            base.rgbleds[led].write(color)
            base.rgbleds[led].write(color)
        sleep(Delay1)
```

- Run the cell of Python code by clicking on it and pressing Run Cell.

### Working With Jupyter Notebooks



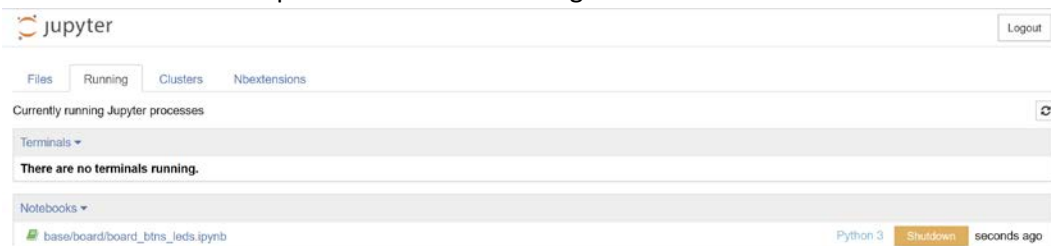
The screenshot shows the Jupyter Notebook toolbar with the following icons and their corresponding functions indicated by red arrows:

- Save:** Represented by a floppy disk icon.
- Run Cell:** Represented by a play button icon.
- Stop:** Represented by a square button with a diagonal line.
- Re Start Kernel:** Represented by a circular arrow icon.
- Cell Type:** Represented by a dropdown menu icon.

Below the toolbar, three code cells are shown to illustrate the execution state:

- When Kernel first starts showing cell has not run:** The first cell shows the code: `In [ ]: from pynq.overlays.base import BaseOverlay; overlay = BaseOverlay('base.bit'); trace_analyzer = overlay.trace_pmode`. The input prompt is `In [ ]:`.
- Asterix shows cell currently running:** The second cell shows the same code. The input prompt is `In [*]:`, indicating execution is in progress.
- Cell Execution Complete Number increments to show how often it was run:** The third cell shows the same code. The input prompt is `In [1]:`, indicating execution is complete and the cell has been run once.

- This will run an application which uses the LEDs and buttons on the Zynq device using the base overlay.
  - Press and hold button 0. You will see the tri color LED cycle around colors while being held.
  - Press and hold button 1. You will see the LED 0 – 3 flash in a pattern right to left.
  - Press and hold button 2. You will see the LED 0 – 3 flash in a pattern left to right.
  - Press button 3 and the demonstration will end.
- Close the notebook `board_btns_leds.ipynb`. This will not stop the kernel running. On the home page, click on Running and you will see that it is still shown as running. Click on Shutdown. This will stop the kernel from running.



- The base overlay provides the ability to quickly and easily interface and work with all of the interfaces on the PYNQ board, including the HDMI, Audio, Pmod, RPI and Shield Connector.

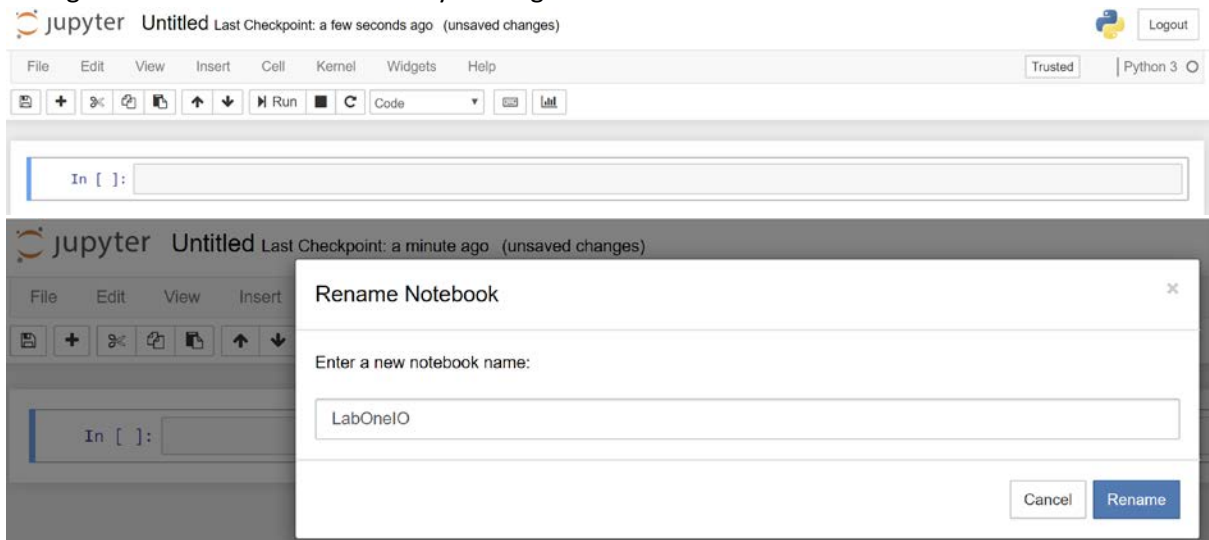
8. Open the logictools directory from the homepage.



9. Create a new notebook by clicking on New->Python3.

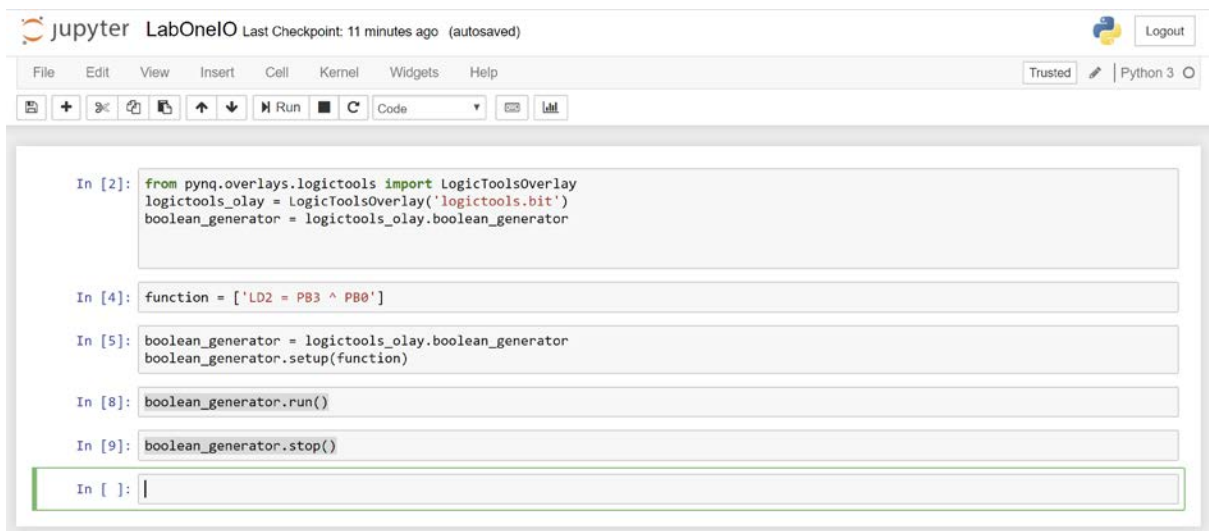


10. Change the name of the notebook by clicking on Untitled and enter the name LabOneIO.



11. Enter the following code into the cell:

```
from pynq.overlays.logictools import LogicToolsOverlay
logictools_olay = LogicToolsOverlay('logictools.bit')
boolean_generator = logictools_olay.boolean_generator
function = {'XOR_gate': 'LD2 = PB3 ^ PB0'}
boolean_generator = logictools_olay.boolean_generator
boolean_generator.setup(function)
boolean_generator.run()
boolean_generator.stop()
```



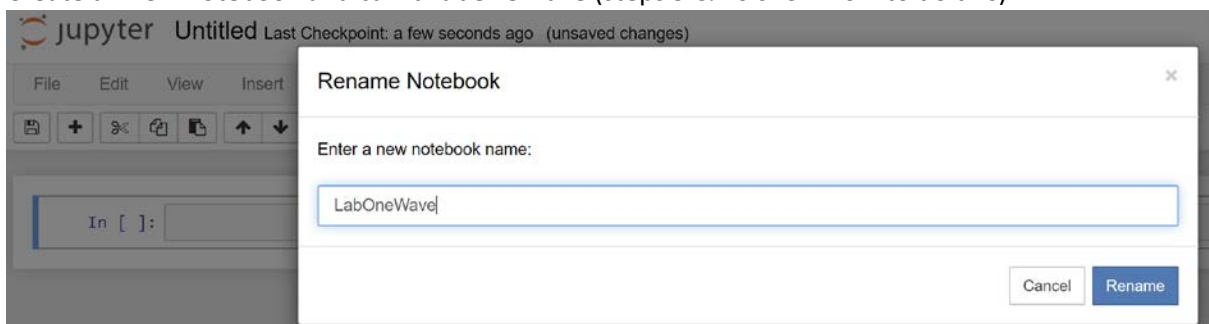
12. Press the BTN3 and BTN2 and see the implementation of an OR gate that lights up the LED 3.

13. Try different logic implementations. AND = &, XOR = ^, OR = | and NOT = ~.

14. Congrats, you have created your first PYNQ application (do not worry we are going to get much more complicated).

15. Close the notebook and shutdown the kernel as we did in step 6.

16. Create a new notebook and call it LabOneWave (steps 9 & 10 show how to do this).



17. Enter the following code:

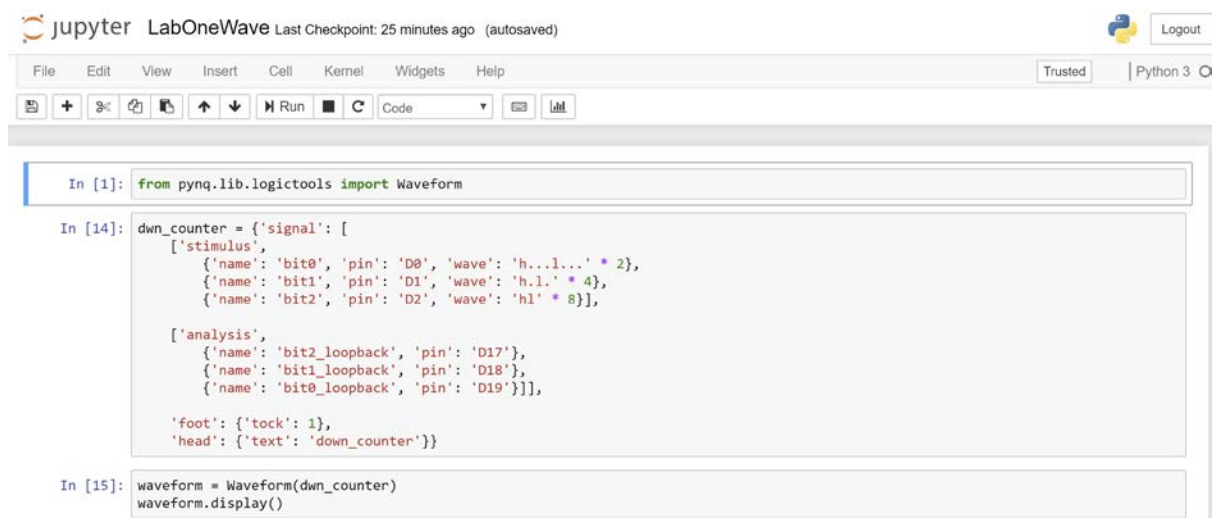
```
from pynq.lib.logictools import Waveform

dwn_counter = {'signal': [
    ['stimulus',
     {'name': 'bit0', 'pin': 'D0', 'wave': 'h...l...' * 2},
     {'name': 'bit1', 'pin': 'D1', 'wave': 'h.l.' * 4},
     {'name': 'bit2', 'pin': 'D2', 'wave': 'hl' * 8}],

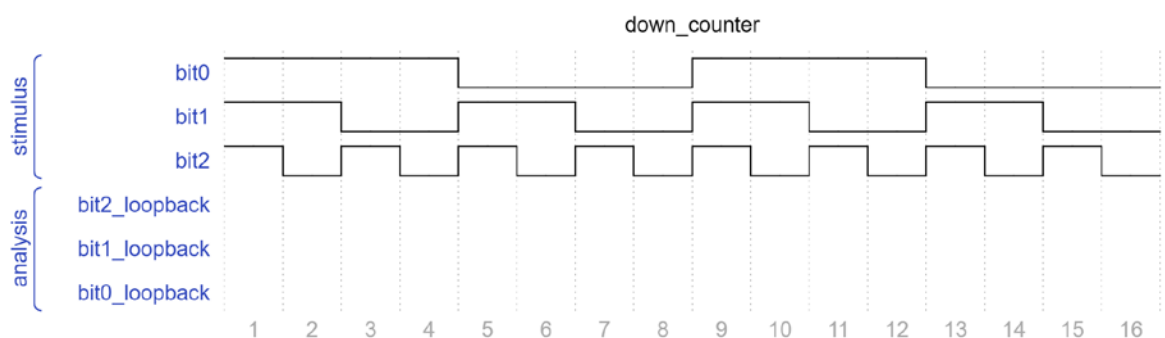
    ['analysis',
     {'name': 'bit2_loopback', 'pin': 'D17'},
     {'name': 'bit1_loopback', 'pin': 'D18'},
     {'name': 'bit0_loopback', 'pin': 'D19'}]],

    'foot': {'tock': 1},
    'head': {'text': 'down_counter'}}

waveform = Waveform(dwn_counter)
waveform.display()
```

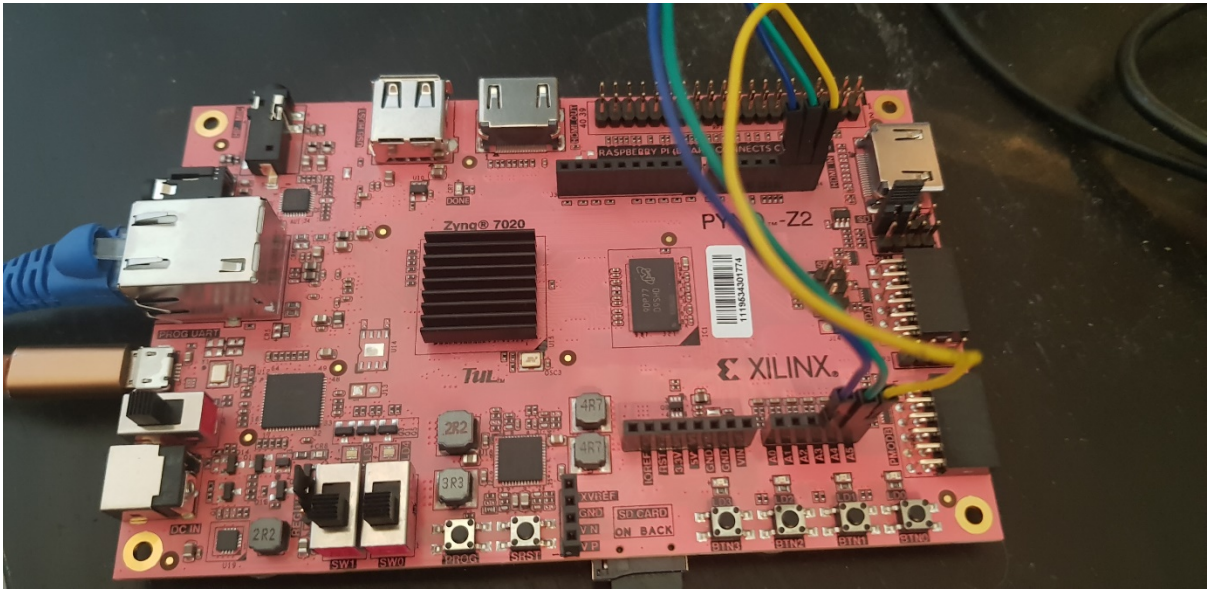


18. Run each of these cells. You should see a binary down counter however, the analysis elements will be empty.





19. To be able to loop these back, we need to connect the flying lead connectors between the Arduino header as shown below. Make sure the PYNQ-Z2 is powered down.



20. Power up the board and re-open the LabOneWave notebook. Enter the following code in the notebook:

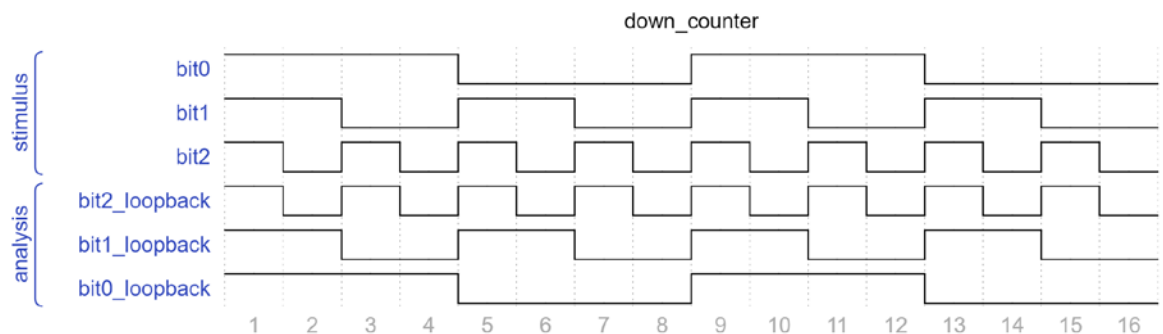
```
pattern_generator = logictools_olay.pattern_generator
pattern_generator.trace(num_analyzer_samples=16)
pattern_generator.setup(dwn_counter, stimulus_group_name='stimulus',
analysis_group_name='analysis')
```

```
pattern_generator.run()
pattern_generator.show_waveform()
```

```
In [11]: pattern_generator = logictools_olay.pattern_generator
        pattern_generator.trace(num_analyzer_samples=16)
        pattern_generator.setup(dwn_counter, stimulus_group_name='stimulus', analysis_group_name='analysis')

In [12]: pattern_generator.run()
        pattern_generator.show_waveform()
```

21. Run the notebook cells and you should see the received patterns as output.



This ability to generate and capture waveforms really helps when we want to quickly and easily interface and test with sensors.



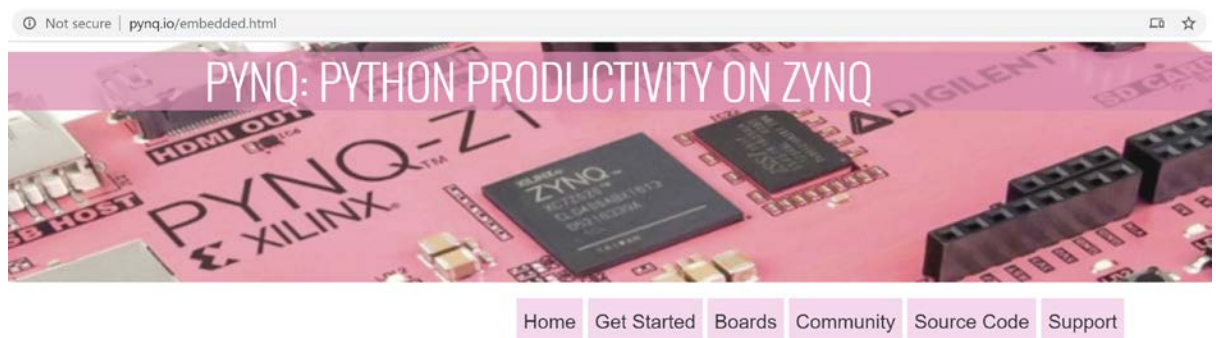
22. Close the notebook and shut it down as we have previously.

23. Open a new terminal window from the home screen.



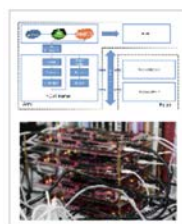
24. We are going to use this terminal to install a new overlay.

25. In a browser on your computer open <http://www.pynq.io/embedded.html>.



## PYNQ community projects

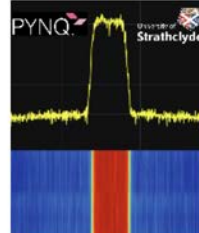
**SNN simulator on PYNQ cluster**  
Jiangnan University, China



**CC-Cam a PYNQ diffuser camera**  
Southeast University, China



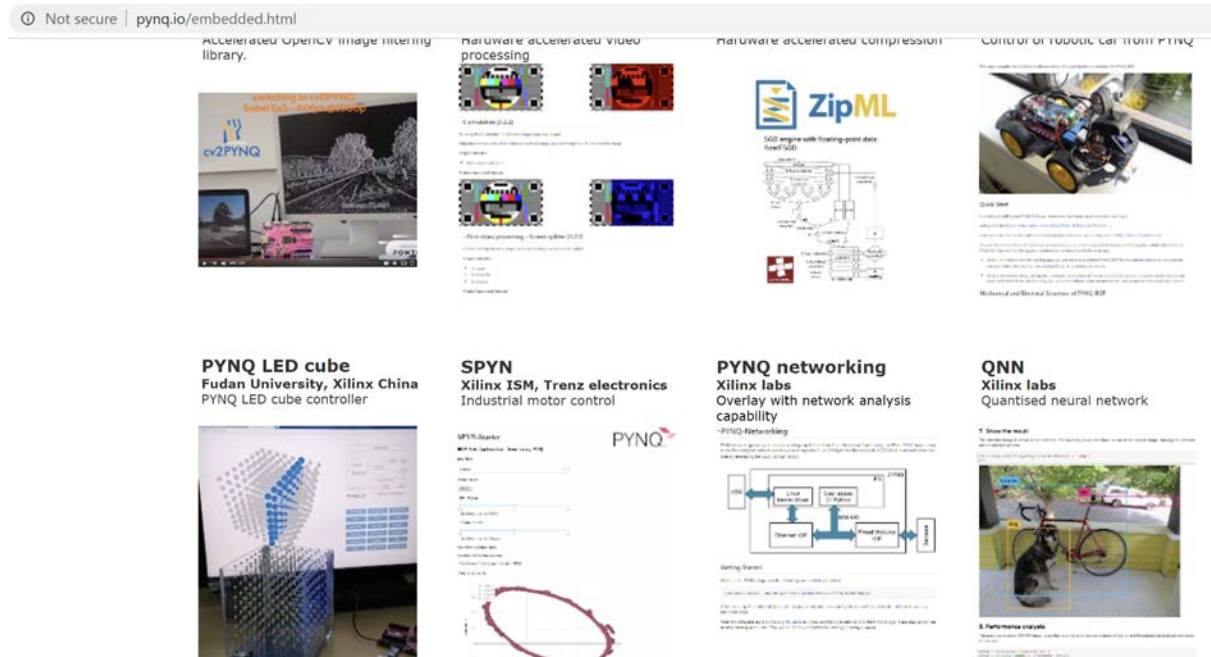
**Open Source RFSoc Spectrum Analyzer**  
University of Strathclyde



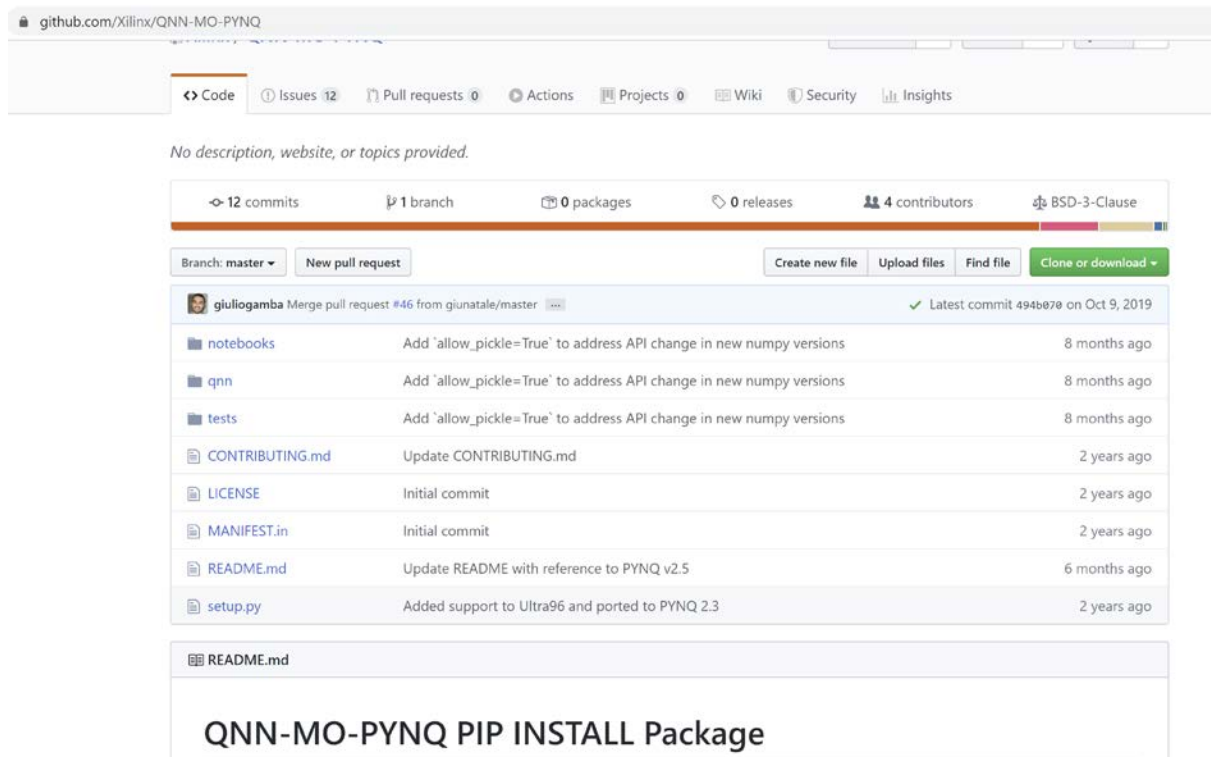
**PYNQ S-Curve motion controller**  
CalPolyPomona  
Mahan Bastani, Nolan Chang, Atsushi Domyo, Daniel Kizito



26. Scroll down until you find the QNN Xilinx Labs.



27. Click on the QNN overlay. This will take you to the GitHub for the overlay.



28. Copy the installation code from the GitHub, paste it in the terminal and press Enter.

## QNN-MO-PYNQ PIP INSTALL Package

This repo contains the pip install package for Quantized Neural Network (QNN) on PYNQ using a Multi-Layer Offload (MO) architecture. Two different overlays are here included, namely W1A2 (1 bit weights, 2 bit activations) and W1A3 (1 bit weights, 3 bit activations), executing in the Programmable Logic 1 Convolutional layer and 1 (optional) Max Pool layer.

### Quick Start

In order to install it on your PYNQ board, connect to the board, open a terminal and type:

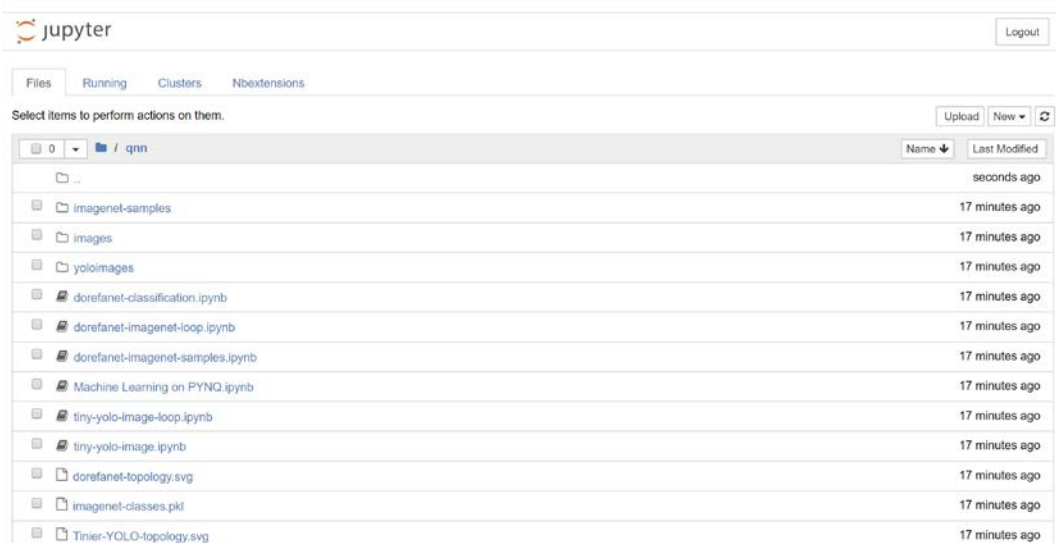
```
# (on PYNQ v2.3 and later versions, tested up to v2.5)
sudo pip3 install git+https://github.com/Xilinx/QNN-MO-PYNQ.git
```

```
root@pynq:/home/xilinx# sudo pip3 install git+https://github.com/Xilinx/QNN-MO-PYNQ.git
Collecting git+https://github.com/Xilinx/QNN-MO-PYNQ.git
  Cloning https://github.com/Xilinx/QNN-MO-PYNQ.git to /tmp/pip-2xhnjx01-build
Installing collected packages: qnn-loopback
  Running setup.py install for qnn-loopback ... done
Successfully installed qnn-loopback-0.1
root@pynq:/home/xilinx#
root@pynq:/home/xilinx#
```

29. On the homepage you should see a new directory called QNN. Open it.



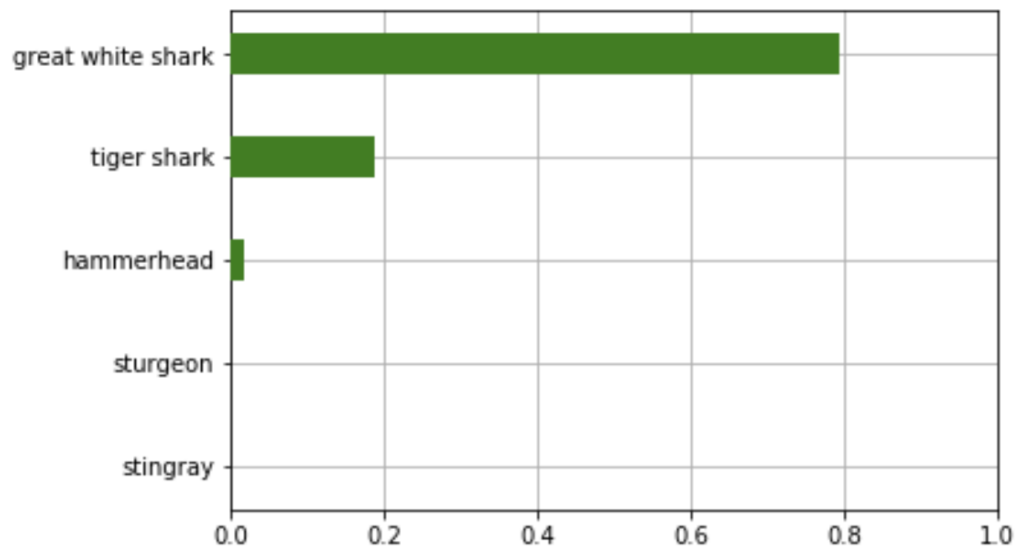
The screenshot shows the JupyterLab interface. At the top, there's a 'jupyter' logo and a 'Logout' button. Below it, there are tabs for 'Files', 'Running', 'Clusters', and 'Nbextensions'. The 'Files' tab is active, showing a list of files and directories. The list includes 'base', 'common', 'getting\_started', 'logictools', 'pynqOpenCV', 'qnn', and 'Welcome to Pynq.ipynb'. The 'qnn' directory is highlighted, indicating it's the selected item.



The screenshot shows the JupyterLab interface with the 'qnn' directory selected. The file browser displays a list of files and directories within 'qnn'. The list includes '...', 'imagenet-samples', 'images', 'yoloimages', 'dorefanet-classification.ipynb', 'dorefanet-imagenet-loop.ipynb', 'dorefanet-imagenet-samples.ipynb', 'Machine Learning on PYNQ.ipynb', 'tiny-yolo-image-loop.ipynb', 'tiny-yolo-image.ipynb', 'dorefanet-topology.svg', 'imagenet-classes.pkl', and 'Tinier-YOLO-topology.svg'. The 'qnn' directory is highlighted, indicating it's the selected item.

30. Open the dorefanet-classification.ipynb and run each of the cells.

class:	great white shark	probability:	79.40%
class:	tiger shark	probability:	18.92%
class:	hammerhead	probability:	1.63%
class:	sturgeon	probability:	0.01%
class:	stingray	probability:	0.01%



This demonstrates not only the ease with which overlays can be added to the system but also how complicated functions which use the programmable logic can be implemented and controlled from Python to give a significant acceleration.

In the next lab we will look at how we can develop our own overlay!