XILINX
ALL PROGRAMMABLE™

PYNQ™

DAC 2018 FPGA design contest

Naveen Purushotham, Xilinx
Jingtong Hu, University of Pittsburgh
Bei Yu, Chinese University of Hong Kong
Xinyi Zhang, University of Pittsburgh

Agenda

- Welcome
- DAC Contest Committee
  - Contest Introduction
  - Webinars
  - Piazza
- PYNQ$_{TM}$ & Reference design discussion
  - Things to know
  - Design rules
  - Reference design
- Questions & Answers

DAC Contest Committee

PYNQ

PYNQ

Python productivity for Zynq

An open-source framework for combining SW and HW libraries on Zynq

- Use SW libraries of Python language
- Exploit programmable logic and microprocessors using HW libraries

Out of Box

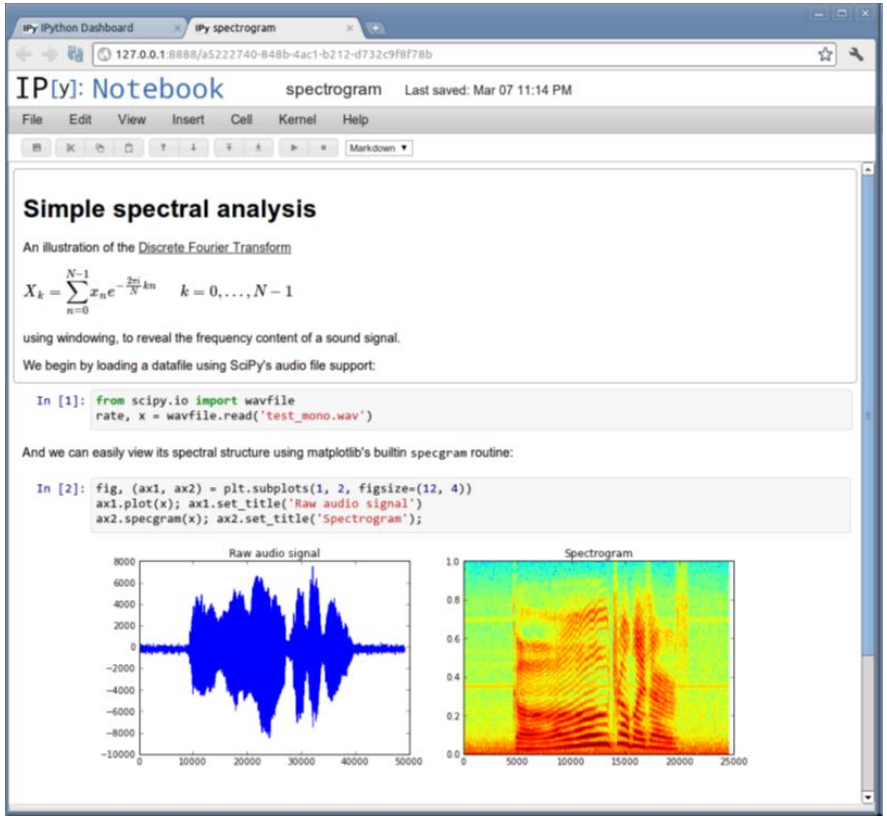
Prebuilt SD card – Python, Jupyter, Ubuntu & Bitstreams

Python & Debian & Bitstream extensible

PYNQ marries data science software and capabilities of zynq and programmable hardware

PYNQ-Z1 – First PYNQ supported board

Jupyter Notebooks: browser-based development … with rich, multi-media support

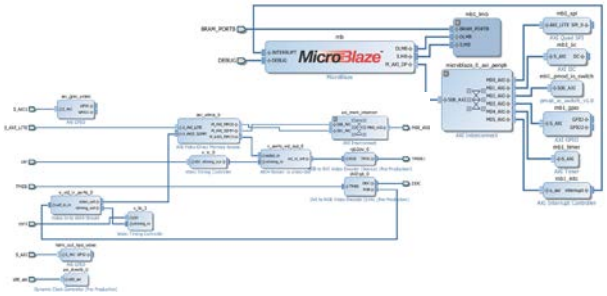


- Designed for
  - Interactive, exploratory computing
  - Reproducible results
- Ideal for
  - Teaching and learning
  - Projects and research
- Provides
  - Interactive design with Zynq
  - application-oriented perspective

github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks

Where to find more notebooks
https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks

Overlays aka hardware libraries – special bitstreams



Step 1:

Create an FPGA design for a class of related applications

```
pmod_init(0,1);
while(1){
    while((MAILBOX_CMD_ADDR & 0x01
    cmd=MAILBOX_CMD_ADDR;

    count = (cmd & 0x0000ff00) >>
    if((count==0) || (count>253)) {
        // clear bit[0] to indicat
        // set rest to 1s to indic
        MAILBOX_CMD_ADDR = 0xffffff
        return -1;
    }
    for(i=0; i<count; i++) {
        if (cmd & 0x08) // Python
        {
            switch ((cmd & 0x06) >> 1) { // use bit[2:1]
                case 0 : MAILBOX_DATA(i) = *(u8 *) MAILBOX_ADDR; break;
                case 1 : MAILBOX_DATA(i) = *(u16 *) MAILBOX_ADDR; break;
                case 2 : break;
                case 3 : MAILBOX_DATA(i) = *(u32 *) MAILBOX_ADDR; break;
            }
```

```
6c78 3963 7367 3232 3500 6300 0b32
332f 3039 2f33 3000 6400 0931 323a
3a31 3900 6500 0532 7cff ffff ffff
ffff ffff ffff ffff ffaa 9955 6630
0720 0031 a103 8031 413d 0831 6109
c204 0010 9330 e100 cf30 c100 8120
0020 0020 0020 0020 0020 0020 0020
0020 0020 0020 0020 0020 0020 0020
813c c831 8108 8134 2100 0032 0100
e1ff ff33 2100 0533 4100 0433 0101
6100 0032 8100 0032 a100 0032 c100
```

Step 2:

Export the bitstream and a C API for programming the design

```
void setNormalDisplay(){
    sendCommand(OLED_Normal_Display_Cmd);
}

void setInverseDisplay(){
    sendCommand(OLED_Inverse_Display_Cmd);
}

int main(void)
{
   int cmd;
   int Row, Column;

   arduino_init(0,0,0,0);
   config_arduino_switch(A_GPIO, A_GPIO, A_GPIO,
                         A_GPIO, A_SDA, A_SCL,
                         D_GPIO, D_GPIO, D_GPIO, D_GPIO, D_GPIO,
                         D_GPIO, D_GPIO, D_GPIO, D_GPIO,
                         D_GPIO, D_GPIO, D_GPIO, D_GPIO);
   // Initialization
   oled_init();
```

Step 3:

Wrap the C API to create a Python library

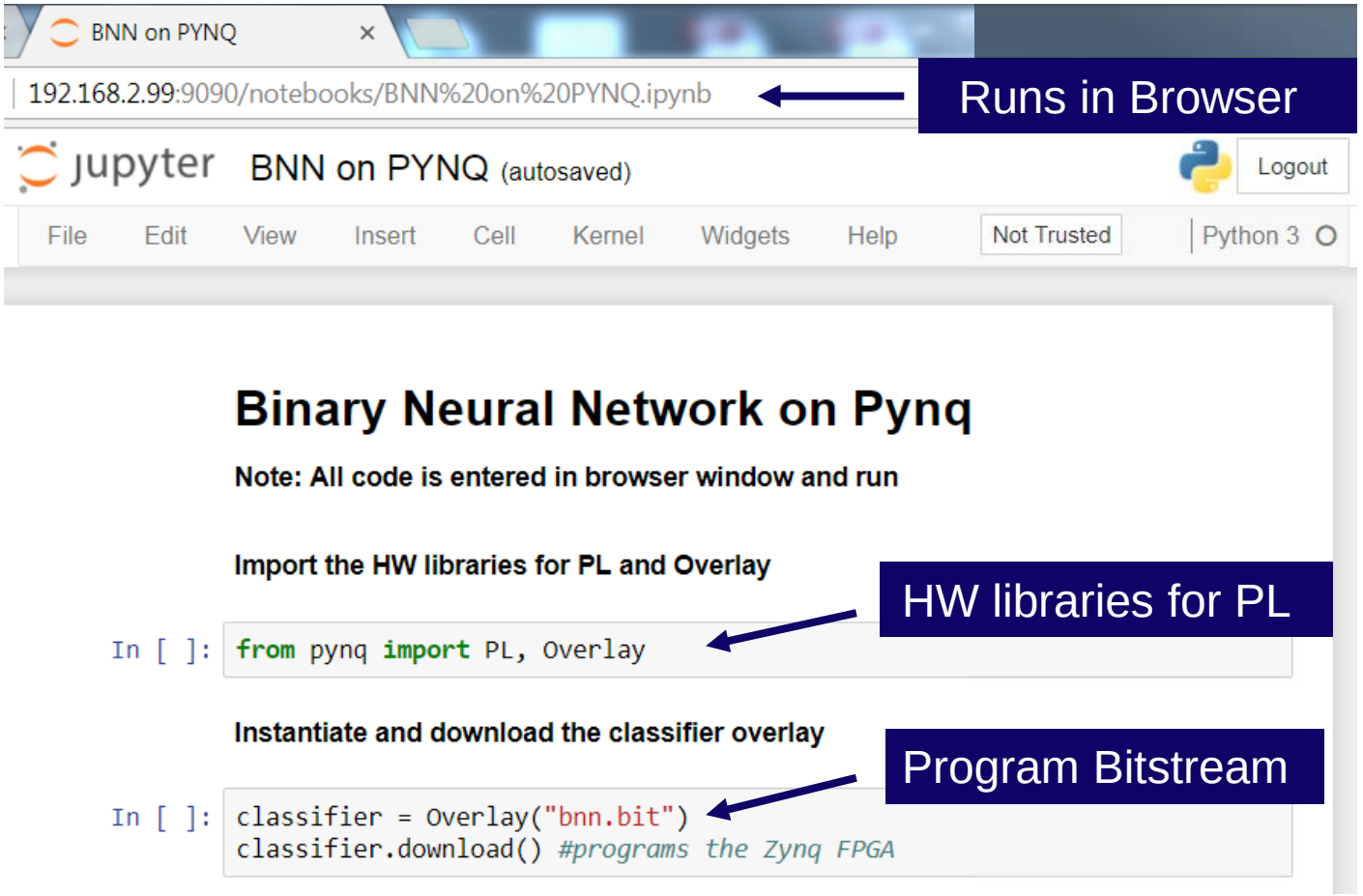
```
from time import sleep
from pynq import Overlay
from pynq.iop import PMOD_ADC, PMOD_DAC

ol = Overlay("base.bit")
ol.download()
# Writing values from 0.0V to 2.0V with step 0.1V.
dac_id = int(input("Type in the PMOD ID of the DAC (1 ~ 2): "))
adc_id = int(input("Type in the PMOD ID of the ADC (1 ~ 2): "))

dac = PMOD_DAC(dac_id)
adc = PMOD_ADC(adc_id)

for j in range(20):
    value = 0.1 * j
    dac.write(value)
    sleep(0.5)
#    readings=adc.read(1,0,0)
#    x1=readings[0]
    print("Voltage read by DAC is: {:.4f} Volts".format(adc.read(1,0,0)[0]))
```

```
6c78 3963 7367 3232 3500 6300 0b32
332f 3039 2f33 3000 6400 0931 323a
f ffff
5 6630
1 6109
0 8120
0 0020
0 0020
2 0100
3 0101
2 c100
```

Step 4:

Import the bitstream and the library in your Python scripts and program

PYNQ Neural Network Application example



PYNQ Neural Network Application example

Import SW python libraries and open image to be classified

In [7]:
```
from PIL import Image
import numpy as np

im = Image.open('deer.png')
im
```

Out[7]:

Import python SW libraries

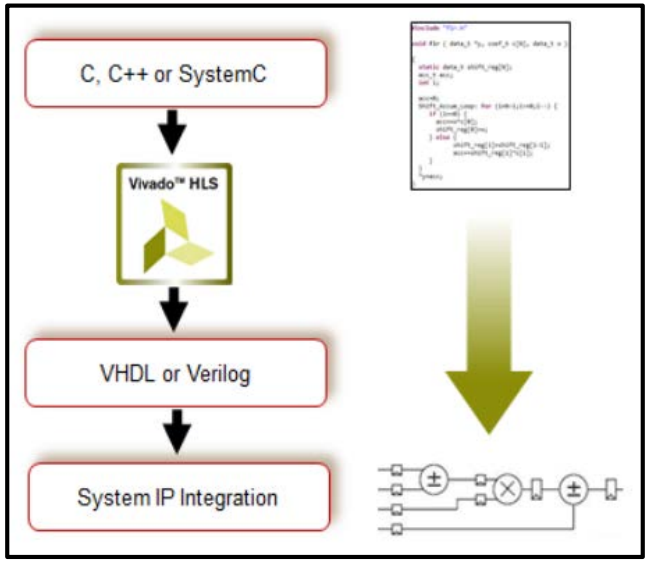
Launching BNN in hardware

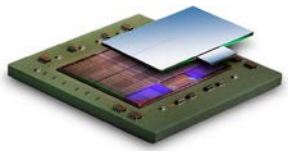
In [5]:
```
class_out=classifier.classify_image(im)
print("Class number: {0}".format(class_out))
print("Class name: {0}".format(classifier.class_name(class_out)))
```

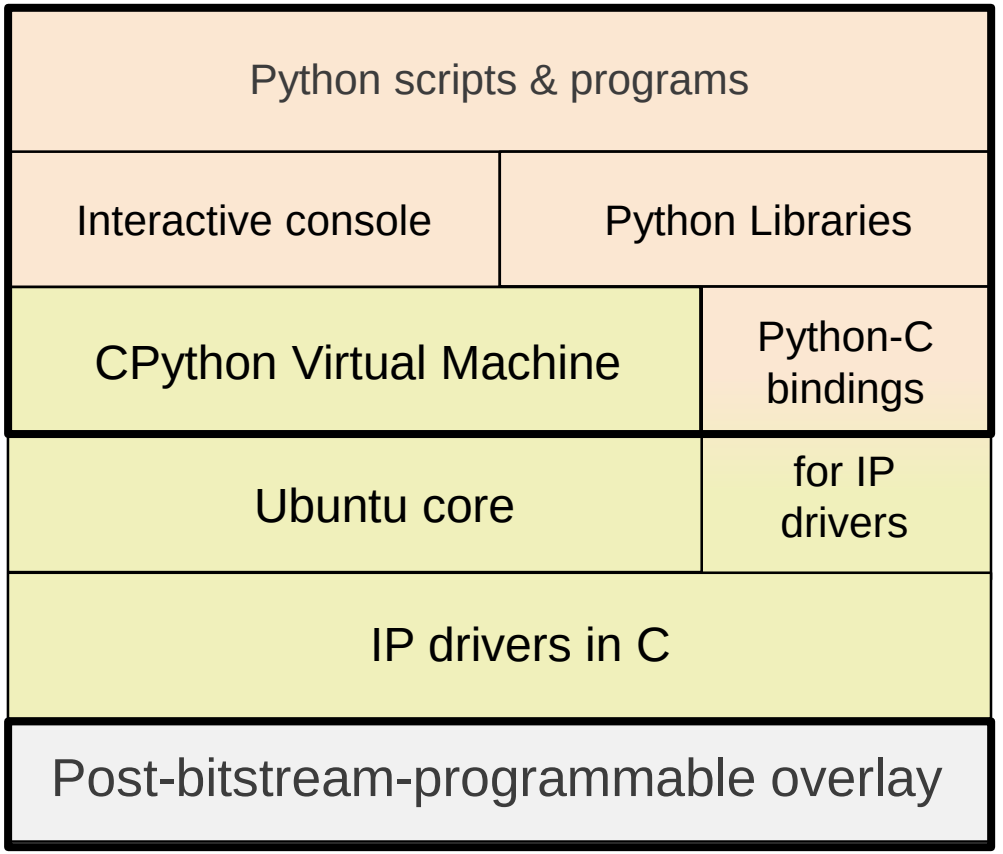
```
Inference took 2240.00 microseconds
Classification rate: 446.43 images per second
Class number: 4
Class name: Deer
```

Show results and analysis

Productivity level tools for Zynq

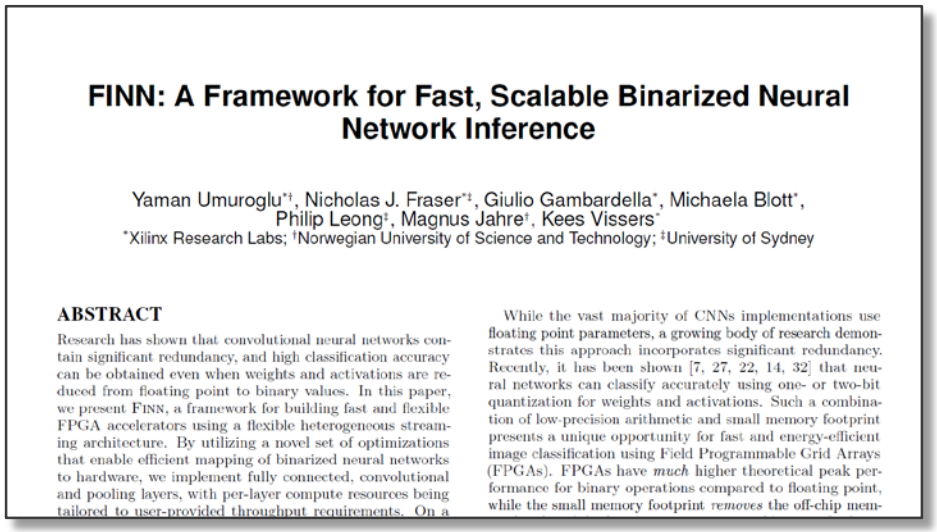






Example PYNQ Project

FINN: Binary Neural Network Overlay on PYNQ



FINN: A Framework for Fast, Scalable Binarized Neural Network Inference

Yaman Umuroglu*†, Nicholas J. Fraser*‡, Giulio Gambardella*, Michaela Blott*, Philip Leong‡, Magnus Jahre†, Kees Vissers*
*Xilinx Research Labs; †Norwegian University of Science and Technology; ‡University of Sydney

ABSTRACT

Research has shown that convolutional neural networks contain significant redundancy, and high classification accuracy can be obtained even when weights and activations are reduced from floating point to binary values. In this paper, we present FINN, a framework for building fast and flexible FPGA accelerators using a flexible heterogeneous streaming architecture. By utilizing a novel set of optimizations that enable efficient mapping of binarized neural networks to hardware, we implement fully connected, convolutional and pooling layers, with per-layer compute resources being tailored to user-provided throughput requirements. On a

While the vast majority of CNNs implementations use floating point parameters, a growing body of research demonstrates this approach incorporates significant redundancy. Recently, it has been shown [7, 27, 22, 14, 32] that neural networks can classify accurately using one- or two-bit quantization for weights and activations. Such a combination of low-precision arithmetic and small memory footprint presents a unique opportunity for fast and energy-efficient image classification using Field Programmable Grid Arrays (FPGAs). FPGAs have *much* higher theoretical peak performance for binary operations compared to floating point, while the small memory footprint *removes* the off-chip mem-

Int. Symposium on FPGAs, Feb. 2017

- Unprecedented image classification rates
- 1,000x speed-up over Raspberry Pi3

NTNU
Norwegian University of Science and Technology

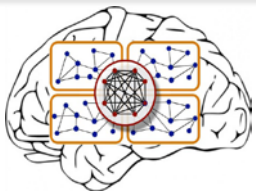
THE UNIVERSITY OF SYDNEY

https://github.com/Xilinx/BNN-PYNQ

SVHN,
Road signs,
CIFAR-10

python

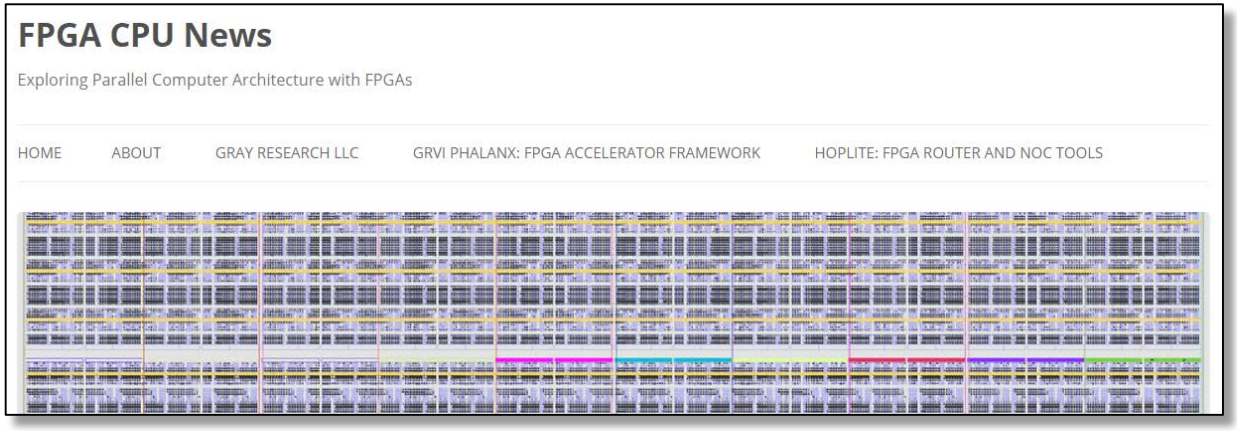
Image pre-processing in Python



Binary Neural Network in FPGA & ARM CPU

“cat”

“A high productivity tool for experienced FPGA designers”

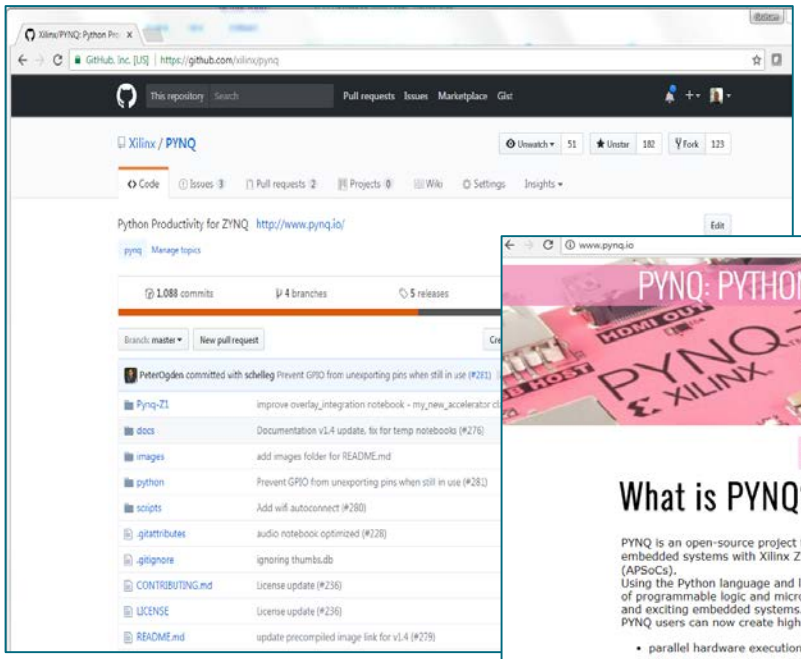


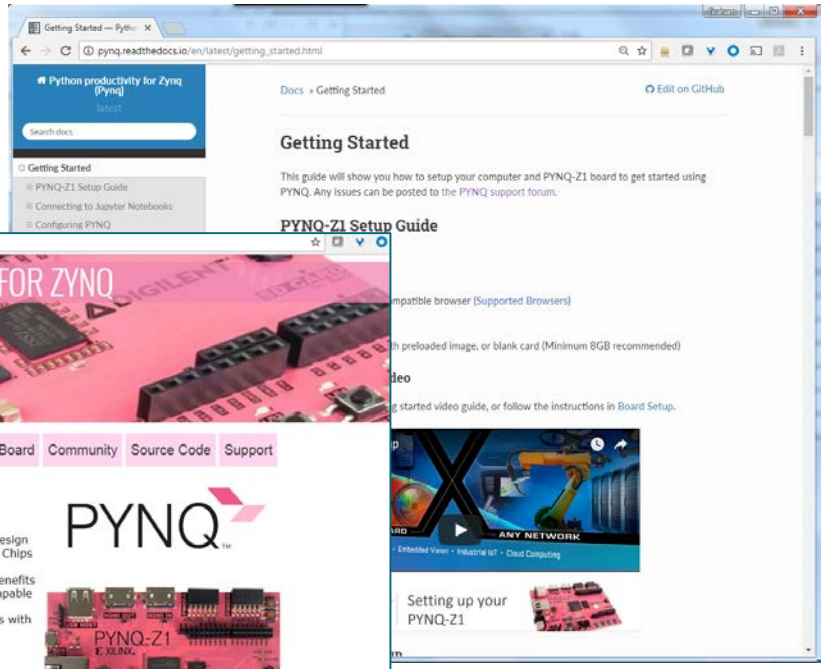
http://fpga.org/2017/09/05/pynq-as-a-high-productivity-platform-for-fpga-design-and-exploration

“Fellow FPGA designers, try Pynq. You’ll like it. Pynq makes exploring new FPGA ideas lightweight, fresh, fast, easy, *fun again.*.”

Jan Gray … feedback on implementing 80 x 32-bit RISC cores on PYNQ-Z1

PYNQ is completely open-source

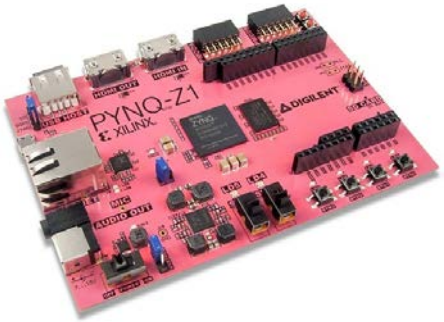




GitHub

Where to find more information
http://www.pynq.io

PYNQ Team standing by…



Build something cool

GitHub

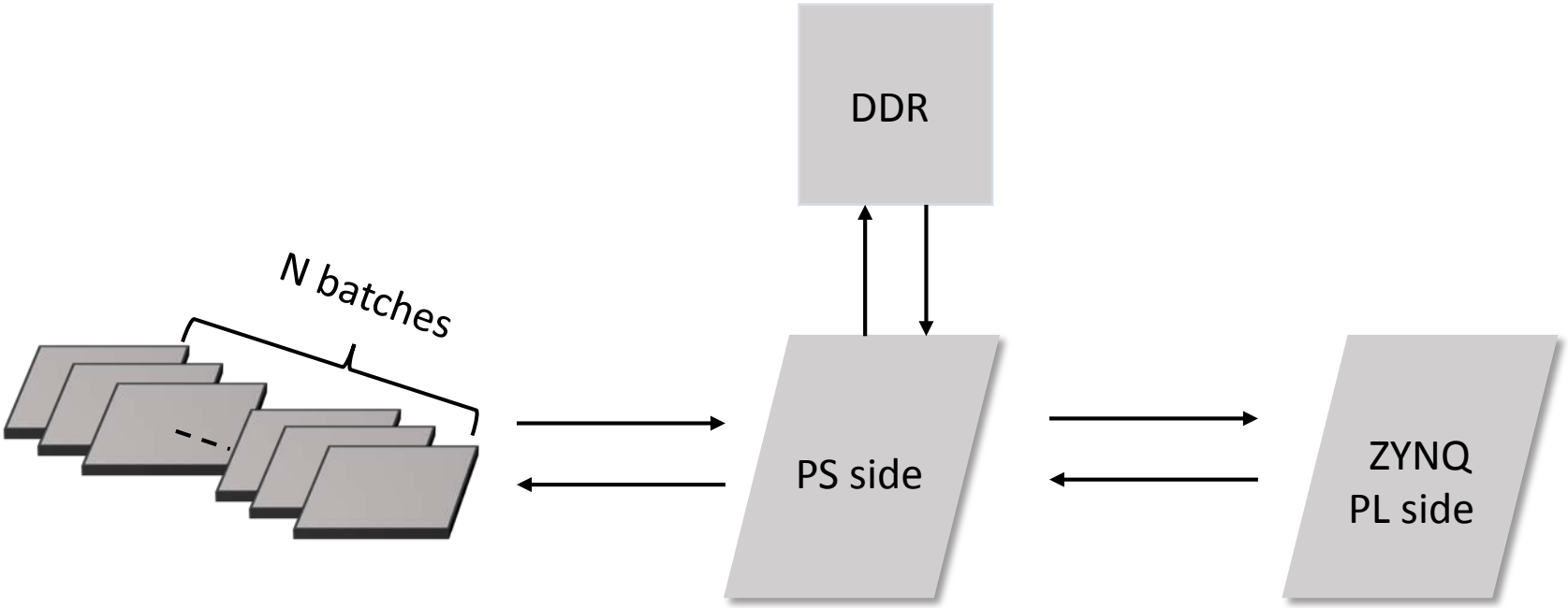
Contribute to Open Source

jupyter

Contribute Reproducible Results

If not already a member, join the PYNQ support forum
http://www.pynq.io/support.html/

DAC Contest reference design



- Timer should start before opening the images and end after PS side receives all detected coordinates.
- Write to XML can be excluded from timer.

Q & A