PyCon China
2020

PyConChina 2020
PYTHON 中国开发者大会 2020

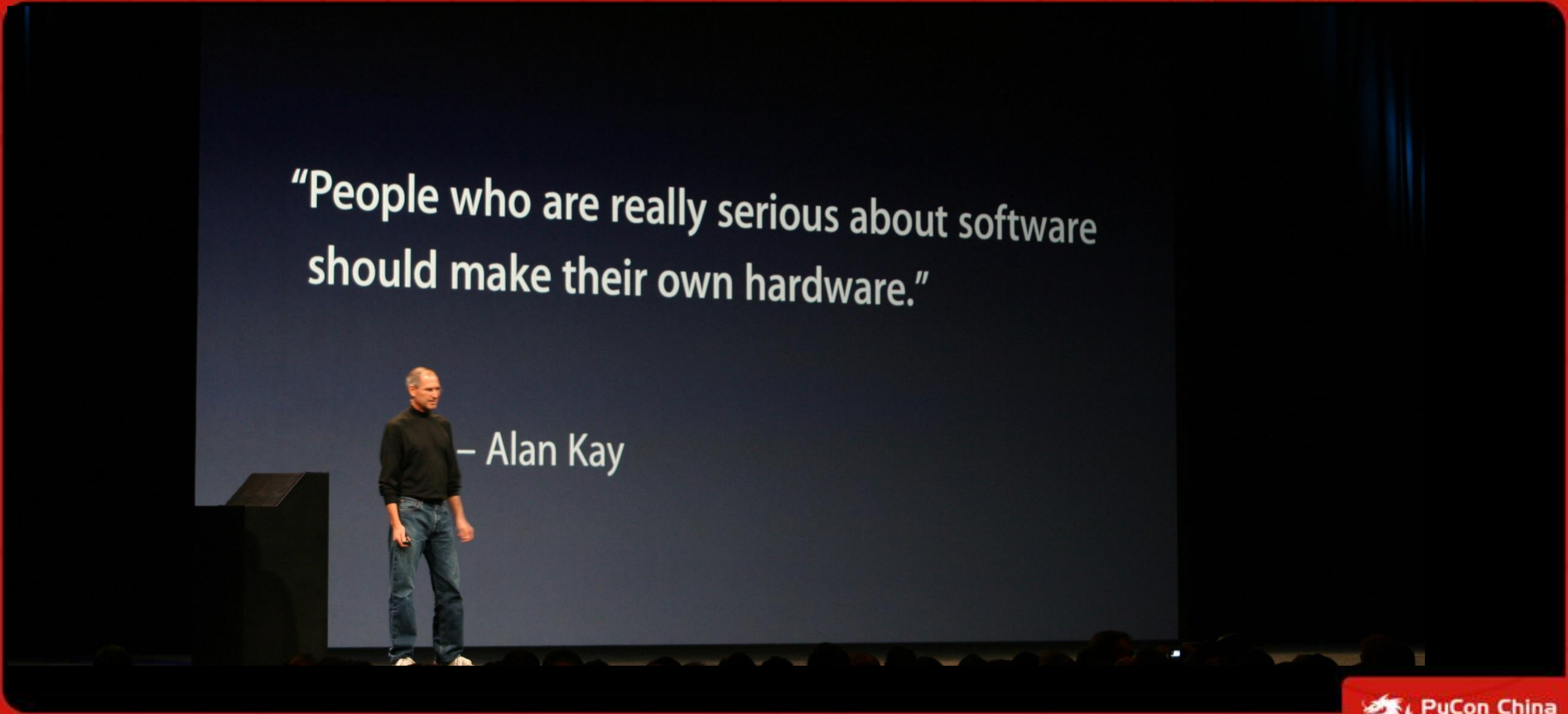
Python For Good

利用PYNQ将Python生态向嵌入式和硬件延伸

PYNQ™

陆佳华
Xilinx 学术与创新生态 高级经理

Pynq_China@Xilinx.com

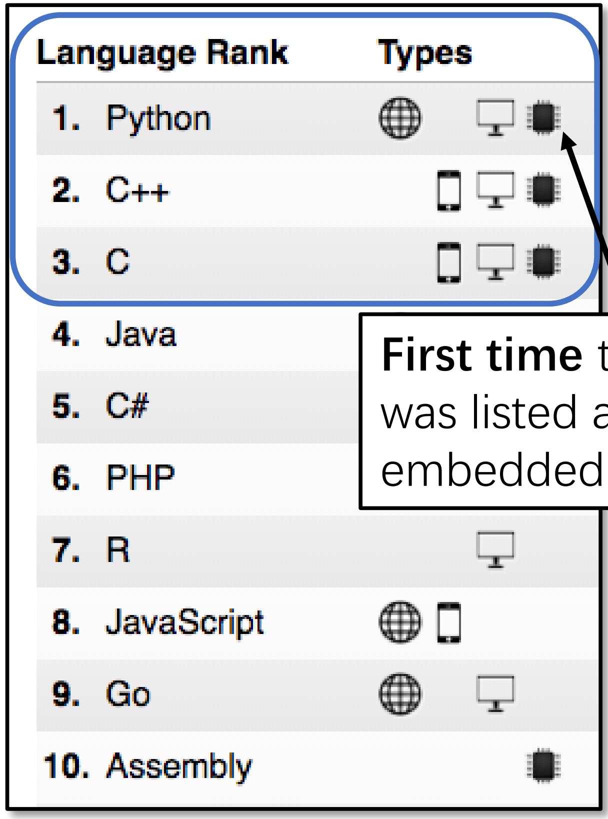


"People who are really serious about software should make their own hardware."

– Alan Kay

Python for Embedded/Edge Systems

Top Programming Languages, IEEE Spectrum, July'18



Language Rank	Types
1. Python	🌐 🖥 ■
2. C++	📱 🖥 ■
3. C	📱 🖥 ■
4. Java	
5. C#	
6. PHP	
7. R	🖥
8. JavaScript	🌐 📱
9. Go	🌐 🖥
10. Assembly	■

First time that Python was listed as an embedded language

https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages

Benefits of Python in Embedded/Edge Apps

- Millions of developers
- On-target development
- Rapid iteration cycles
- Huge ecosystem and community
- Interoperability with C/C++
- Agile hardware & software codesign
- Portable code

Python is the fastest growing language: driven by data science, AI, ML and academia

Assumptions

- Everybody knows some Python and Linux
- Have heard about Raspberry Pi and Arduino
- May be familiar with Jupyter notebooks and JupyterLab
- Are interested in exploiting cool hardware
- Keen to learn more awesome things that you can do with Python!

Outline

- Motivation: Platform Evolution
- Opportunity: Zynq Programmable Platforms
- Inspiration: Python and Jupyter
- PYNQ Framework
- Cases study: PYNQ in Action
- Next steps

Platform Evolution

Raspberry PI

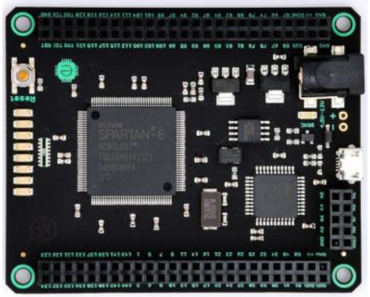
Desktop Linux on Arm Microprocessors

Arduino

Low-level, ‘bit-banging’ microcontroller

FPGAs

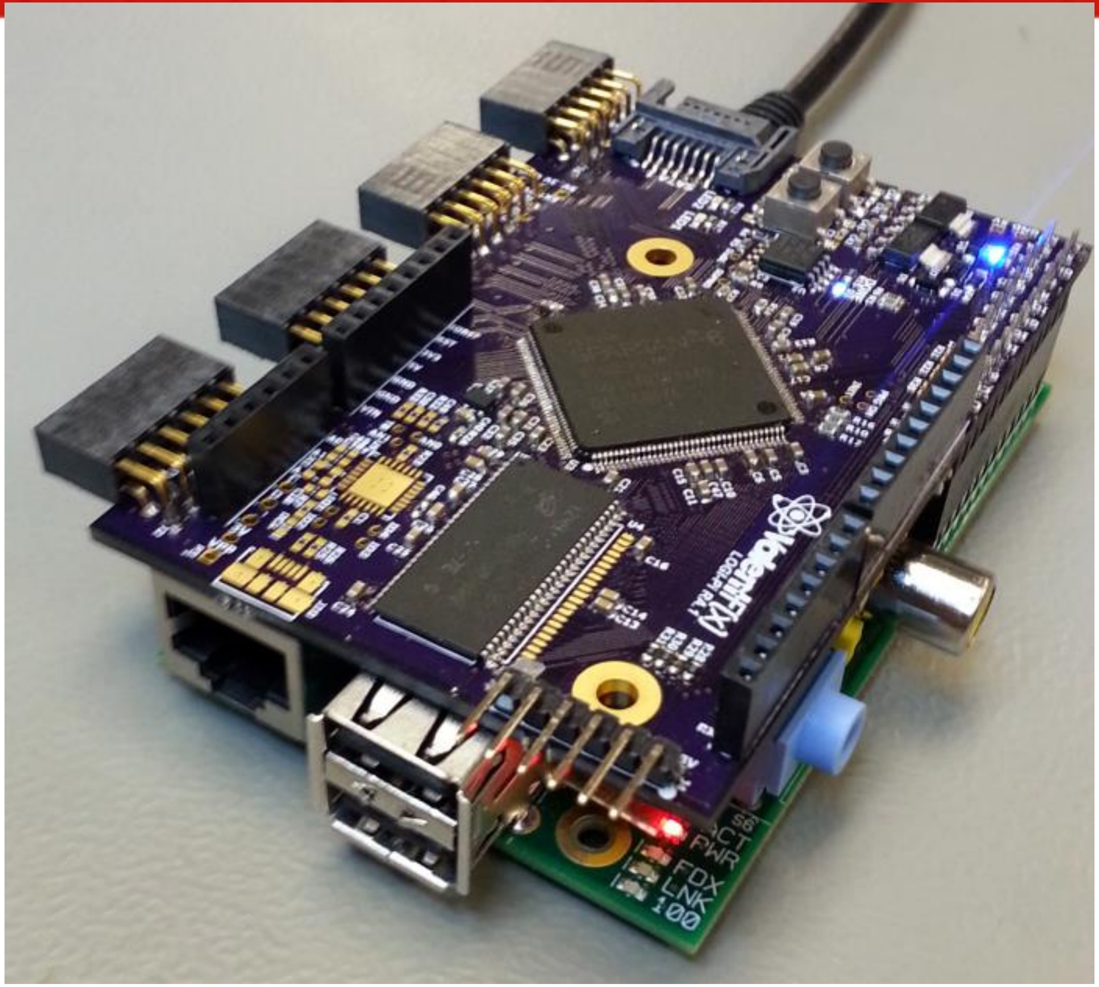
FPGA



Field Programmable Gate Arrays

Fast, parallel, customizable logic

Example of a Raspberry Pi and FPGA Hat

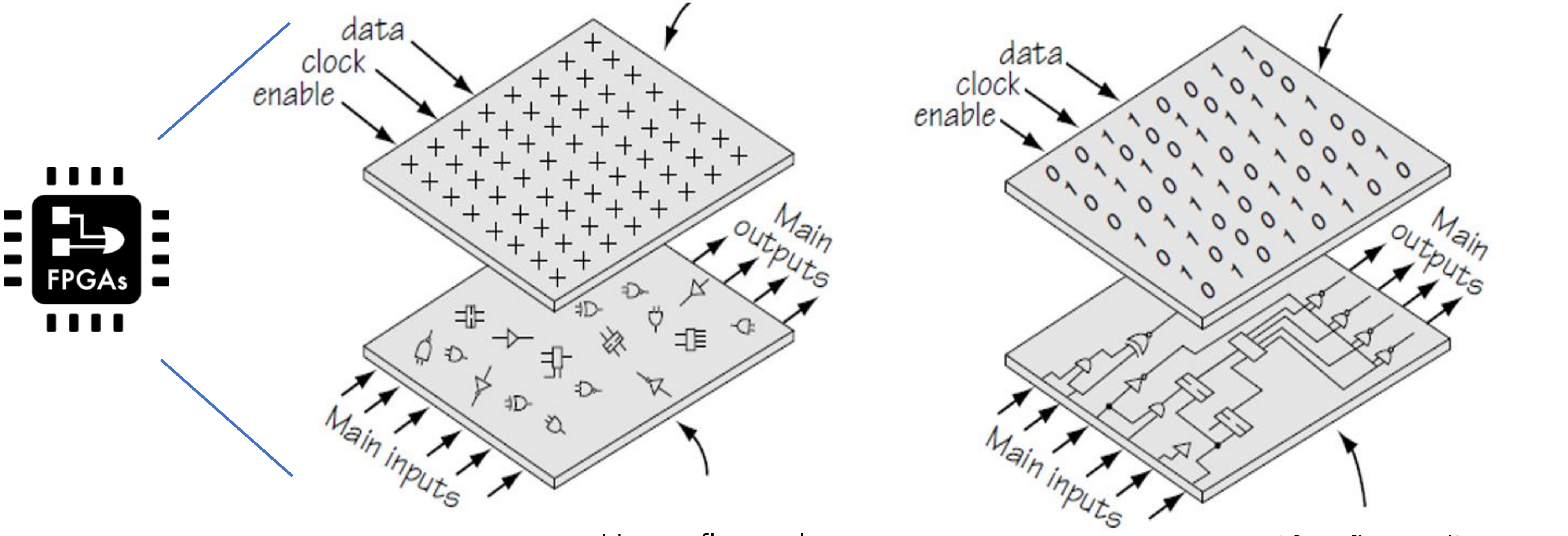


http://linuxgizmos.com/beaglebone-raspberry-pi-gain-fpga-expansion-boards/

Field Programmable Gate Arrays (FPGAs)

Unprogrammed
configuration memory

'Programmed'
configuration memory



Unconfigured
logic circuit

'Configured'
logic circuit

Credit: 'Bebop to the Boolean Boogie: An Unconventional Guide to Electronics'

Zynq: Integrating Microprocessors, Microcontrollers and Programmable Logic

Systems-on-Chip integration

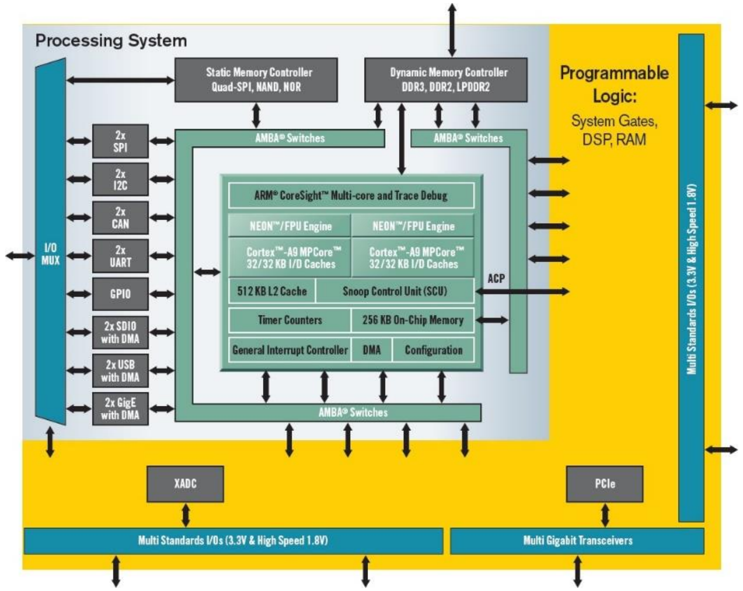
Zynq Programmable Platform integrate

- Arm microprocessors
- Programmable logic (FPGA)
- High-speed, programmable IO
- As many ‘soft’ microcontrollers as needed
- Fast connections between components

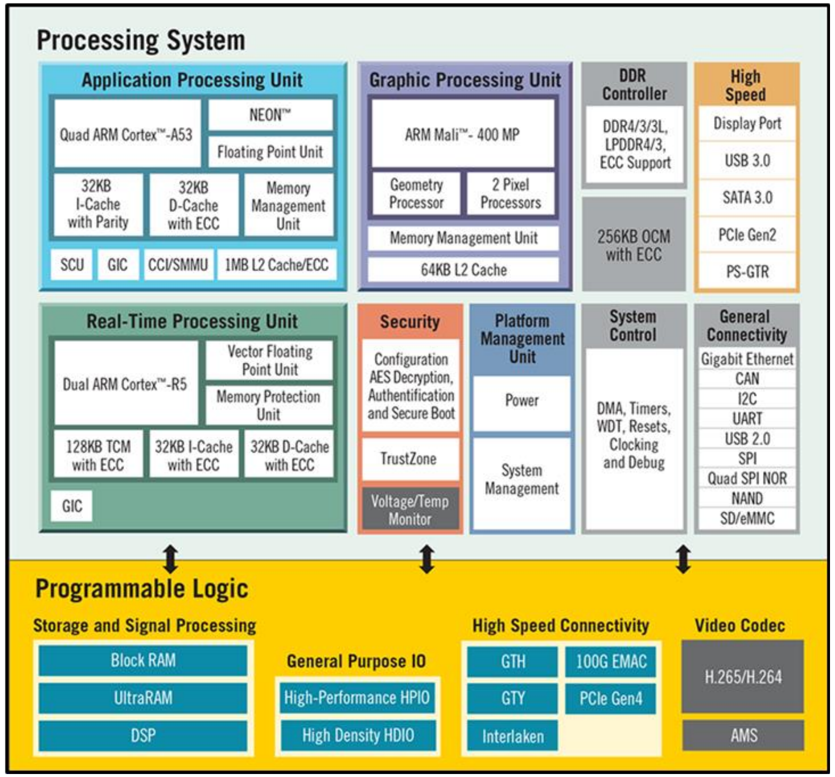
ZYNQ and ZYNQ UltraSCALE$^+$

Best-in-class, All Programmable SoCs

ZYNQ 7000

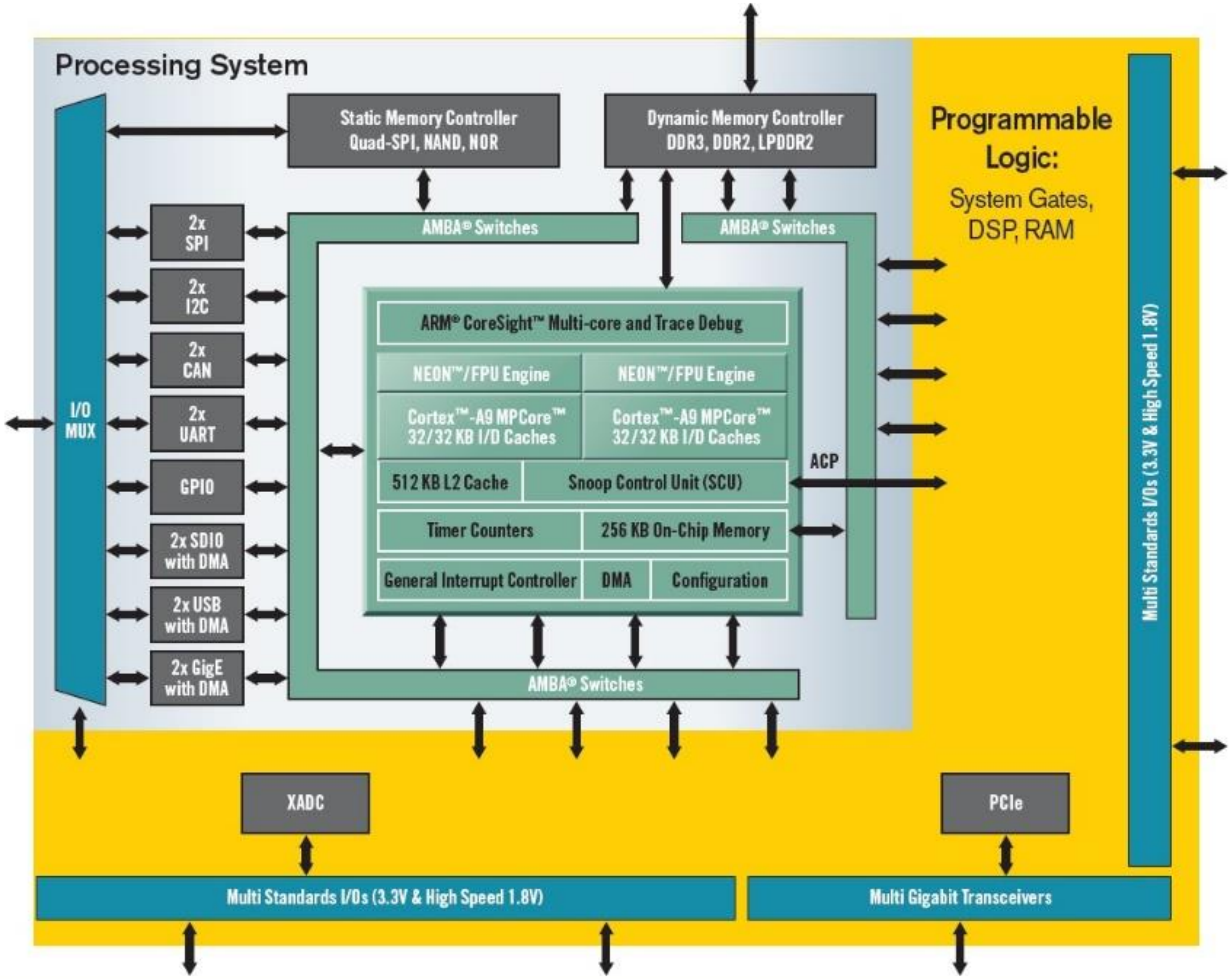


ZYNQ UltraSCALE$^+$



FPGAs and tightly-integrated CPUs enable entirely new opportunities

ZYNQ-7000


Processing System
Static Memory Controller
Quad-SPI, NAND, NOR
Dynamic Memory Controller
DDR3, DDR2, LPDDR2
Programmable Logic:
System Gates, DSP, RAM
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
I/O MUX
AMBA® Switches
AMBA® Switches
ARM® CoreSight™ Multi-core and Trace Debug
NEON™/FPU Engine
NEON™/FPU Engine
Cortex™-A9 MPCore™ 32/32 KB I/D Caches
Cortex™-A9 MPCore™ 32/32 KB I/D Caches
512 KB L2 Cache
Snoop Control Unit (SCU)
ACP
Timer Counters
256 KB On-Chip Memory
General Interrupt Controller
DMA
Configuration
AMBA® Switches
XADC
PCIe
Multi Standards I/Os (3.3V & High Speed 1.8V)
Multi Gigabit Transceivers
Multi Standards I/Os (3.3V & High Speed 1.8V)

Platform Comparison Summary

	Raspberry PI	Arduino	Zynq
Arm Application Microprocessor	YES	NO	YES
Real-time Microcontroller	NO	YES	YES*
Integrated Programmable Logic	NO	NO	YES
Linux	YES	NO	YES
CPython Ecosystem	YES	NO**	YES

* Multiple soft microcontrollers in programmable logic

** More limited MicroPython and CircuitPython options are available

Zynq in Embedded/Edge Applications

Typical Application Areas

> Hardware-accelerated algorithms

> Precision robotics

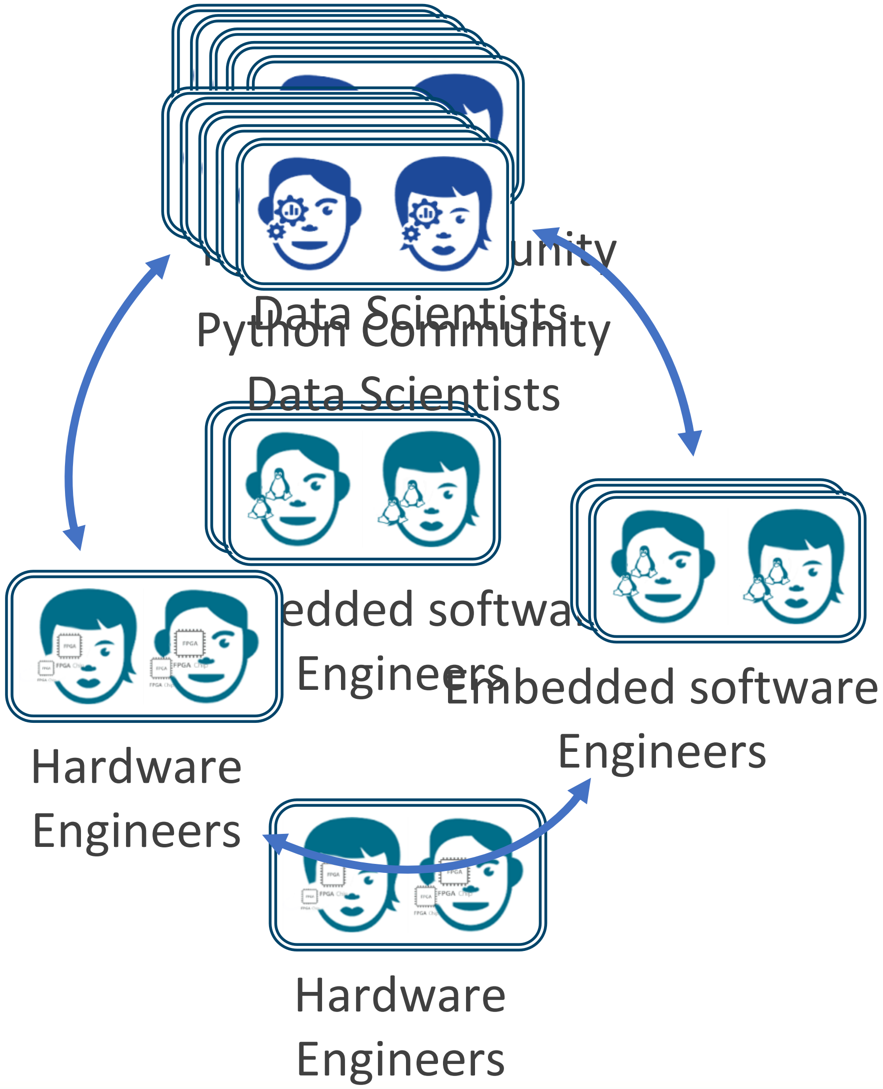
> Real-time, high-resolution video processing

> State-of-the-art instrumentation

> Unique, highly-differentiated designs in research and spin-offs

> Teaching: logic design, computer architecture, digital signal processing, control, projects

ython for Embedded and Edge Applications



Python Community

Data Scientists

Embedded software Engineers

Hardware Engineers

New users can be Python programmers, Data Scientists and domain experts of all kinds

PYNQ

A Pythonic Framework that gives Python developers access to the benefits of programmable platforms

AND

A Pythonic methodology for hardware designers to:

- *make them more productive*
- *make their designs accessible to more people*

What is PYNQ?

- It is not a product
- It is an open-source framework which enables improving productivity on Xilinx ZYNQ-based design and verification by using other appropriate open-source resources
  - GitHub for repository
  - JupyterLab and Jupyter Notebooks for data and code entry, execution of code, and viewing output
  - Readthedocs for documentation
  - Linux as underlying OS
  - Python as a coding language
- Of course, it uses Xilinx FPGAs – ZYNQ and non-ZYNQ

PYNQ Python Productivity for Zynq

Jupyter notebooks,
browser-based interface

PYNQ enables JupyterLab
on Zynq and ZU+

Jupyter web server

IPython kernel

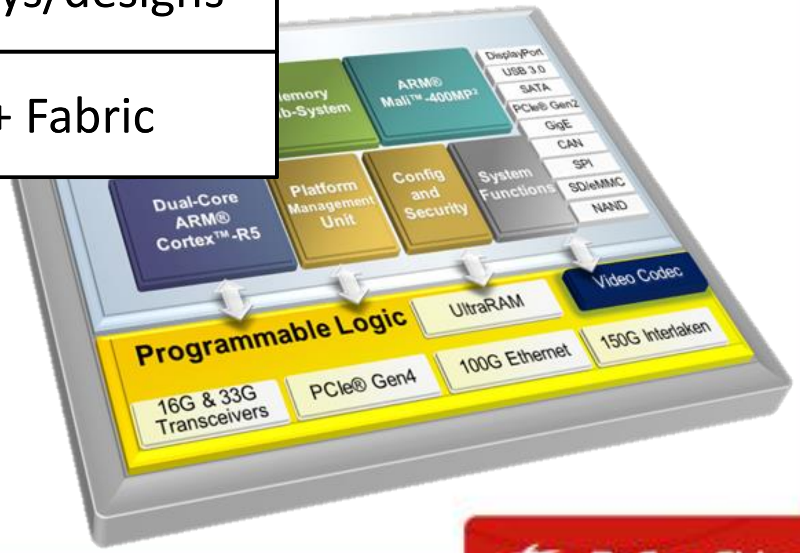
Ubuntu-based Linux | Overlays/designs

ARM A9 / A53 | ZU+ Fabric

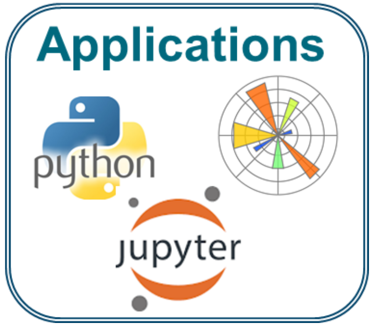
Hardware C-drivers wrapped in Python packages

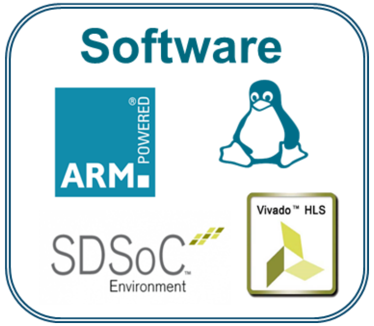
Most competitive, open
and advanced GUI

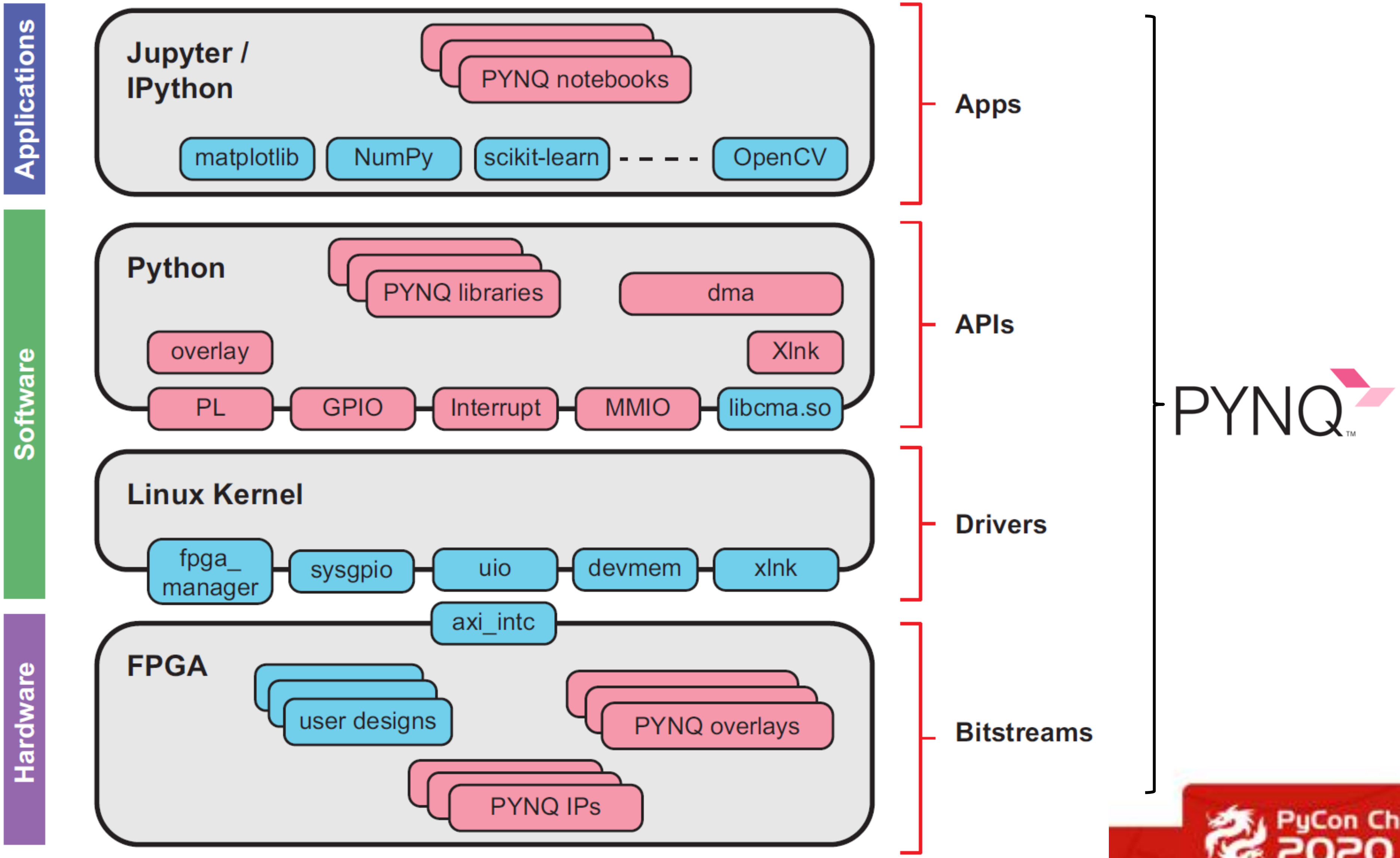
SVG HTML CSS JavaScript



PYNQ™ is a Framework





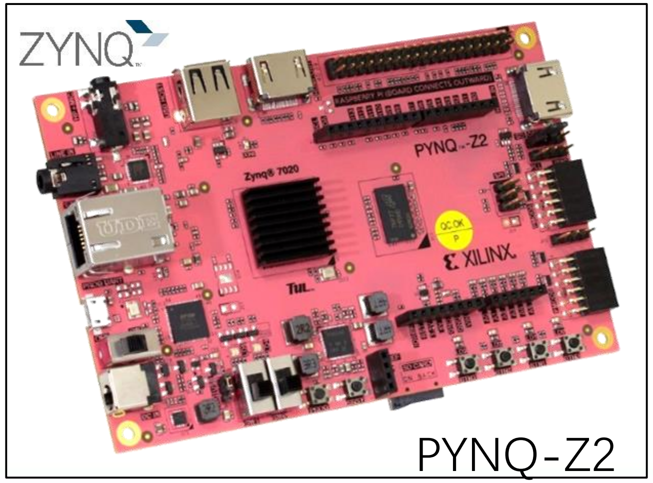


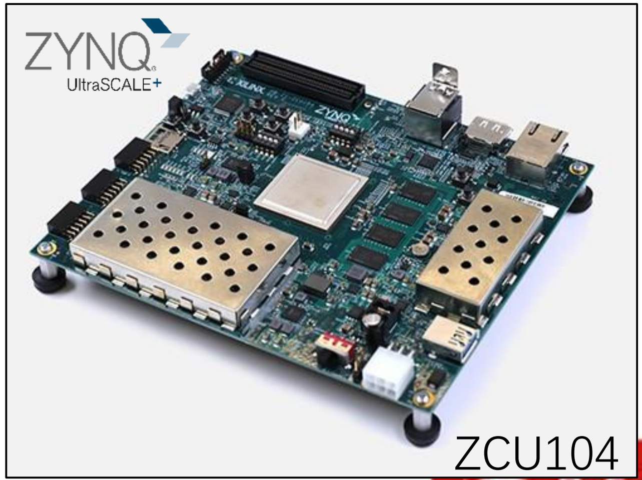
PYNQ-enabled boards

- **Python productivity for Zynq**
  - Open source
  - Build image for other Zynq boards

- **Downloadable SD card image**
  - Zynq 7000
    - PYNQ-Z1 (Digilent)
    - PYNQ-Z2 (TUL)
  - Zynq Ultrascale+
    - ZCU104 (Xilinx)

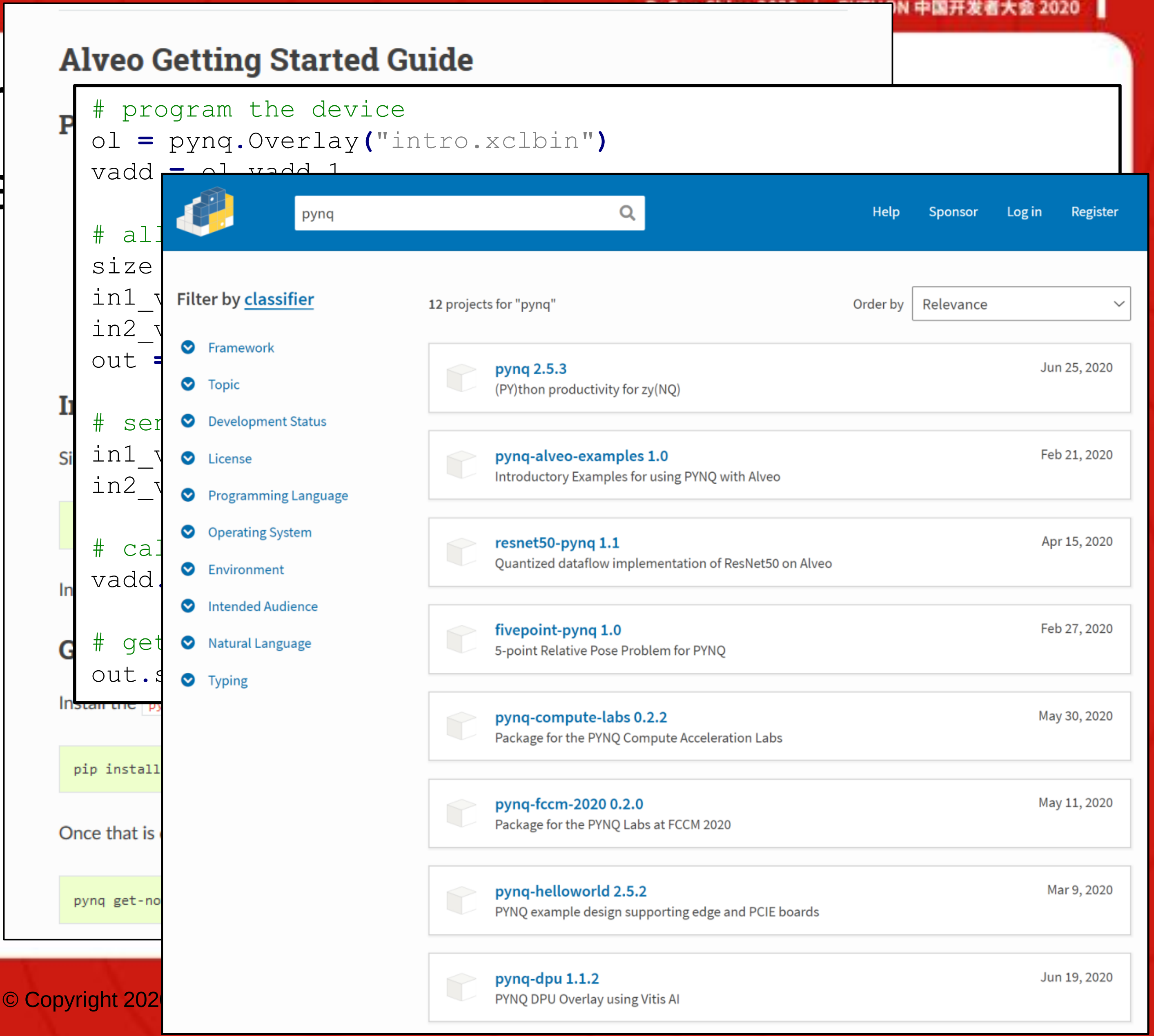
PYNQ-Z1


PYNQ-Z2


ZCU104

PYNQ-enabled non-zynq board: Alveo

- PYNQ runs on Alveo Acceler
- High-level APIs for hardware
- Simplify design installation



Alveo Getting Started Guide

P

```
# program the device
ol = pynq.Overlay("intro.xclbin")
vadd = ol.vadd_1

# al
size
in1_
in2_
out =

# se
in1_
in2_

# ca
vadd.

# get
out.s
```

I

Si

In

G

Install the py

```
pip install
```

Once that is

```
pynq get-no
```

Filter by classifier

- Framework
- Topic
- Development Status
- License
- Programming Language
- Operating System
- Environment
- Intended Audience
- Natural Language
- Typing

12 projects for "pynq"

Order by Relevance

pynq 2.5.3 — Jun 25, 2020
(PY)thon productivity for zy(NQ)

pynq-alveo-examples 1.0 — Feb 21, 2020
Introductory Examples for using PYNQ with Alveo

resnet50-pynq 1.1 — Apr 15, 2020
Quantized dataflow implementation of ResNet50 on Alveo

fivepoint-pynq 1.0 — Feb 27, 2020
5-point Relative Pose Problem for PYNQ

pynq-compute-labs 0.2.2 — May 30, 2020
Package for the PYNQ Compute Acceleration Labs

pynq-fccm-2020 0.2.0 — May 11, 2020
Package for the PYNQ Labs at FCCM 2020

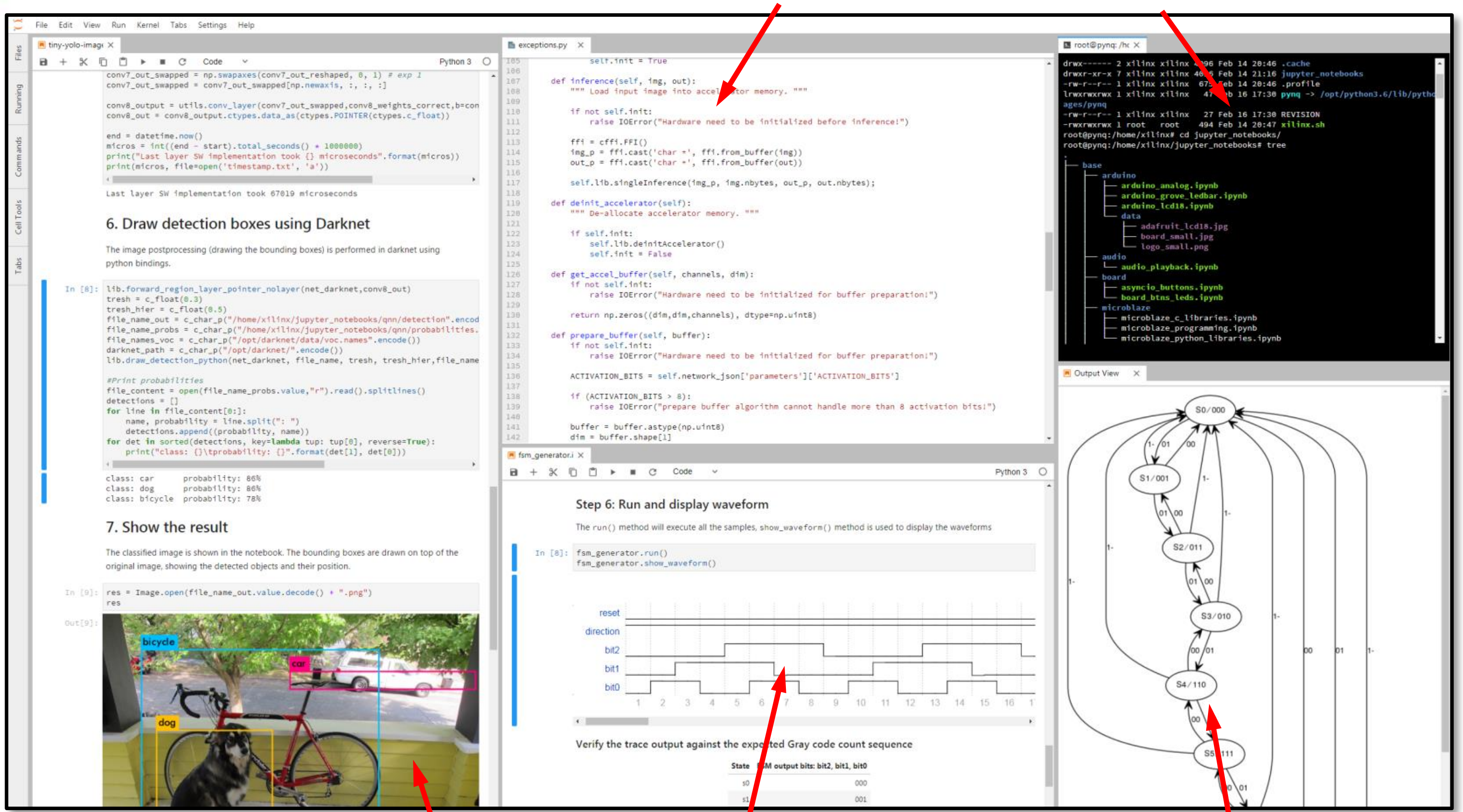
pynq-helloworld 2.5.2 — Mar 9, 2020
PYNQ example design supporting edge and PCIE boards

pynq-dpu 1.1.2 — Jun 19, 2020
PYNQ DPU Overlay using Vitis AI

Jupyter Notebooks to JupyterLab IDE

Code editor

Terminal



```
conv7_out_swapped = np.swapaxes(conv7_out_reshaped, 0, 1) # exp 1
conv7_out_swapped = conv7_out_swapped[np.newaxis, :, :, :]

conv8_output = utils.conv_layer(conv7_out_swapped,conv8_weights_correct,b=con
conv8_out = conv8_output.ctypes.data_as(ctypes.POINTER(ctypes.c_float))

end = datetime.now()
micros = int((end - start).total_seconds() * 1000000)
print("Last layer SW implementation took {} microseconds".format(micros))
print(micros, file=open('timestamp.txt', 'a'))
```

```
Last layer SW implementation took 67019 microseconds
```

6. Draw detection boxes using Darknet

The image postprocessing (drawing the bounding boxes) is performed in darknet using python bindings.

In [8]:

```
lib.forward_region_layer_pointer_nolayer(net_darknet,conv8_out)
tresh = c_float(0.3)
tresh_hier = c_float(0.5)
file_name_out = c_char_p("/home/xilinx/jupyter_notebooks/qnn/detection".encod
file_name_probs = c_char_p("/home/xilinx/jupyter_notebooks/qnn/probabilities.
file_names_voc = c_char_p("/opt/darknet/data/voc.names".encode())
darknet_path = c_char_p("/opt/darknet/".encode())
lib.draw_detection_python(net_darknet, file_name, tresh, tresh_hier,file_name

#Print probabilities
file_content = open(file_name_probs.value,"r").read().splitlines()
detections = []
for line in file_content[0:]:
    name, probability = line.split(": ")
    detections.append((probability, name))
for det in sorted(detections, key=lambda tup: tup[0], reverse=True):
    print("class: {}\tprobability: {}".format(det[1], det[0]))
```

```
class: car      probability: 86%
class: dog      probability: 86%
class: bicycle  probability: 78%
```

7. Show the result

The classified image is shown in the notebook. The bounding boxes are drawn on top of the original image, showing the detected objects and their position.

In [9]:

```
res = Image.open(file_name_out.value.decode() + ".png")
res
```

Out[9]:

```
        self.init = True

    def inference(self, img, out):
        """ Load input image into accelerator memory. """

        if not self.init:
            raise IOError("Hardware need to be initialized before inference!")

        ffi = cffi.FFI()
        img_p = ffi.cast('char *', ffi.from_buffer(img))
        out_p = ffi.cast('char *', ffi.from_buffer(out))

        self.lib.singleInference(img_p, img.nbytes, out_p, out.nbytes);

    def deinit_accelerator(self):
        """ De-allocate accelerator memory. """

        if self.init:
            self.lib.deinitAccelerator()
            self.init = False

    def get_accel_buffer(self, channels, dim):
        if not self.init:
            raise IOError("Hardware need to be initialized for buffer preparation!")

        return np.zeros((dim,dim,channels), dtype=np.uint8)

    def prepare_buffer(self, buffer):
        if not self.init:
            raise IOError("Hardware need to be initialized for buffer preparation!")

        ACTIVATION_BITS = self.network_json['parameters']['ACTIVATION_BITS']

        if (ACTIVATION_BITS > 8):
            raise IOError("prepare buffer algorithm cannot handle more than 8 activation bits!")

        buffer = buffer.astype(np.uint8)
        dim = buffer.shape[1]
```

```
drwx------ 2 xilinx xilinx 4096 Feb 14 20:46 .cache
drwxr-xr-x 7 xilinx xilinx 4096 Feb 14 21:16 jupyter_notebooks
-rw-r--r-- 1 xilinx xilinx  675 Feb 14 20:46 .profile
lrwxrwxrwx 1 xilinx xilinx   47 Feb 16 17:30 pynq -> /opt/python3.6/lib/pytho
ages/pynq
-rw-r--r-- 1 xilinx xilinx   27 Feb 16 17:30 REVISION
-rwxrwxrwx 1 root   root    494 Feb 14 20:47 xilinx.sh
root@pynq:/home/xilinx# cd jupyter_notebooks/
root@pynq:/home/xilinx/jupyter_notebooks# tree
.
├── base
│   ├── arduino
│   │   ├── arduino_analog.ipynb
│   │   ├── arduino_grove_ledbar.ipynb
│   │   ├── arduino_lcd18.ipynb
│   │   └── data
│   │       ├── adafruit_lcd18.jpg
│   │       ├── board_small.jpg
│   │       └── logo_small.png
│   ├── audio
│   │   └── audio_playback.ipynb
│   ├── board
│   │   ├── asyncio_buttons.ipynb
│   │   └── board_btns_leds.ipynb
│   ├── microblaze
│   │   ├── microblaze_c_libraries.ipynb
│   │   ├── microblaze_programming.ipynb
│   │   └── microblaze_python_libraries.ipynb
```

Step 6: Run and display waveform

The run() method will execute all the samples, show_waveform() method is used to display the waveforms

In [8]:

```
fsm_generator.run()
fsm_generator.show_waveform()
```

Verify the trace output against the expected Gray code count sequence

State	FSM output bits: bit2, bit1, bit0
s0	000
s1	001

Jupyter notebooks

Visualization

Next-generation Integrated Development Environment

Browser-based GUI

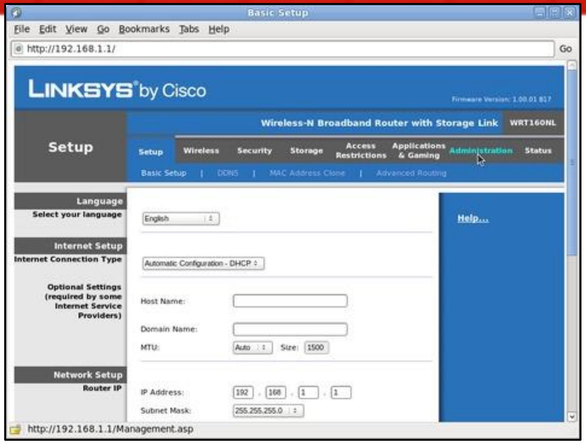
Multiple re-sizable frames in one browser window

Completely extensible

Jupyter Notebooks are one of many plug-ins in JupyterLab

Embedded Configuration Portals …

to Embedded IDEs



http://192.168.1.1/

Firmware Version: 1.00.01 817

Wireless-N Broadband Router with Storage Link WRT160NL

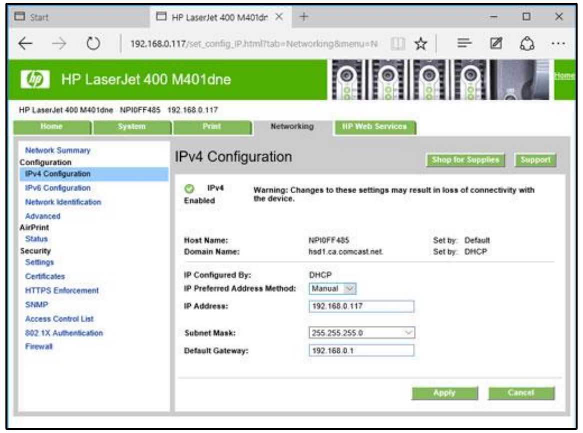
Language
Select your language: English

Internet Setup
Internet Connection Type: Automatic Configuration - DHCP

Optional Settings (required by some Internet Service Providers)
Host Name:
Domain Name:
MTU: Auto Size: 1500

Network Setup
Router IP
IP Address: 192 . 168 . 1 . 1
Subnet Mask: 255.255.255.0

http://192.168.1.1/Management.asp



192.168.0.117/set_config_IP.html?tab=Networking&menu=N

HP LaserJet 400 M401dne

HP LaserJet 400 M401dne NPI0FF485 192.168.0.117

IPv4 Configuration

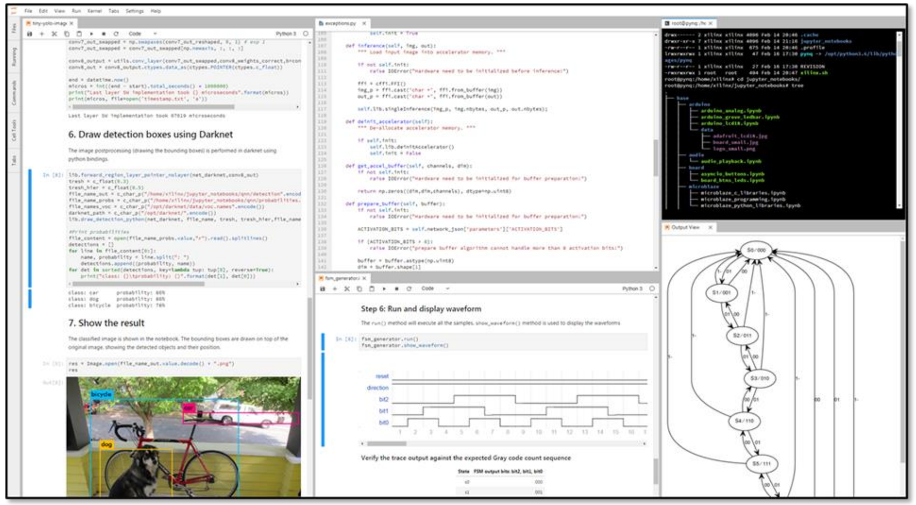
IPv4 Enabled

Warning: Changes to these settings may result in loss of connectivity with the device.

Host Name: NPI0FF485 Set by: Default
Domain Name: hsd1.ca.comcast.net Set by: DHCP

IP Configured By: DHCP
IP Preferred Address Method: Manual
IP Address: 192.168.0.117
Subnet Mask: 255.255.255.0
Default Gateway: 192.168.0.1

PYNQ uses Jupyter's server architecture to host a browser interface to an integrated development environment (IDE) on Zynq



6. Draw detection boxes using Darknet

The image postprocessing (drawing the bounding boxes) is performed in darknet using python bindings.

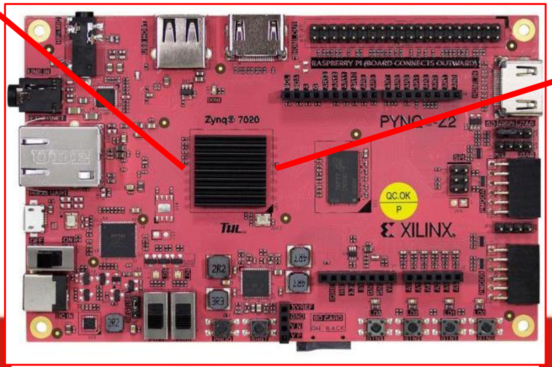
7. Show the result

The classified image is shown in the notebook. The bounding boxes are drawn on top of the original image, showing the detected objects and their position.

Step 6: Run and display waveform

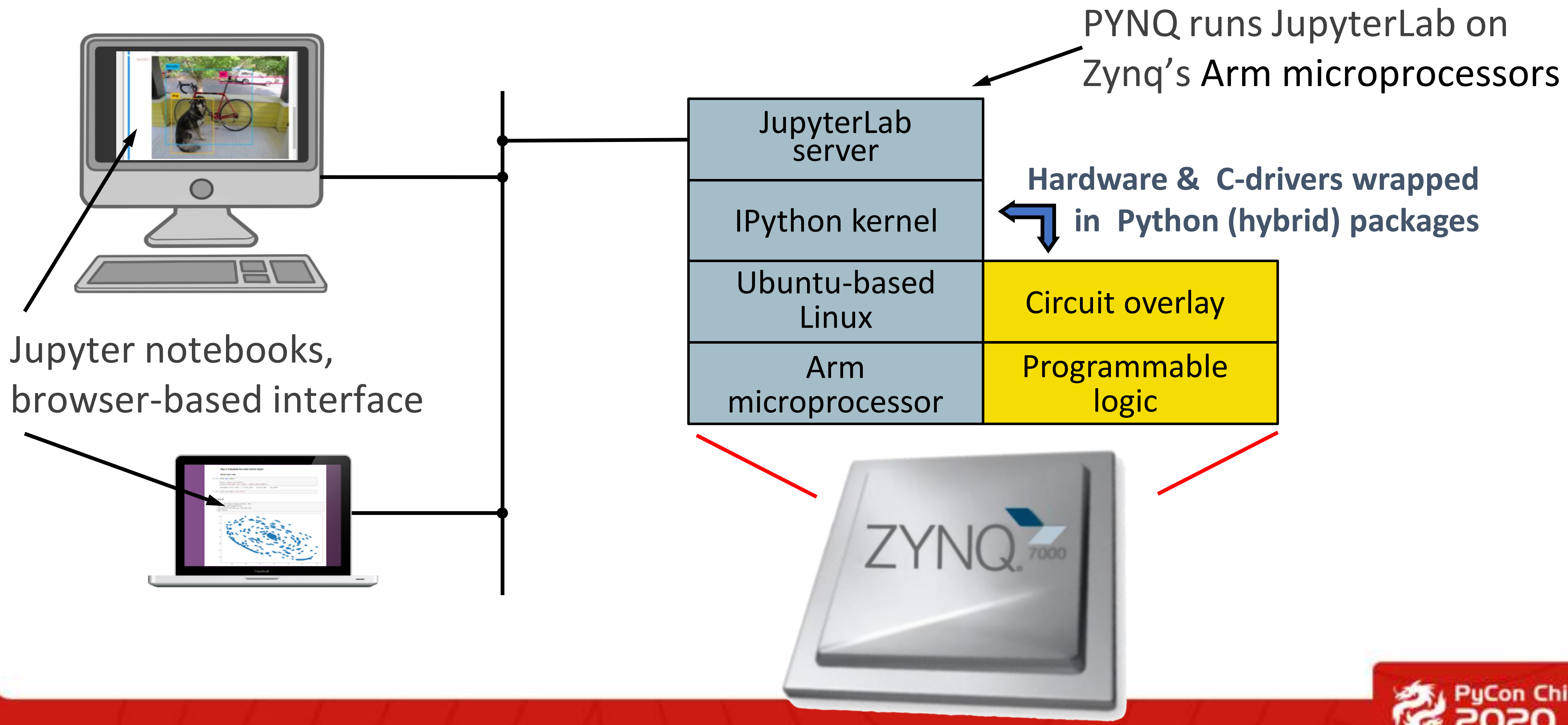
The run() method will execute all the samples, show_waveform() method is used to display the waveforms

Verify the trace output against the expected Gray code count sequence



We are familiar with embedded servers hosting browser interfaces for product configuration

PYNQ Wraps FPGA Circuit & Drivers in Python Package



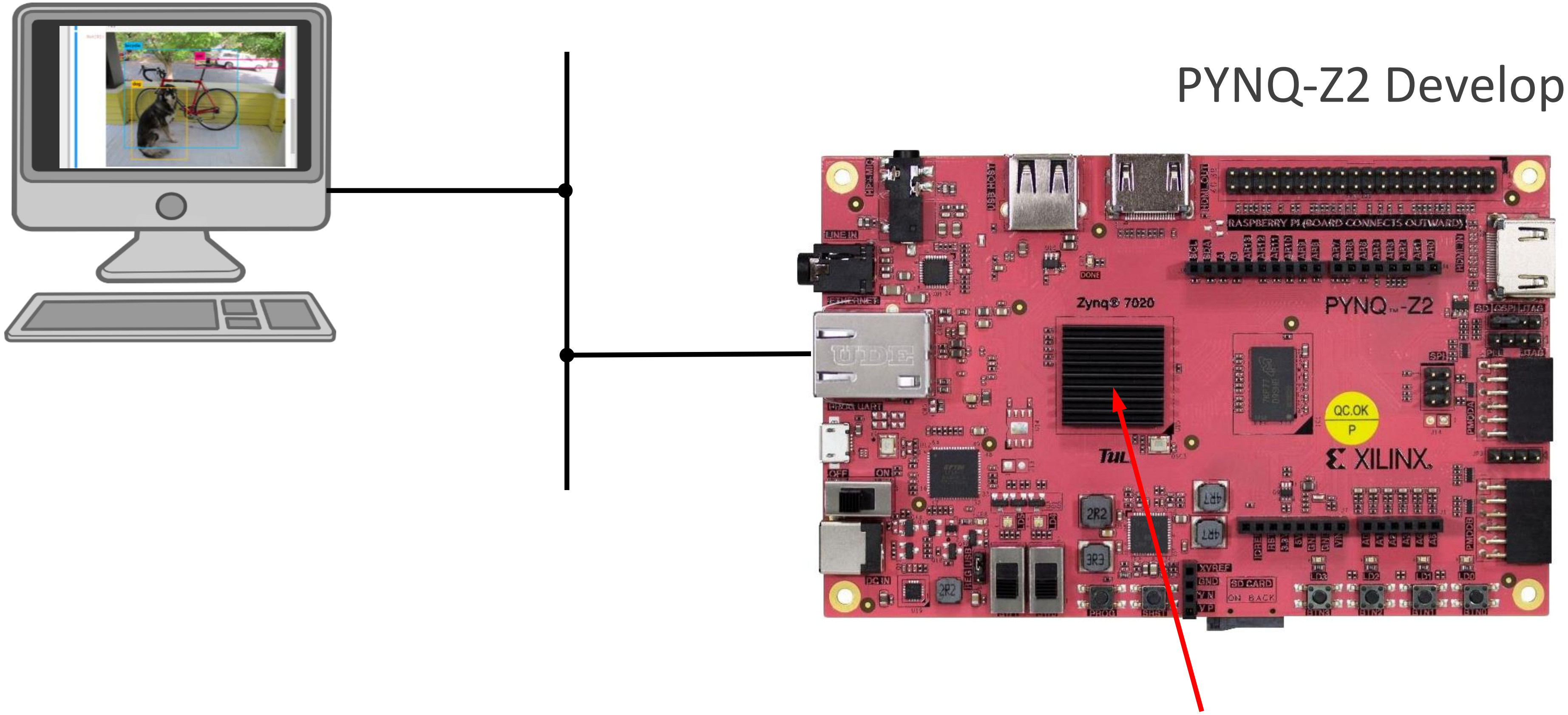
PYNQ runs JupyterLab on Zynq's Arm microprocessors

Hardware & C-drivers wrapped in Python (hybrid) packages

Jupyter notebooks, browser-based interface

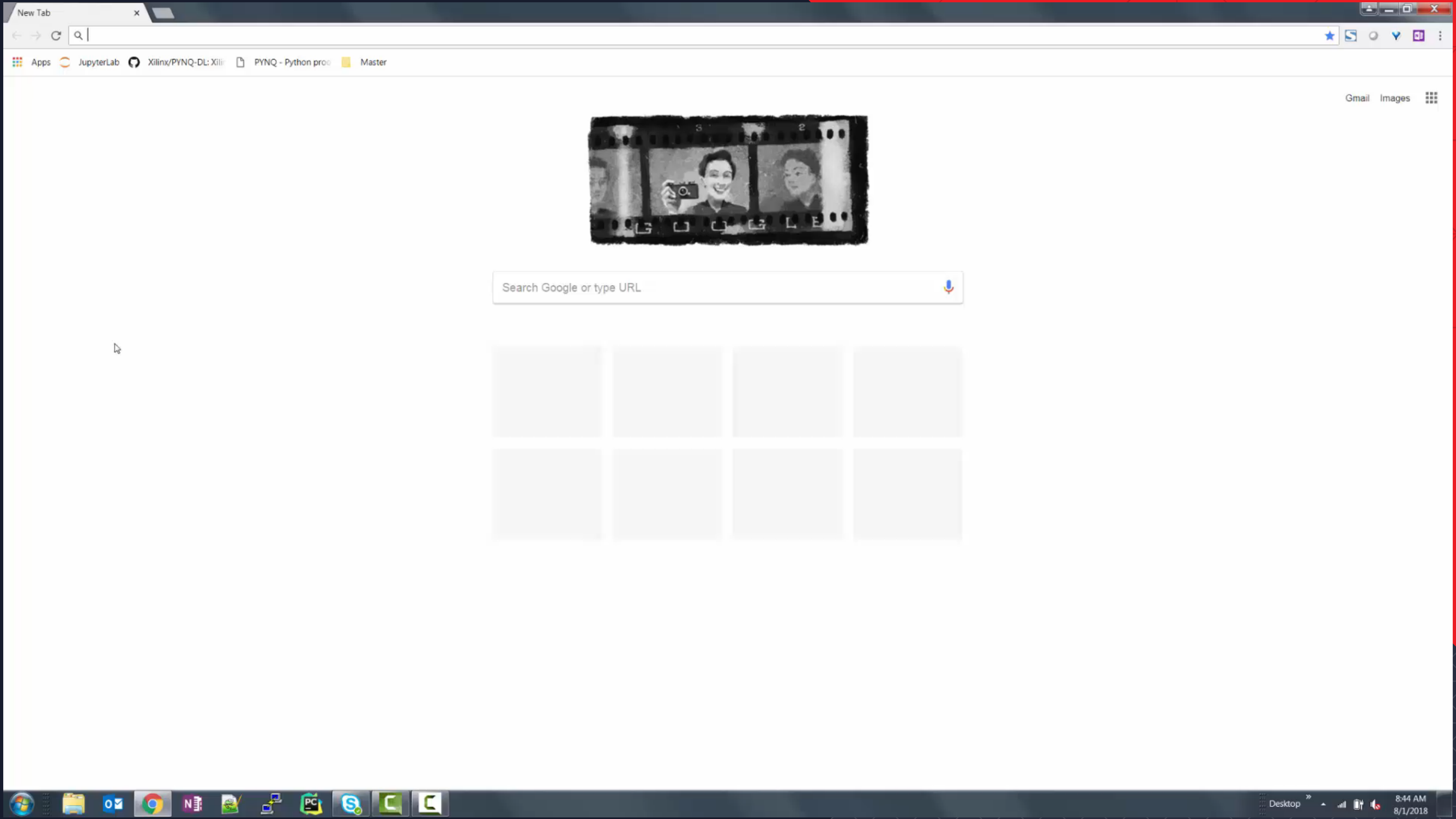
JupyterLab server	
IPython kernel	
Ubuntu-based Linux	Circuit overlay
Arm microprocessor	Programmable logic

PYNQ Demo Set-up

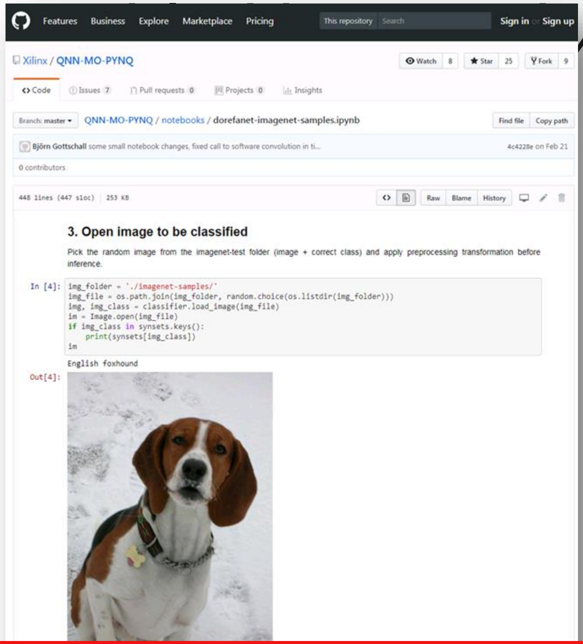


PYNQ-Z2 Development Board

Zynq Programmable Platform


New Tab
Apps
JupyterLab
Xilinx/PYNQ-DL: Xili
PYNQ - Python pro
Master
Gmail
Images
Search Google or type URL
Desktop
8:44 AM
8/1/2018

Installing Software and Hardware with PIP!



Xilinx / QNN-MO-PYNQ

QNN-MO-PYNQ / notebooks / dorefanet-imagenet-samples.ipynb

Björn Gottschall some small notebook changes, fixed call to software convolution in ti... 4c4228e on Feb 21

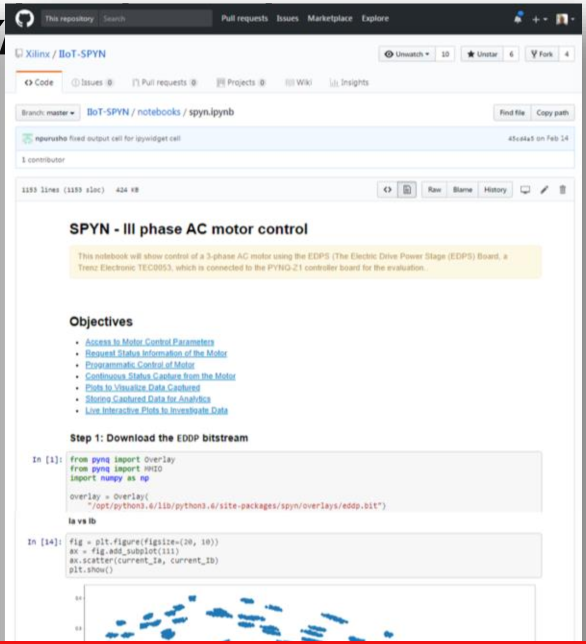
0 contributors

448 lines (447 sloc) 253 KB

3. Open image to be classified

Pick the random image from the imagenet-test folder (image + correct class) and apply preprocessing transformation before inference.

```
In [4]: img_folder = './imagenet-samples/'
        img_file = os.path.join(img_folder, random.choice(os.listdir(img_folder)))
        img, img_class = classifier.load_image(img_file)
        im = Image.open(img_file)
        if img_class in synsets.keys():
            print(synsets[img_class])
        im
        English foxhound
Out[4]:
```



Xilinx / IIoT-SPYN

IIoT-SPYN / notebooks / spyn.ipynb

npurusho fixed output cell for ipywidget cell 45cd4a5 on Feb 14

1 contributor

1153 lines (1153 sloc) 424 KB

SPYN - III phase AC motor control

This notebook will show control of a 3-phase AC motor using the EDPS (The Electric Drive Power Stage (EDPS) Board, a Trenz Electronic TEC0053, which is connected to the PYNQ-Z1 controller board for the evaluation.

Objectives

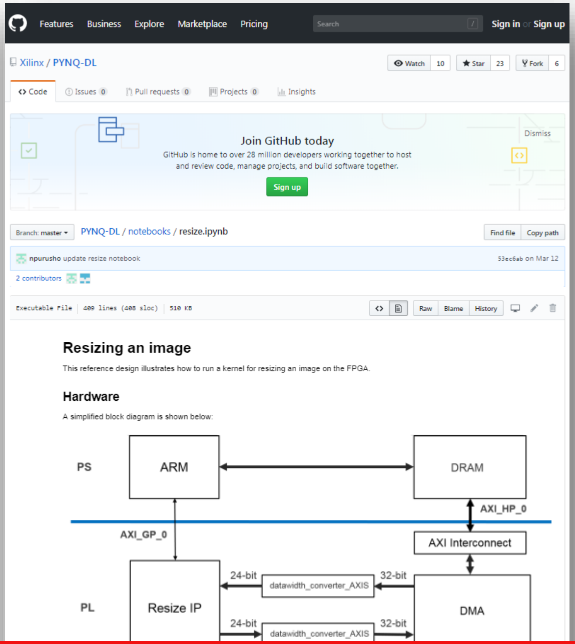
- Access to Motor Control Parameters
- Request Status Information of the Motor
- Programmatic Control of Motor
- Continuous Status Capture from the Motor
- Plots to Visualize Data Captured
- Storing Captured Data for Analytics
- Live Interactive Plots to Investigate Data

Step 1: Download the EDDP bitstream

```
In [1]: from pynq import Overlay
        from pynq import MMIO
        import numpy as np

        overlay = Overlay(
            "/opt/python3.6/lib/python3.6/site-packages/spyn/overlays/eddp.bit")
```

Ia vs Ib

```
In [14]: fig = plt.figure(figsize=(20, 10))
         ax = fig.add_subplot(111)
         ax.scatter(current_Ia, current_Ib)
         plt.show()
```



Xilinx / PYNQ-DL

Join GitHub today

GitHub is home to over 28 million developers working together to host and review code, manage projects, and build software together.

PYNQ-DL / notebooks / resize.ipynb

npurusho update resize notebook 53ec6ab on Mar 12

2 contributors

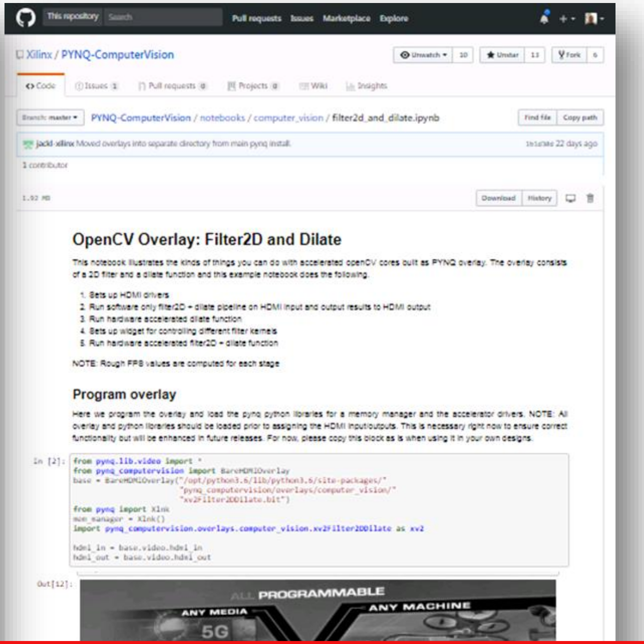
Executable File 409 lines (408 sloc) 510 KB

Resizing an image

This reference design illustrates how to run a kernel for resizing an image on the FPGA.

Hardware

A simplified block diagram is shown below:



Xilinx / PYNQ-ComputerVision

PYNQ-ComputerVision / notebooks / computer_vision / filter2d_and_dilate.ipynb

jackl-xilinx Moved overlays into separate directory from main pynq install. 1b1a9e 22 days ago

1 contributor

1.92 MB

OpenCV Overlay: Filter2D and Dilate

This notebook illustrates the kinds of things you can do with accelerated openCV cores built as PYNQ overlay. The overlay consists of a 2D filter and a dilate function and this example notebook does the following.

1. Sets up HDMI drivers
2. Run software only filter2D + dilate pipeline on HDMI input and output results to HDMI output
3. Run hardware accelerated dilate function
4. Sets up widget for controlling different filter kernels
5. Run hardware accelerated filter2D + dilate function

NOTE: Rough FPS values are computed for each stage

Program overlay

Here we program the overlay and load the pynq python libraries for a memory manager and the accelerator drivers. NOTE: All overlay and python libraries should be loaded prior to assigning the HDMI input/outputs. This is necessary right now to ensure correct functionality but will be enhanced in future releases. For now, please copy this block as is when using it in your own designs.

```
In [2]: from pynq.lib.video import *
        from pynq_computervision import BareHDMIOverlay
        base = BareHDMIOverlay("/opt/python3.6/lib/python3.6/site-packages/"
                               "pynq_computervision/overlays/computer_vision/"
                               "xv2Filter2DDilate.bit")
        from pynq import Xlnk
        mem_manager = Xlnk()
        import pynq_computervision.overlays.computer_vision.xv2Filter2DDilate as xv2

        hdmi_in = base.video.hdmi_in
        hdmi_out = base.video.hdmi_out
Out[12]:
```

Download a design from GitHub with a single Python command:

```
pip install git+https://github.com/Xilinx/pynq-helloworld.git
```

Load the `resizer` Hybrid Package

```
from pynq import Overlay
resizer = Overlay('./resizer.bit')
```

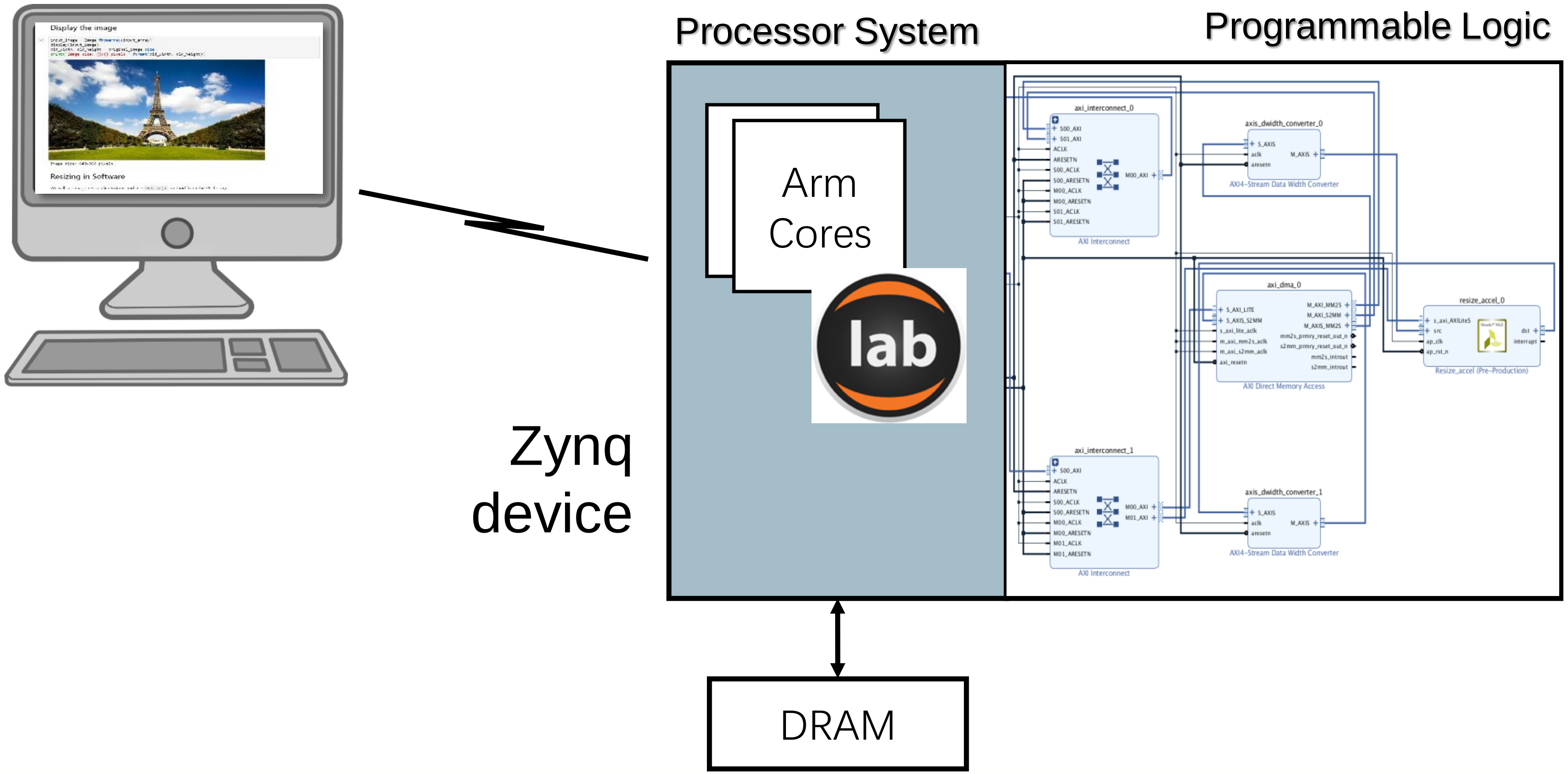
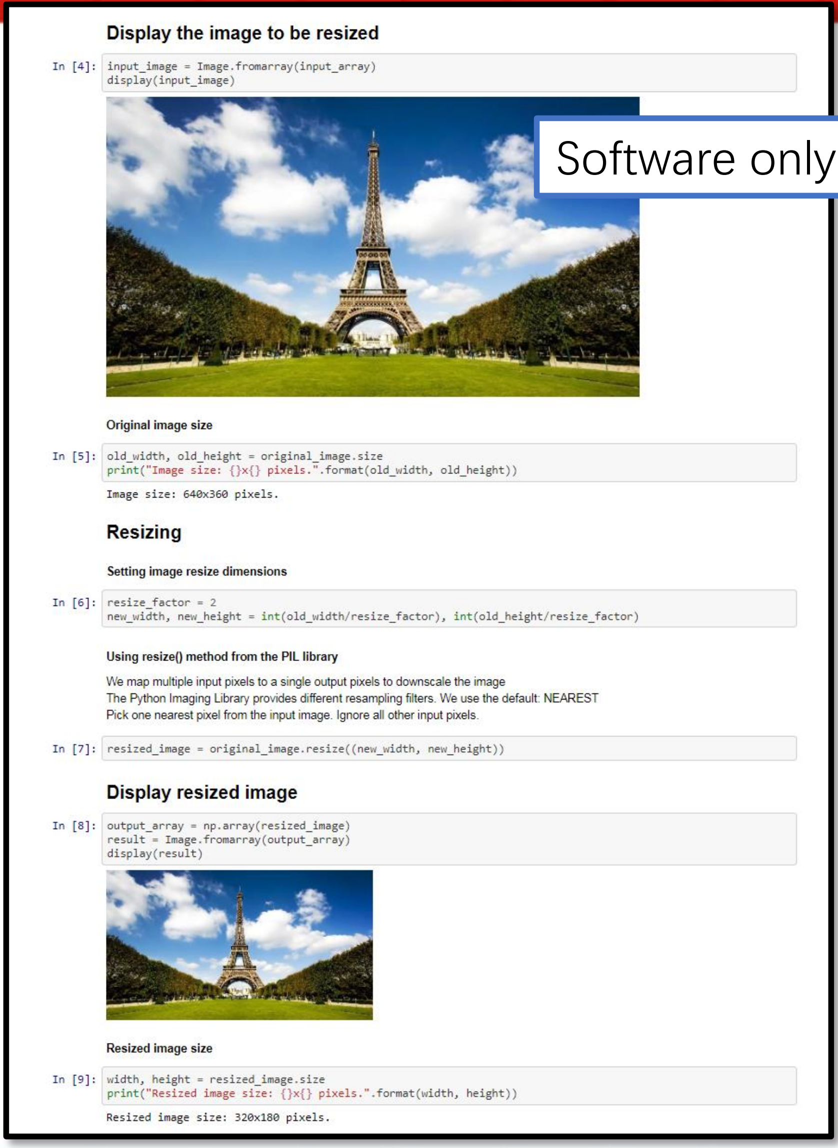


Image Resizer



Software only resizing

Display the image to be resized

```
In [4]: input_image = Image.fromarray(input_array)
        display(input_image)
```

Original image size

```
In [5]: old_width, old_height = original_image.size
        print("Image size: {}x{} pixels.".format(old_width, old_height))

        Image size: 640x360 pixels.
```

Resizing

Setting image resize dimensions

```
In [6]: resize_factor = 2
        new_width, new_height = int(old_width/resize_factor), int(old_height/resize_factor)
```

Using resize() method from the PIL library

We map multiple input pixels to a single output pixels to downscale the image
The Python Imaging Library provides different resampling filters. We use the default: NEAREST
Pick one nearest pixel from the input image. Ignore all other input pixels.

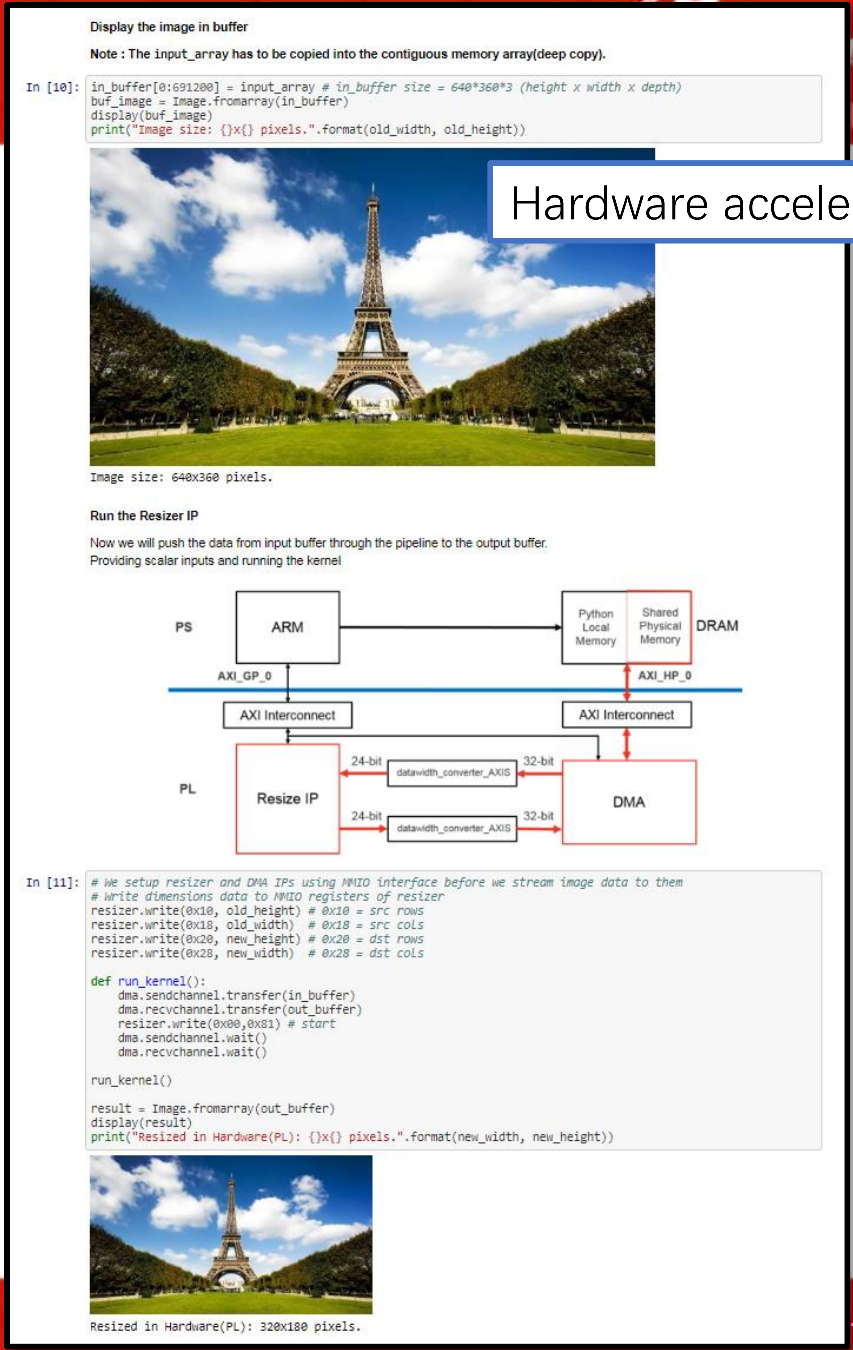
```
In [7]: resized_image = original_image.resize((new_width, new_height))
```

Display resized image

```
In [8]: output_array = np.array(resized_image)
        result = Image.fromarray(output_array)
        display(result)
```

Resized image size

```
In [9]: width, height = resized_image.size
        print("Resized image size: {}x{} pixels.".format(width, height))

        Resized image size: 320x180 pixels.
```



Hardware accelerated resizing

Display the image in buffer

Note : The input_array has to be copied into the contiguous memory array(deep copy).

```
In [10]: in_buffer[0:691200] = input_array # in_buffer size = 640*360*3 (height x width x depth)
         buf_image = Image.fromarray(in_buffer)
         display(buf_image)
         print("Image size: {}x{} pixels.".format(old_width, old_height))

         Image size: 640x360 pixels.
```

Run the Resizer IP

Now we will push the data from input buffer through the pipeline to the output buffer.
Providing scalar inputs and running the kernel

```
In [11]: # We setup resizer and DMA IPs using MMIO interface before we stream image data to them
         # Write dimensions data to MMIO registers of resizer
         resizer.write(0x10, old_height) # 0x10 = src rows
         resizer.write(0x18, old_width)  # 0x18 = src cols
         resizer.write(0x20, new_height) # 0x20 = dst rows
         resizer.write(0x28, new_width)  # 0x28 = dst cols

         def run_kernel():
             dma.sendchannel.transfer(in_buffer)
             dma.recvchannel.transfer(out_buffer)
             resizer.write(0x00,0x81) # start
             dma.sendchannel.wait()
             dma.recvchannel.wait()

         run_kernel()

         result = Image.fromarray(out_buffer)
         display(result)
         print("Resized in Hardware(PL): {}x{} pixels.".format(new_width, new_height))

         Resized in Hardware(PL): 320x180 pixels.
```

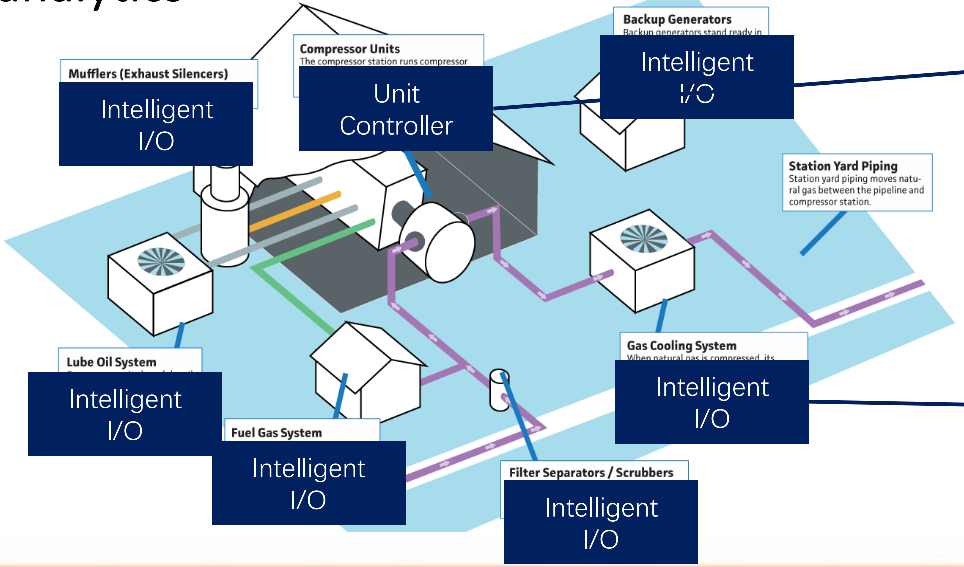
Case Study - Industrial Controls IIoT

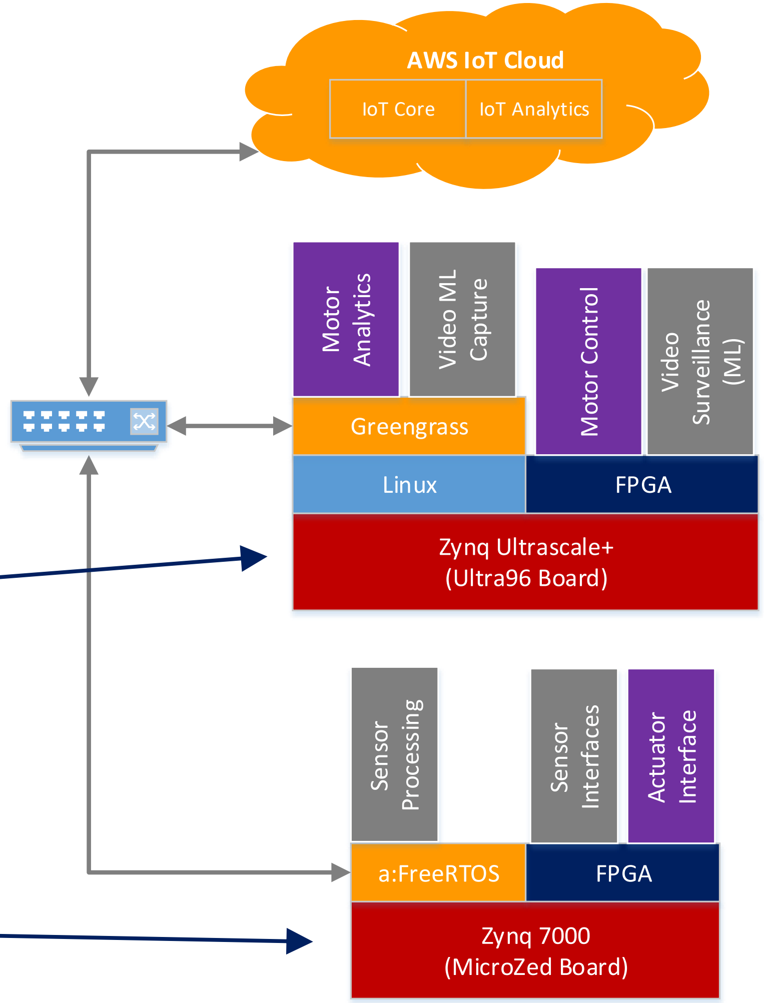
Zynq 7000 w/ AWS FreeRTOS

- Targets limited resource devices
- Often bridge physical & digital domains

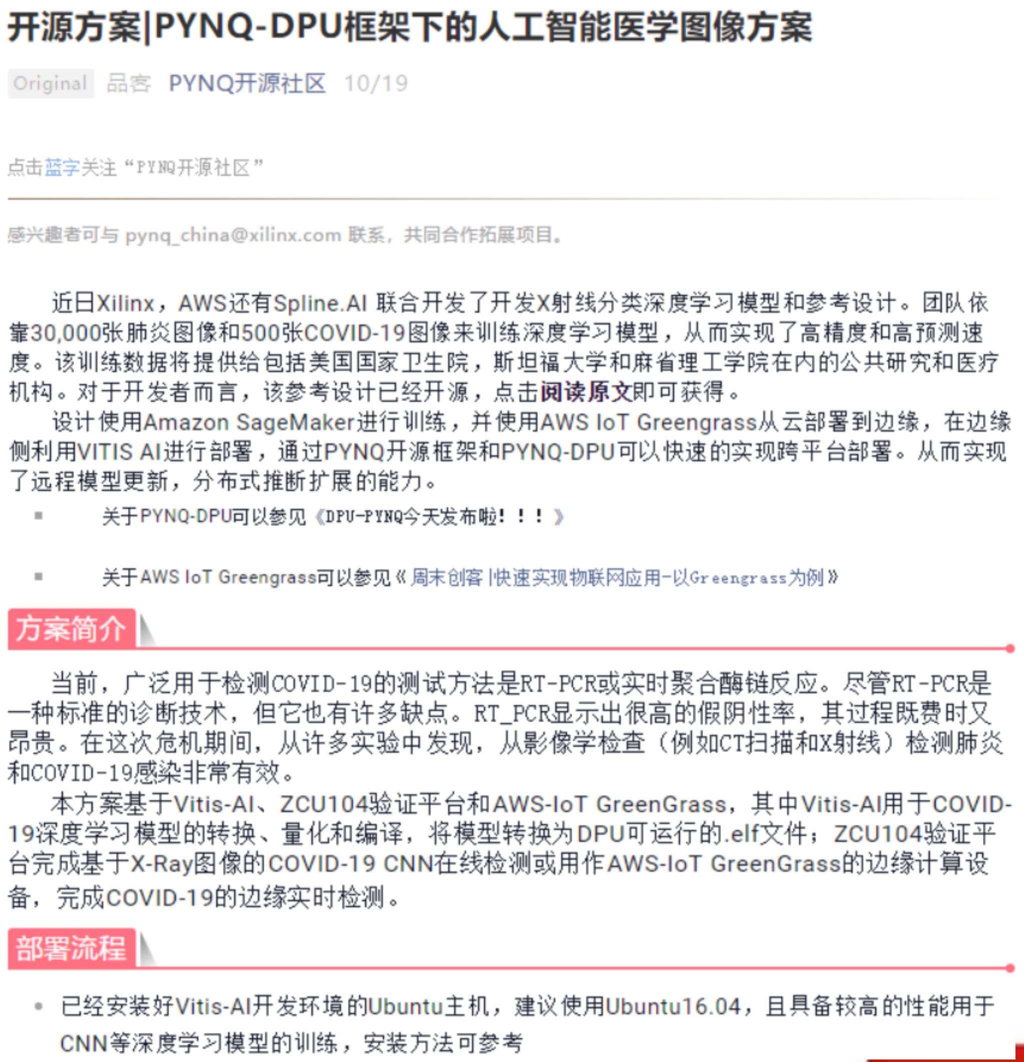
Zynq US+ w/ Linux & Greengrass

- Targets more capable embedded devices
- Brings together control action & local analytics


Mufflers (Exhaust Silencers)
Intelligent I/O
Compressor Units
The compressor station runs compressor
Unit Controller
Backup Generators
Backup generators stand ready in
Intelligent I/O
Station Yard Piping
Station yard piping moves natural gas between the pipeline and compressor station.
Lube Oil System
Intelligent I/O
Fuel Gas System
Intelligent I/O
Gas Cooling System
When natural gas is compressed, its
Intelligent I/O
Filter Separators / Scrubbers
Intelligent I/O



X 射线分型深度学习模型和参考设计



开源方案|PYNQ-DPU框架下的人工智能医学图像方案

Original 品客 PYNQ开源社区 10/19

点击蓝字关注“PYNQ开源社区”

感兴趣者可与 pynq_china@xilinx.com 联系，共同合作拓展项目。

近日Xilinx，AWS还有Spline.AI 联合开发了开发X射线分类深度学习模型和参考设计。团队依靠30,000张肺炎图像和500张COVID-19图像来训练深度学习模型，从而实现了高精度和高预测速度。该训练数据将提供给包括美国国家卫生院，斯坦福大学和麻省理工学院在内的公共研究和医疗机构。对于开发者而言，该参考设计已经开源，点击**阅读原文**即可获得。

设计使用Amazon SageMaker进行训练，并使用AWS IoT Greengrass从云部署到边缘，在边缘侧利用VITIS AI进行部署，通过PYNQ开源框架和PYNQ-DPU可以快速的实现跨平台部署。从而实现了远程模型更新，分布式推断扩展的能力。

- 关于PYNQ-DPU可以参见《DPU-PYNQ今天发布啦！！！》
- 关于AWS IoT Greengrass可以参见《周末创客 |快速实现物联网应用-以Greengrass为例》

方案简介

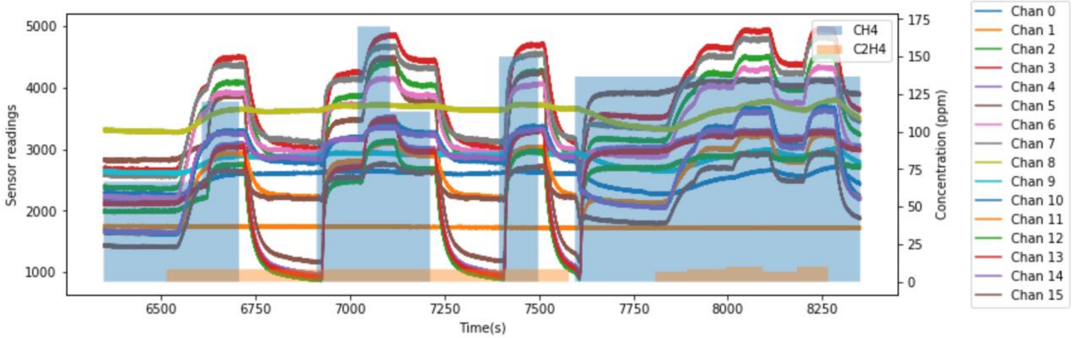
当前，广泛用于检测COVID-19的测试方法是RT-PCR或实时聚合酶链反应。尽管RT-PCR是一种标准的诊断技术，但它也有许多缺点。RT_PCR显示出很高的假阴性率，其过程既费时又昂贵。在这次危机期间，从许多实验中发现，从影像学检查（例如CT扫描和X射线）检测肺炎和COVID-19感染非常有效。

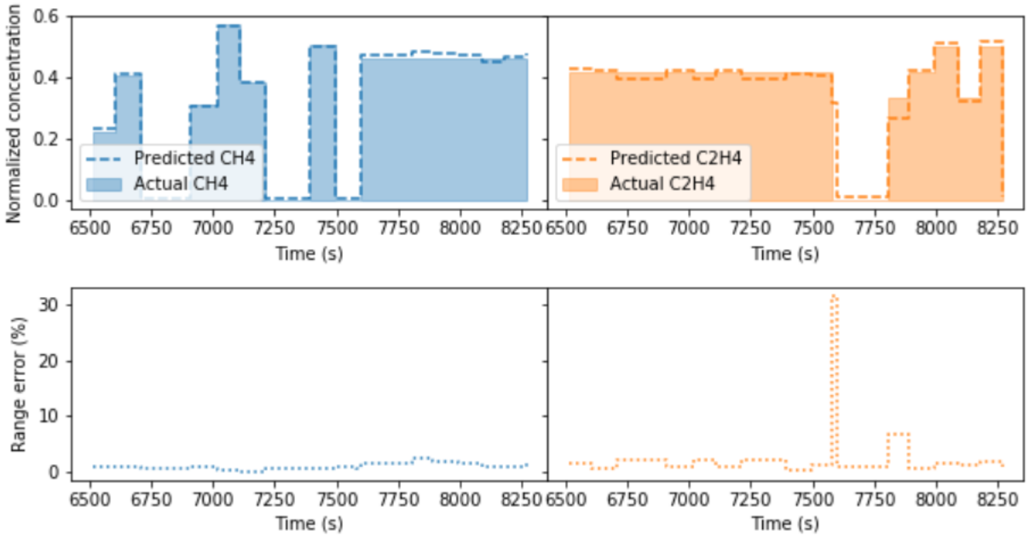
本方案基于Vitis-AI、ZCU104验证平台和AWS-IoT GreenGrass，其中Vitis-AI用于COVID-19深度学习模型的转换、量化和编译，将模型转换为DPU可运行的.elf文件；ZCU104验证平台完成基于X-Ray图像的COVID-19 CNN在线检测或用作AWS-IoT GreenGrass的边缘计算设备，完成COVID-19的边缘实时检测。

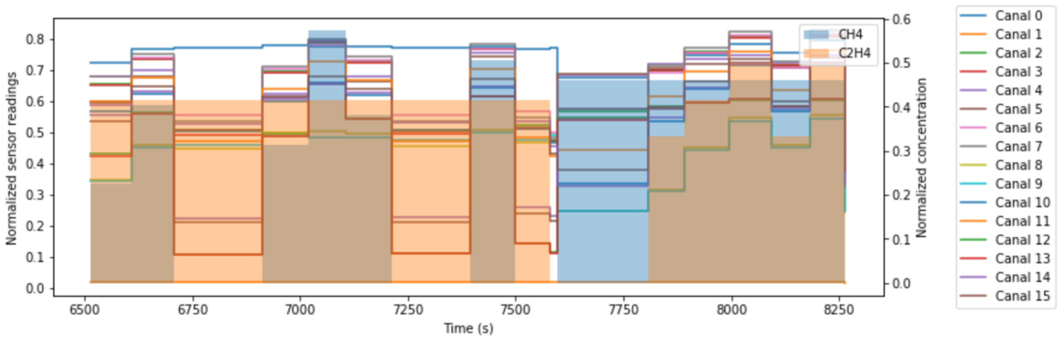
部署流程

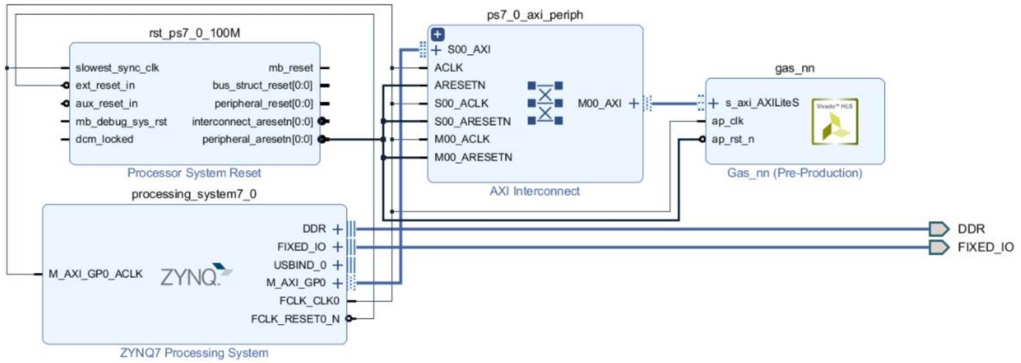
- 已经安装好Vitis-AI开发环境的Ubuntu主机，建议使用Ubuntu16.04，且具备较高的性能用于CNN等深度学习模型的训练，安装方法可参考

智能大气环境实时预测系统



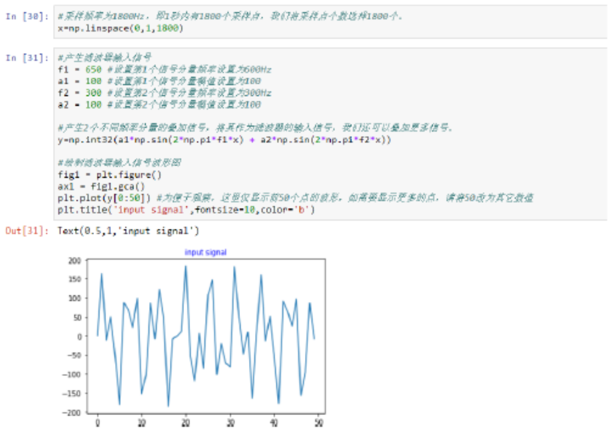




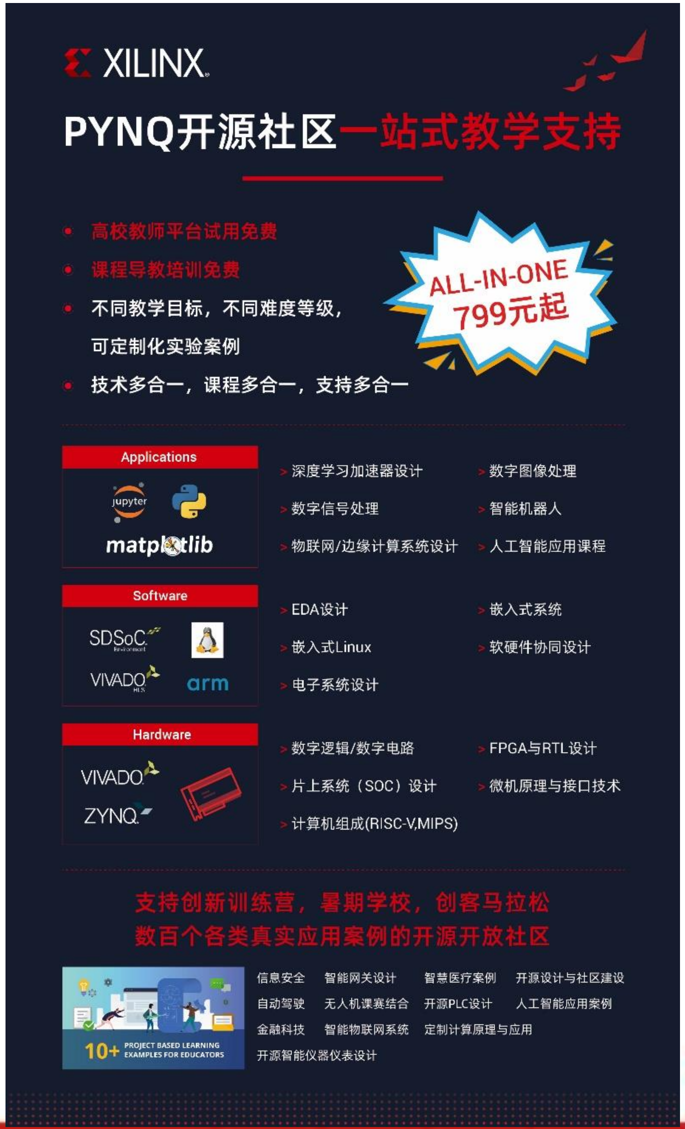


课程学习

打开 fir_pynq 目录，运行预先设计好的 fir11.ipynb，按照 notebook 中的步骤完成接下来的实验步骤。



```
In [30]: #采样频率为1800Hz，即1秒内有1800个采样点，我们将采样点个数选择1800个。
         x=np.linspace(0,1,1800)

In [31]: #产生滤波器输入信号
         f1 = 650 #设置第1个信号分量频率设置为600Hz
         a1 = 100 #设置第1个信号分量幅值设置为100
         f2 = 300 #设置第2个信号分量频率设置为300Hz
         a2 = 100 #设置第2个信号分量幅值设置为100

         #产生2个不同频率分量的叠加信号，将其作为滤波器的输入信号，我们还可以叠加更多信号。
         y=np.int32(a1*np.sin(2*np.pi*f1*x) + a2*np.sin(2*np.pi*f2*x))

         #绘制滤波器输入信号波形图
         fig1 = plt.figure()
         ax1 = fig1.gca()
         plt.plot(y[0:50]) #为便于观察，这里仅显示前50个点的波形，如需要显示更多的点，请将50改为其它数值
         plt.title('input signal',fontsize=10,color='b')

Out[31]: Text(0.5,1,'input signal')
```



XILINX
ALL-IN-ONE
799元起
Applications
jupyter
matplotlib
Software
SDSoC
VIVADO
arm
Hardware
VIVADO
ZYNQ
10+ PROJECT BASED LEARNING EXAMPLES FOR EDUCATORS

PYNQ开源社区一站式教学支持

- 高校教师平台试用免费
- 课程导教培训免费
- 不同教学目标，不同难度等级，可定制化实验案例
- 技术多合一，课程多合一，支持多合一

Applications

- 深度学习加速器设计
- 数字图像处理
- 数字信号处理
- 智能机器人
- 物联网/边缘计算系统设计
- 人工智能应用课程

Software

- EDA设计
- 嵌入式系统
- 嵌入式Linux
- 软硬件协同设计
- 电子系统设计

Hardware

- 数字逻辑/数字电路
- FPGA与RTL设计
- 片上系统（SOC）设计
- 微机原理与接口技术
- 计算机组成(RISC-V,MIPS)

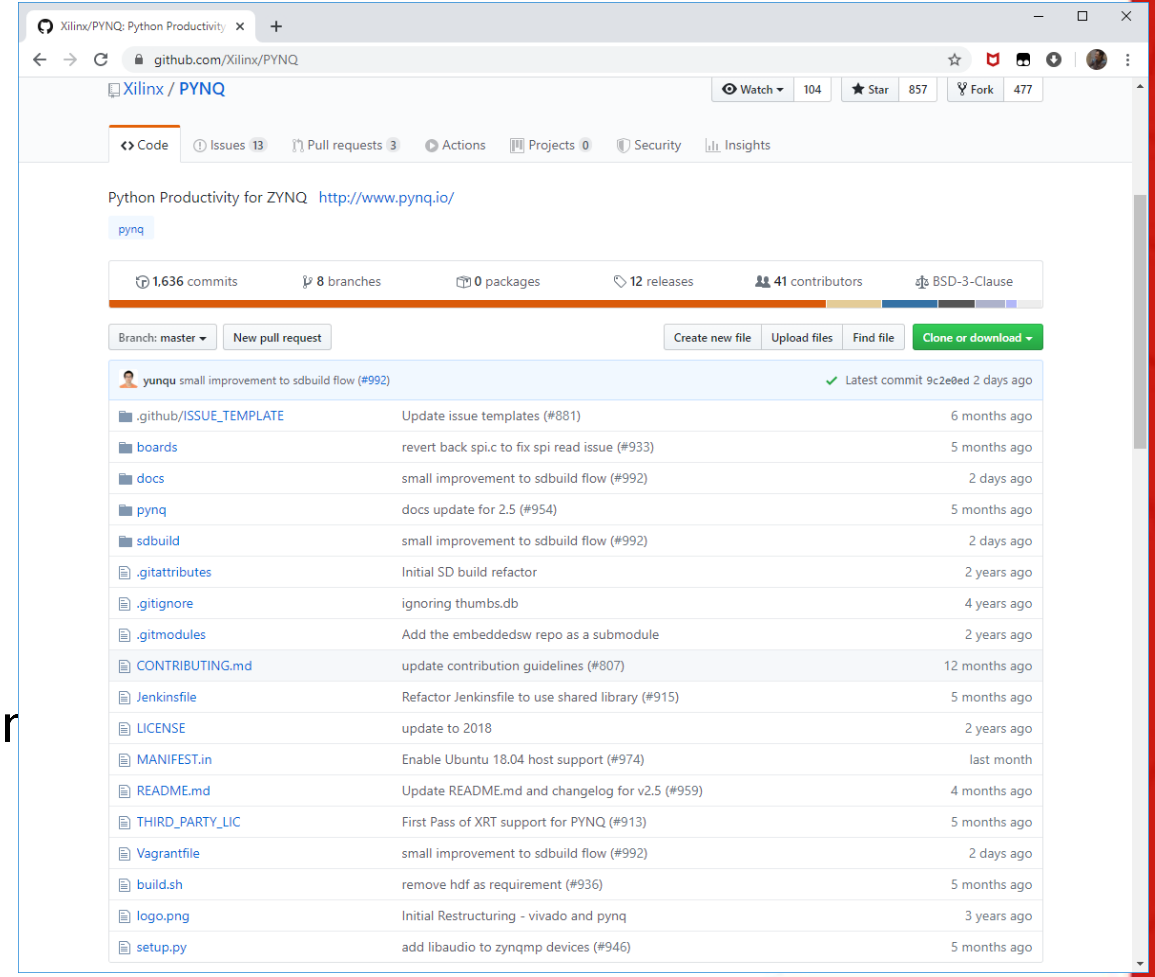
支持创新训练营，暑期学校，创客马拉松
数百个各类真实应用案例的开源开放社区

信息安全　智能网关设计　智慧医疗案例　开源设计与社区建设
自动驾驶　无人机课赛结合　开源PLC设计　人工智能应用案例
金融科技　智能物联网系统　定制计算原理与应用
开源智能仪器仪表设计

PYNQ 访问硬件的API

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

- pynq - PYNQ packages
  - Python API modules
- boards - 板卡相关的文件
  - Vivado 工程, bitstreams
  - Petlinux BSPs
  - Overlays
  - 示例notebooks
- sdbuild – PYNQ移植相关文件
  - Pynq related meta files
  - Makefile, to generate
    - SD card image、boot files、 sysroot、 Petalir
  - Scripts
  - Additional RootFS packages
    - pandas、Jupyter、python packages, etc



Python Productivity for ZYNQ http://www.pynq.io/

1,636 commits | 8 branches | 0 packages | 12 releases | 41 contributors | BSD-3-Clause

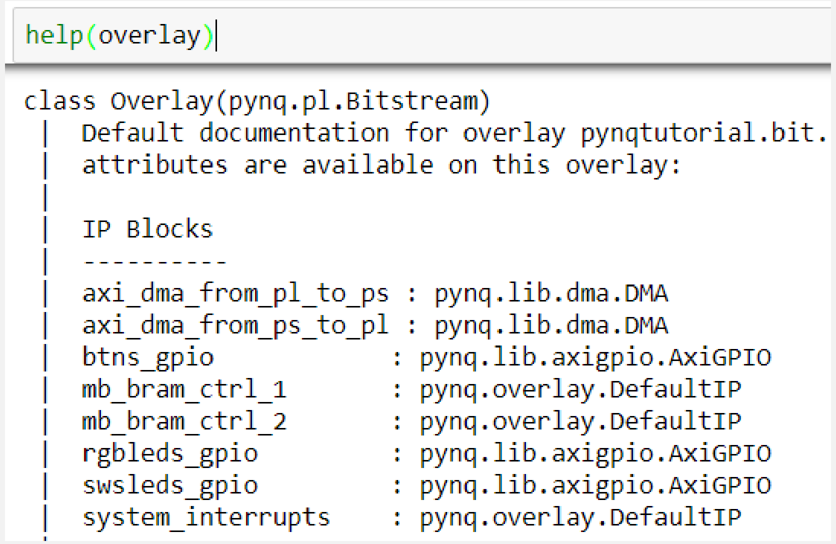
yunqu small improvement to sdbuild flow (#992) ✓ Latest commit 9c2e0ed 2 days ago

.github/ISSUE_TEMPLATE	Update issue templates (#881)	6 months ago
boards	revert back spi.c to fix spi read issue (#933)	5 months ago
docs	small improvement to sdbuild flow (#992)	2 days ago
pynq	docs update for 2.5 (#954)	5 months ago
sdbuild	small improvement to sdbuild flow (#992)	2 days ago
.gitattributes	Initial SD build refactor	2 years ago
.gitignore	ignoring thumbs.db	4 years ago
.gitmodules	Add the embeddedsw repo as a submodule	2 years ago
CONTRIBUTING.md	update contribution guidelines (#807)	12 months ago
Jenkinsfile	Refactor Jenkinsfile to use shared library (#915)	5 months ago
LICENSE	update to 2018	2 years ago
MANIFEST.in	Enable Ubuntu 18.04 host support (#974)	last month
README.md	Update README.md and changelog for v2.5 (#959)	4 months ago
THIRD_PARTY_LIC	First Pass of XRT support for PYNQ (#913)	5 months ago
Vagrantfile	small improvement to sdbuild flow (#992)	2 days ago
build.sh	remove hdf as requirement (#936)	5 months ago
logo.png	Initial Restructuring - vivado and pynq	3 years ago
setup.py	add libaudio to zynqmp devices (#946)	5 months ago

Python overlay API

- PYNQ Overlay class supports basic overlay functionality
  - Requires Tcl/HWH
  - Allows download, and overlay discove
  - Assigns default drivers to IP
    - Read() and write() access to IP address sp

```
from pynq import Overlay
overlay = Overlay("pynqtutorial.bit")
```

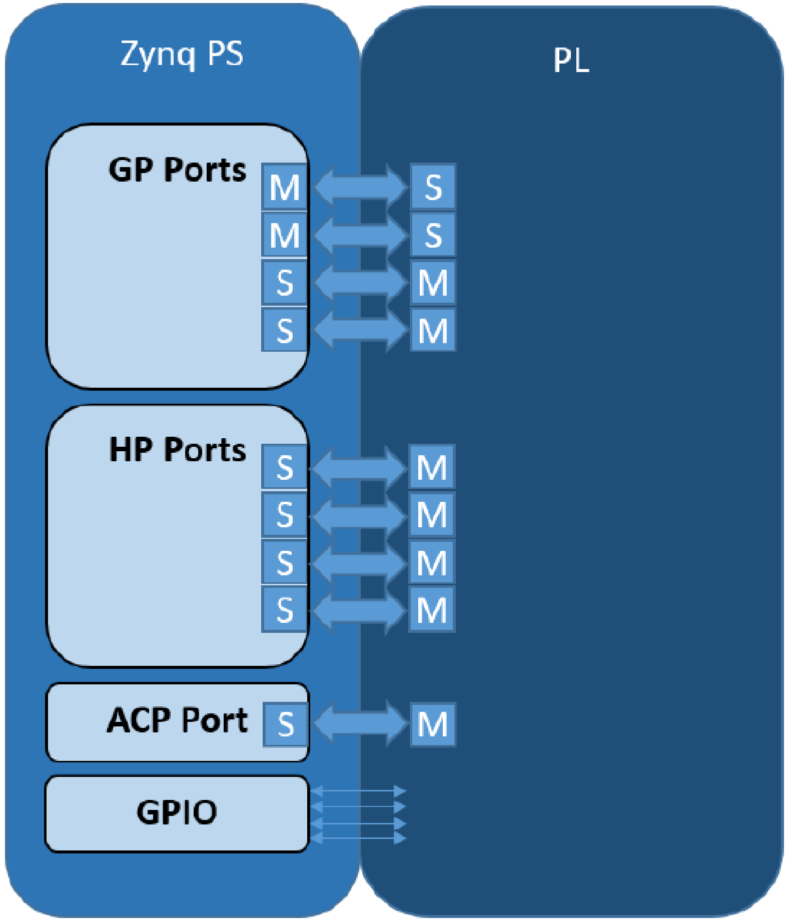


```
help(overlay)
```

```
class Overlay(pynq.pl.Bitstream)
 |  Default documentation for overlay pynqtutorial.bit.
 |  attributes are available on this overlay:
 |
 |  IP Blocks
 |  ----------
 |  axi_dma_from_pl_to_ps : pynq.lib.dma.DMA
 |  axi_dma_from_ps_to_pl : pynq.lib.dma.DMA
 |  btns_gpio            : pynq.lib.axigpio.AxiGPIO
 |  mb_bram_ctrl_1       : pynq.overlay.DefaultIP
 |  mb_bram_ctrl_2       : pynq.overlay.DefaultIP
 |  rgbleds_gpio         : pynq.lib.axigpio.AxiGPIO
 |  swsleds_gpio         : pynq.lib.axigpio.AxiGPIO
 |  system_interrupts    : pynq.overlay.DefaultIP
```

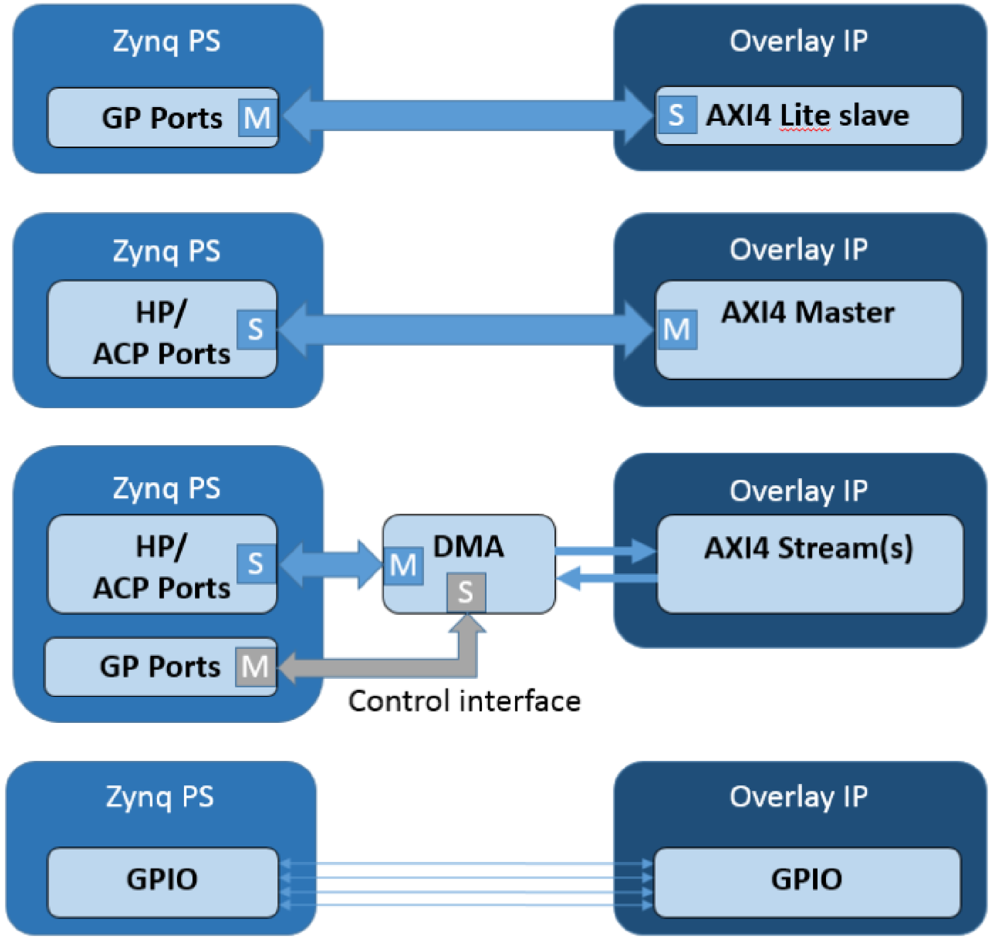
```
overlay.rgbleds_gpio.write(0, 3)
overlay.swsleds_gpio.read()
```

ZYNQ PL与PS接口回顾



- AXI General Purpose ports
  - 2x Master (32-bit)
  - 2x Slave (32-bit)
- AXI High Performance
  - 4x Slave (64-bit)
    - 1K FIFOs
- ACP
  - 1x Slave (64-bit)
  - Cache access
- GPIO (EMIO)
  - Wires (64)

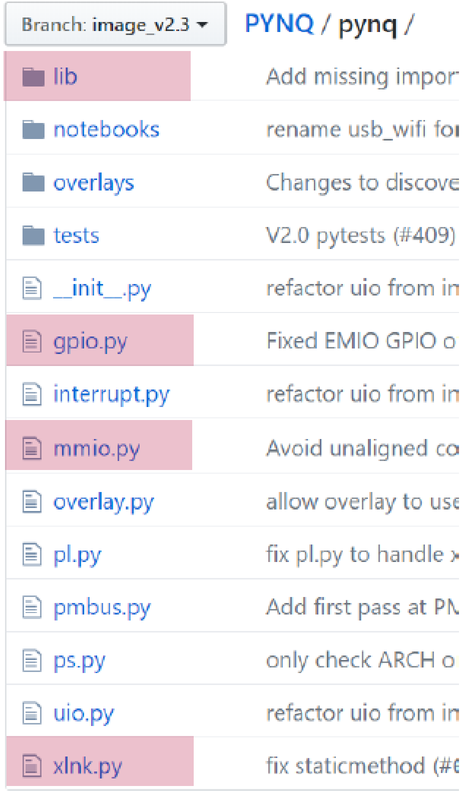
常用IP连接方式介绍



- > **AXI (Lite) Slave IP**
  - >> General Purpose ports
  - >> Typically lower performance IP
- > **AXI Master**
  - >> AXI HP/ACP
  - >> Typically higher performance IP
- > **AXI Stream**
  - >> Via DMA
  - >> HP/ACP ports for data path
  - >> GP slave for control
- > **GPIO**
  - >> Control type data

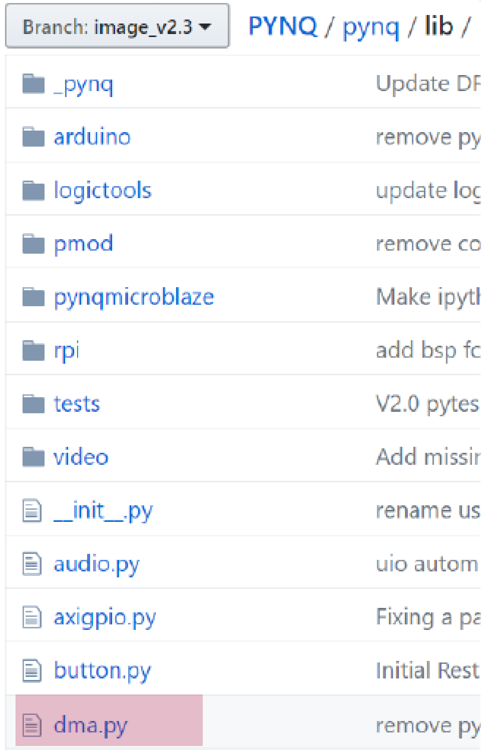
PYNQ - PL与PS接口类

- **MMIO (pynq.mmio)**
  - Memory Mapped Input Output
  - Register based memory mapped transactions
- **Xlnk (pynq.xlnk)**
  - Memory allocation (used in DMA or for AXI Master peripheral)
- **DMA (pynq.lib.dma)**
  - Direct Memory Access
  - Offload memory transfers from main CPU
- **GPIO (pynq.gpio)**
  - Read/Write GPIO wires



Branch: image_v2.3 ▾ PYNQ / pynq /

lib	Add missing impor
notebooks	rename usb_wifi for
overlays	Changes to discove
tests	V2.0 pytests (#409)
__init__.py	refactor uio from in
gpio.py	Fixed EMIO GPIO o
interrupt.py	refactor uio from in
mmio.py	Avoid unaligned co
overlay.py	allow overlay to use
pl.py	fix pl.py to handle x
pmbus.py	Add first pass at PN
ps.py	only check ARCH o
uio.py	refactor uio from in
xlnk.py	fix staticmethod (#6

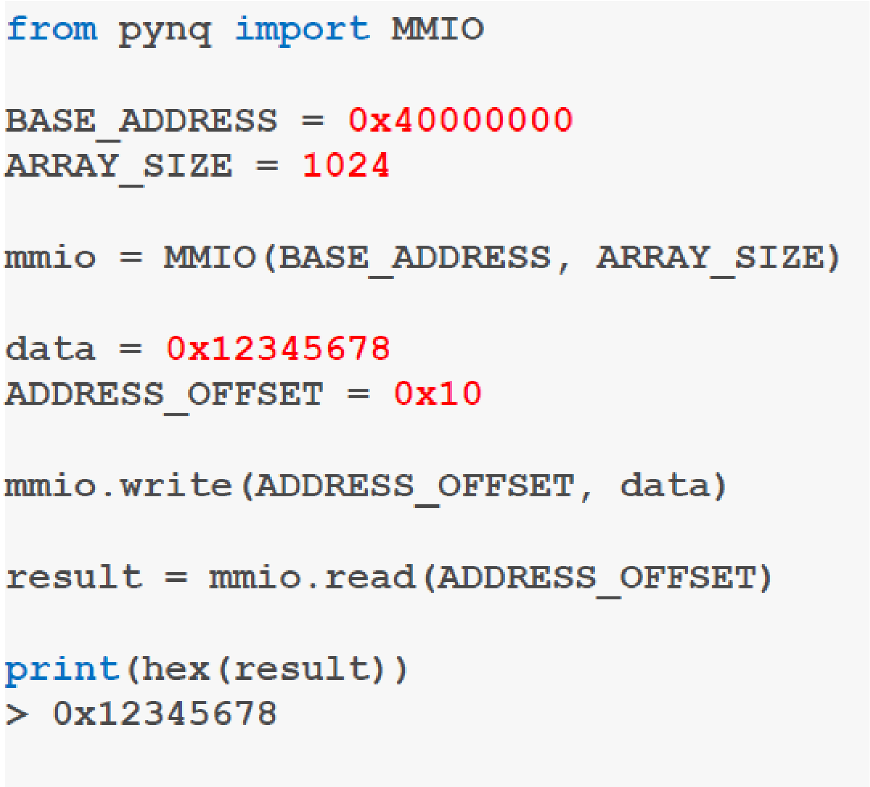


Branch: image_v2.3 ▾ PYNQ / pynq / lib /

_pynq	Update DF
arduino	remove py
logictools	update log
pmod	remove co
pynqmicroblaze	Make ipytl
rpi	add bsp fc
tests	V2.0 pytes
video	Add missir
__init__.py	rename us
audio.py	uio autom
axigpio.py	Fixing a pa
button.py	Initial Rest
dma.py	remove py

PYNQ接口 - MMIO类

- Import MMIO
- Define memory mapped region
  - BASE_ADDRESS: starting location
  - ARRAY_SIZE: length of accessible memory (optional, default 4 bytes)
- Read and Write 32-bit values
  - ADDRESS_OFFSET: Offset from BASE_ADDRESS, should be multiples of 4
  - Need to ensure the memory can be read/written



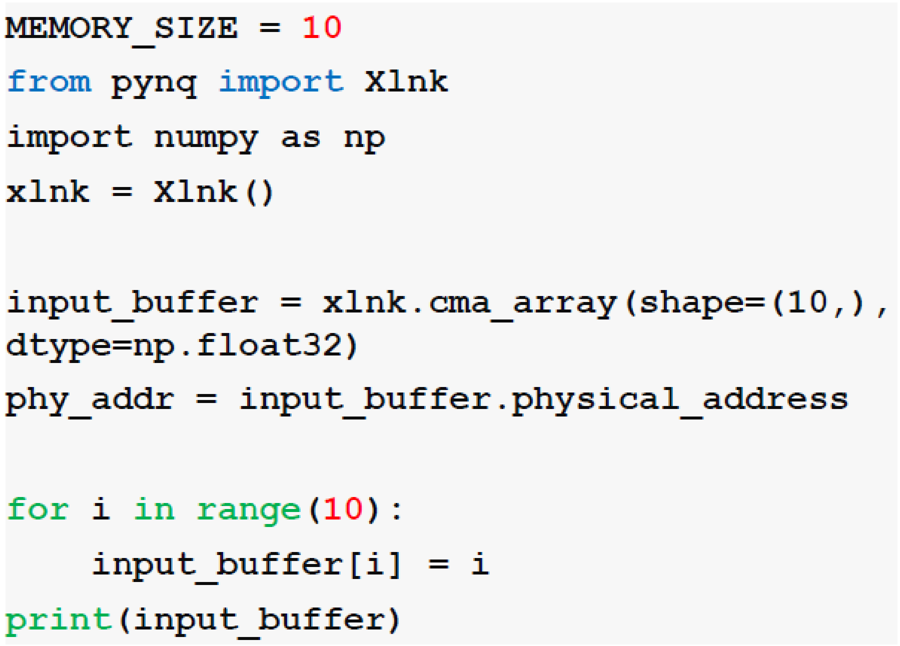
```
from pynq import MMIO

BASE_ADDRESS = 0x40000000
ARRAY_SIZE = 1024

mmio = MMIO(BASE_ADDRESS, ARRAY_SIZE)

data = 0x12345678
ADDRESS_OFFSET = 0x10

mmio.write(ADDRESS_OFFSET, data)

result = mmio.read(ADDRESS_OFFSET)

print(hex(result))
> 0x12345678
```

PYNQ接口 - Xlnk类

- Memory managed by Linux
  - Virtual
- Memory must be allocated before PL Master can access it
  - PL needs the physical address of a memory buffer
- Xlnk can allocate (contiguous) memory buffers (using NumPy)
  - Maps virtual and physical addresses
  - Contiguous memory is more efficient/allows simpler DMA logic
  - Array can be specified as NumPy data type, and size/shape
- Once buffer is allocated a DMA can be used to transfer data between PS/PL
- DMA class uses Xlnk for memory allocation

PYNQ接口 - Xlnk类案例

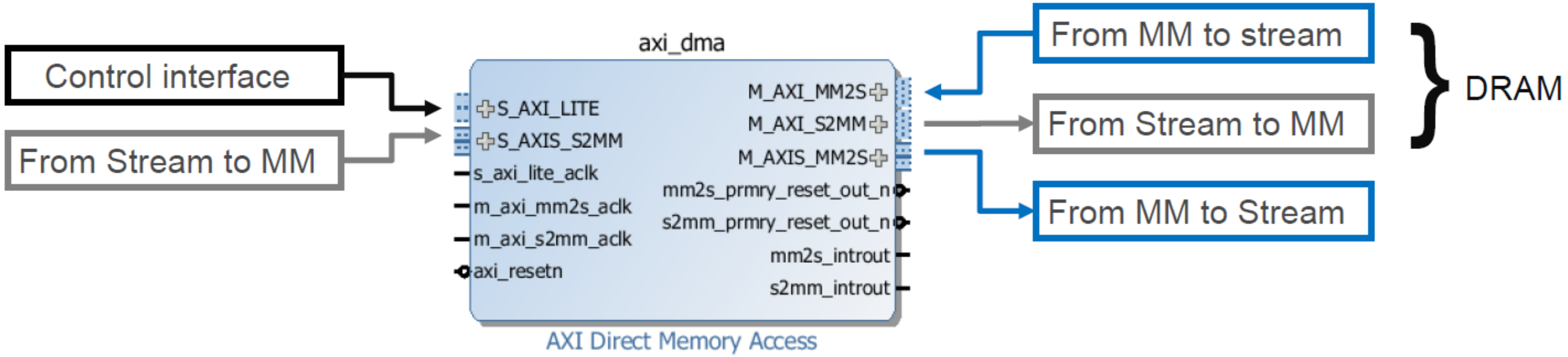
- Import Xlnk
- Allocate contiguous buffer
  - cma_array()
  - Returns Linux virtual address
- Get Physical Address
  - `.physical_address`
  - Can be used by PL to access DDR (Linux) memory
- Read/write buffer



```
MEMORY_SIZE = 10
from pynq import Xlnk
import numpy as np
xlnk = Xlnk()

input_buffer = xlnk.cma_array(shape=(10,),
dtype=np.float32)
phy_addr = input_buffer.physical_address

for i in range(10):
    input_buffer[i] = i
print(input_buffer)
```

PYNQ接口 - AXI DMA IP

- AXI Lite control interface (AXI GP port)
- Memory mapped interface (AXI interface, HP/ACP ports)
- AXI Stream interface (AXI stream accelerator)
- Transfer between streams and memory mapped locations
  - Paths from PL to DRAM and DRAM to PL
  - Memory Mapped to Stream (MM2S)/Stream to Memory Mapped (S2MM)



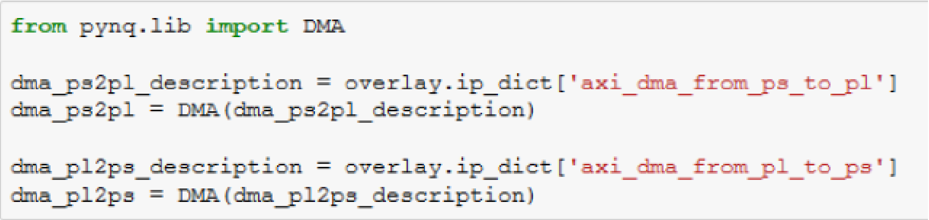
PYNQ接口 - AXI DMA IP

- **Direct memory access**
  - Transfer data between memories directly
    - PS - PL
  - Bypasses CPU
    - Doesn’t waste CPU cycles on data transfer
  - Speed up memory transfers with burst transactions
- **Xilinx *AXI Direct Memory Access* IP block supported in PYNQ**
  - Read and Write Paths from PL to DDR and DDR to PL
  - Memory Mapped to Stream
  - Stream to Memory Mapped
- **Needs to stream to/from an allocated memory buffer**
  - PYNQ DMA class inherits from xlnk for memory allocation

PYNQ接口 - DMA类使用案例

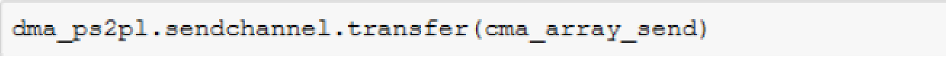
- Setup DMAs
- Allocate memory buffers
- Start DMA transactions

Create DMA instances



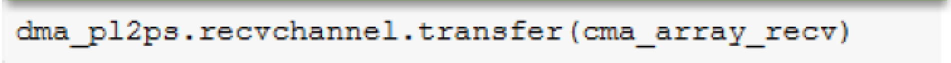
```
from pynq.lib import DMA

dma_ps2pl_description = overlay.ip_dict['axi_dma_from_ps_to_pl']
dma_ps2pl = DMA(dma_ps2pl_description)

dma_pl2ps_description = overlay.ip_dict['axi_dma_from_pl_to_ps']
dma_pl2ps = DMA(dma_pl2ps_description)
```

DMA from DDR to Device

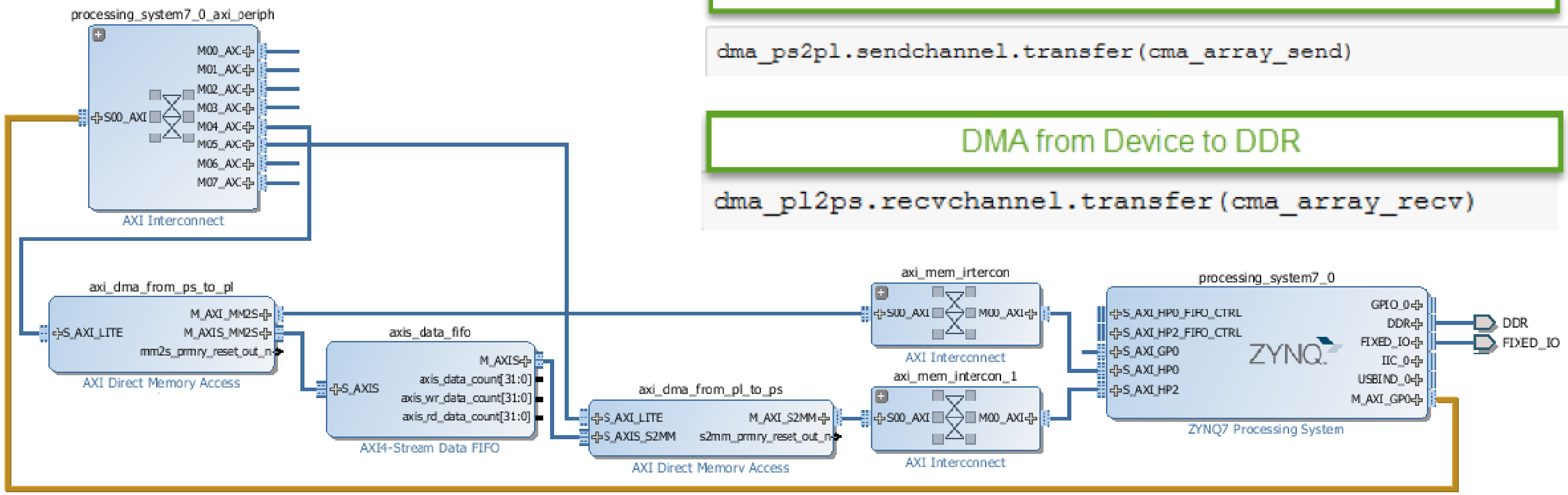


```
dma_ps2pl.sendchannel.transfer(cma_array_send)
```

DMA from Device to DDR

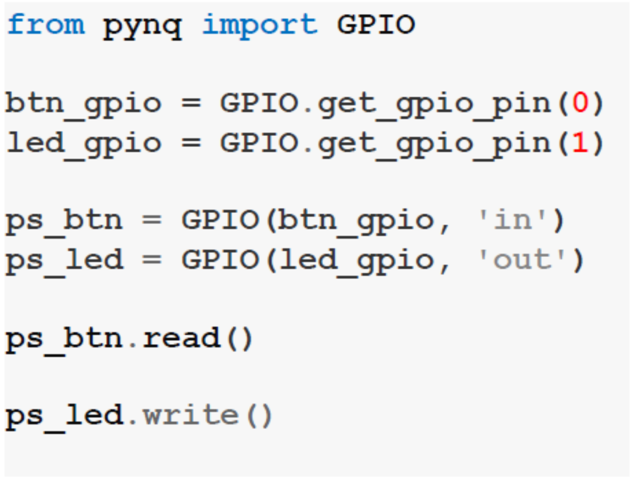


```
dma_pl2ps.recvchannel.transfer(cma_array_recv)
```



PYNQ接口 - GPIO类

- Up to 64 GPIO wires from PS
  - Tri-state
- Accessed from Linux
- Setup GPIO instance
- Map to GPIO pin
  - Translated pin numbers to Linux GPIO number
  - get_gpio_pin()
- Read/Write
- Most appropriate for simple control
  - Set, reset, start, done …

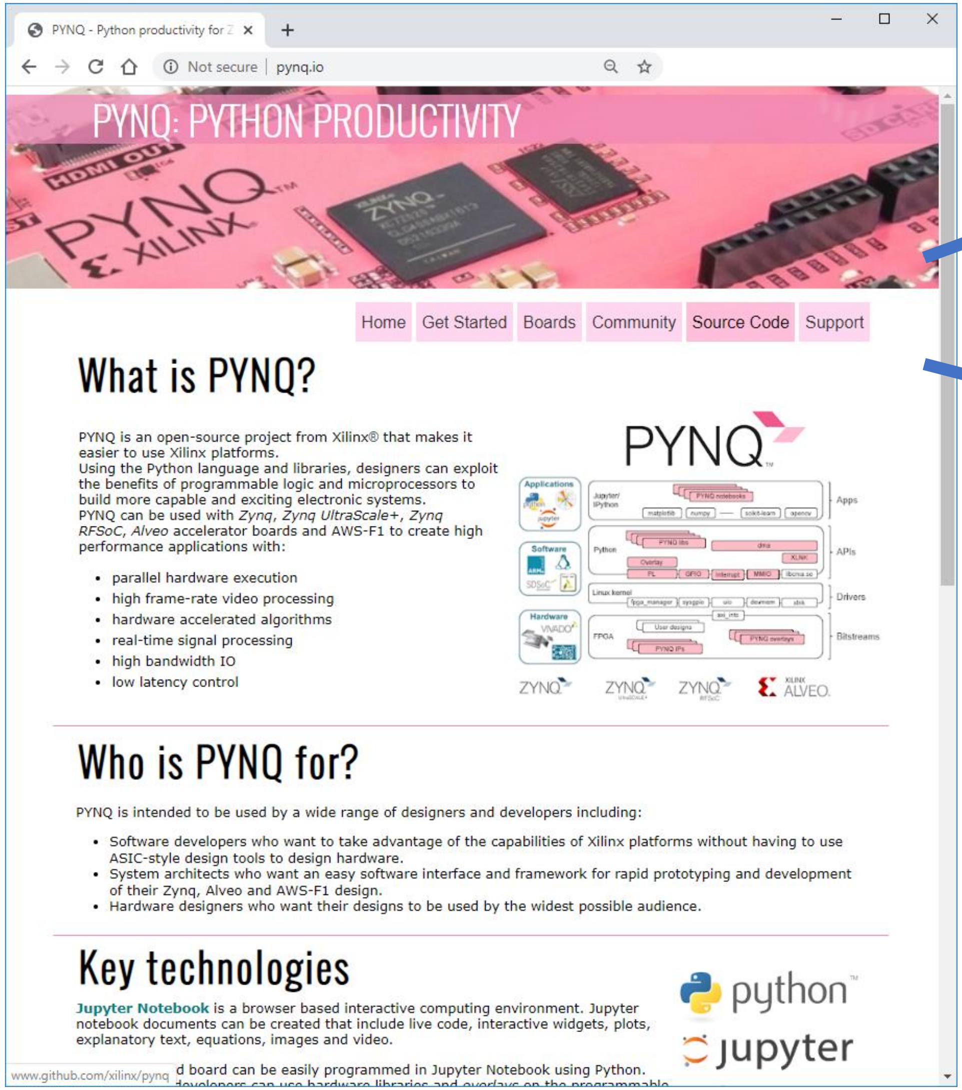


```
from pynq import GPIO

btn_gpio = GPIO.get_gpio_pin(0)
led_gpio = GPIO.get_gpio_pin(1)

ps_btn = GPIO(btn_gpio, 'in')
ps_led = GPIO(led_gpio, 'out')

ps_btn.read()

ps_led.write()
```

Next Steps

Getting Started with PYNQ

Find Out More at pynq.io



What is PYNQ?

PYNQ is an open-source project from Xilinx® that makes it easier to use Xilinx platforms.
Using the Python language and libraries, designers can exploit the benefits of programmable logic and microprocessors to build more capable and exciting electronic systems.
PYNQ can be used with *Zynq*, *Zynq UltraScale+*, *Zynq RFSoC*, *Alveo* accelerator boards and AWS-F1 to create high performance applications with:

- parallel hardware execution
- high frame-rate video processing
- hardware accelerated algorithms
- real-time signal processing
- high bandwidth IO
- low latency control

Who is PYNQ for?

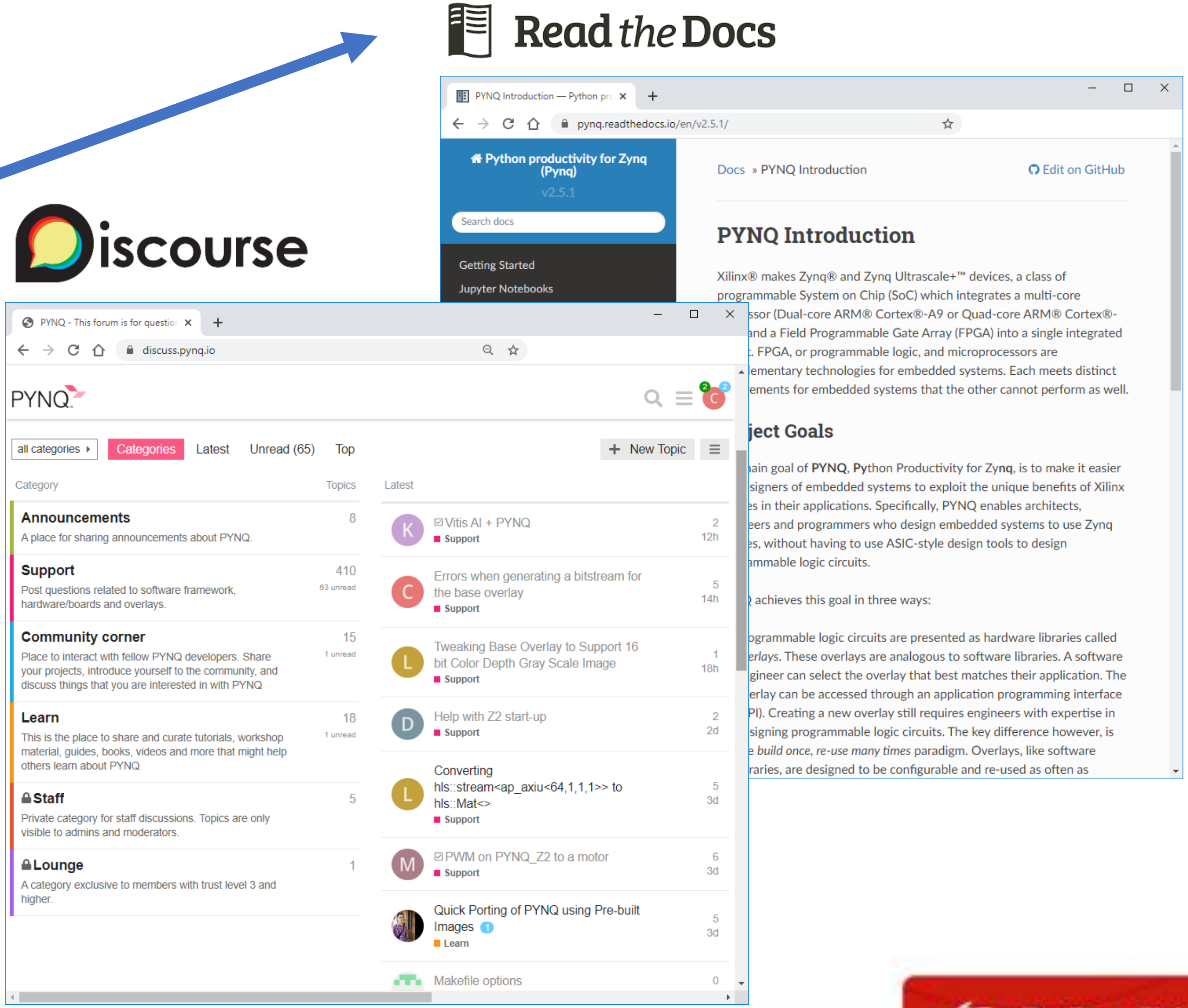
PYNQ is intended to be used by a wide range of designers and developers including:

- Software developers who want to take advantage of the capabilities of Xilinx platforms without having to use ASIC-style design tools to design hardware.
- System architects who want an easy software interface and framework for rapid prototyping and development of their Zynq, Alveo and AWS-F1 design.
- Hardware designers who want their designs to be used by the widest possible audience.

Key technologies

Jupyter Notebook is a browser based interactive computing environment. Jupyter notebook documents can be created that include live code, interactive widgets, plots, explanatory text, equations, images and video.

d board can be easily programmed in Jupyter Notebook using Python.
developers can use hardware libraries and overlays on the programmable



PYNQ Introduction

Xilinx® makes Zynq® and Zynq Ultrascale+™ devices, a class of programmable System on Chip (SoC) which integrates a multi-core
ssor (Dual-core ARM® Cortex®-A9 or Quad-core ARM® Cortex®-
and a Field Programmable Gate Array (FPGA) into a single integrated
. FPGA, or programmable logic, and microprocessors are
lementary technologies for embedded systems. Each meets distinct
ements for embedded systems that the other cannot perform as well.

ject Goals

ain goal of **PYNQ**, **Py**thon Productivity for Zy**nq**, is to make it easier
signers of embedded systems to exploit the unique benefits of Xilinx
es in their applications. Specifically, PYNQ enables architects,
eers and programmers who design embedded systems to use Zynq
es, without having to use ASIC-style design tools to design
ammable logic circuits.

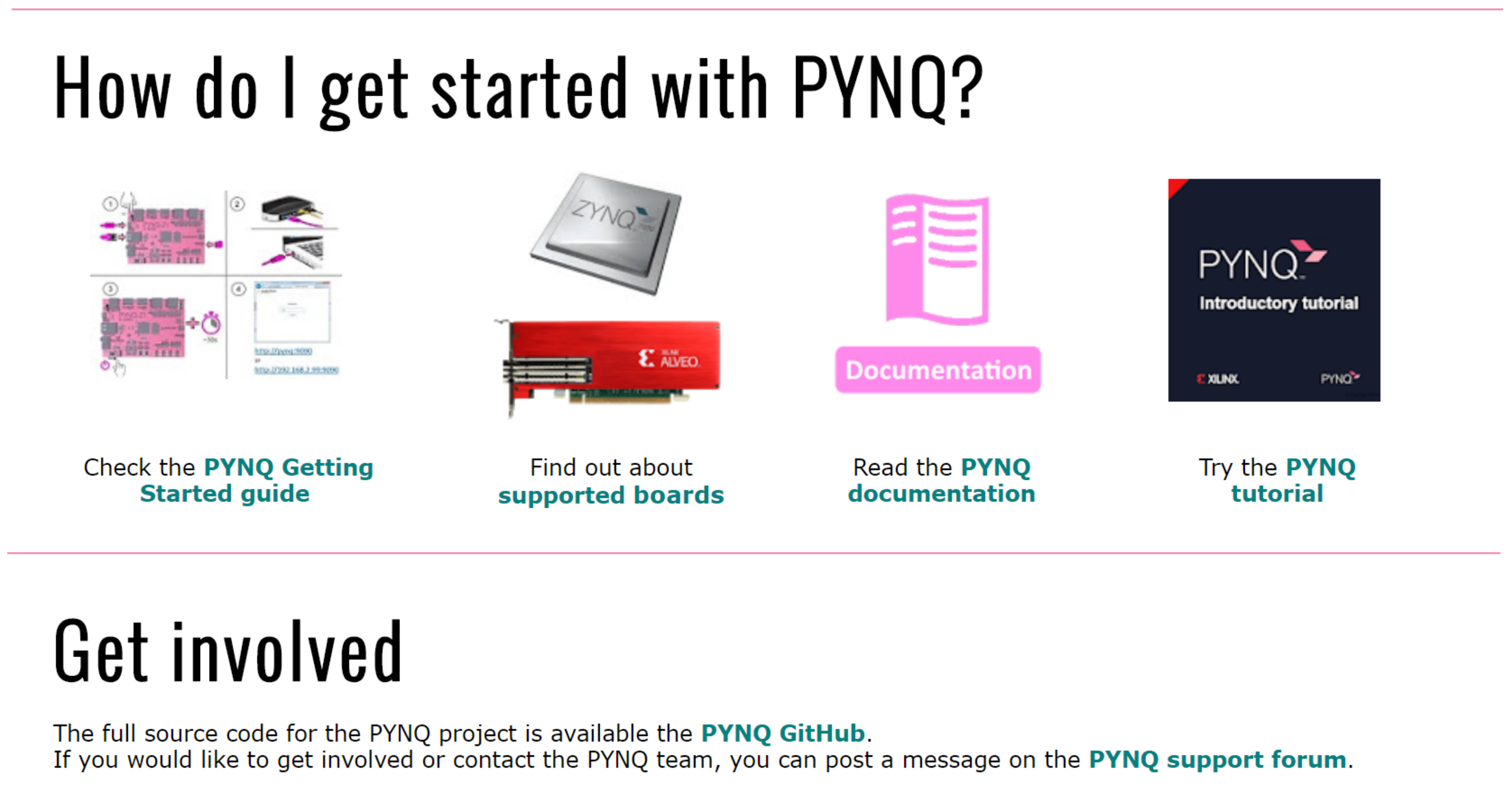
achieves this goal in three ways:

ogrammable logic circuits are presented as hardware libraries called
erlays. These overlays are analogous to software libraries. A software
gineer can select the overlay that best matches their application. The
erlay can be accessed through an application programming interface
PI). Creating a new overlay still requires engineers with expertise in
signing programmable logic circuits. The key difference however, is
e *build once, re-use many times* paradigm. Overlays, like software
raries, are designed to be configurable and re-used as often as

Category	Topics
Announcements A place for sharing announcements about PYNQ.	8
Support Post questions related to software framework, hardware/boards and overlays.	410 63 unread
Community corner Place to interact with fellow PYNQ developers. Share your projects, introduce yourself to the community, and discuss things that you are interested in with PYNQ	15 1 unread
Learn This is the place to share and curate tutorials, workshop material, guides, books, videos and more that might help others learn about PYNQ	18 1 unread
Staff Private category for staff discussions. Topics are only visible to admins and moderators.	5
Lounge A category exclusive to members with trust level 3 and higher.	1

Latest		
K	☑ Vitis AI + PYNQ Support	2 12h
C	Errors when generating a bitstream for the base overlay Support	5 14h
L	Tweaking Base Overlay to Support 16 bit Color Depth Gray Scale Image Support	1 18h
D	Help with Z2 start-up Support	2 2d
L	Converting hls::stream<ap_axiu<64,1,1,1>> to hls::Mat<> Support	5 3d
M	☑ PWM on PYNQ_Z2 to a motor Support	6 3d
	Quick Porting of PYNQ using Pre-built Images 1 Learn	5 3d
	Makefile options	0

More Information on Supported Boards at pynq.io



How do I get started with PYNQ?

Check the **PYNQ Getting Started guide**

Find out about **supported boards**

Read the **PYNQ documentation**

Try the **PYNQ tutorial**

Get involved

The full source code for the PYNQ project is available the **PYNQ GitHub**.
If you would like to get involved or contact the PYNQ team, you can post a message on the **PYNQ support forum**.

pynq.io

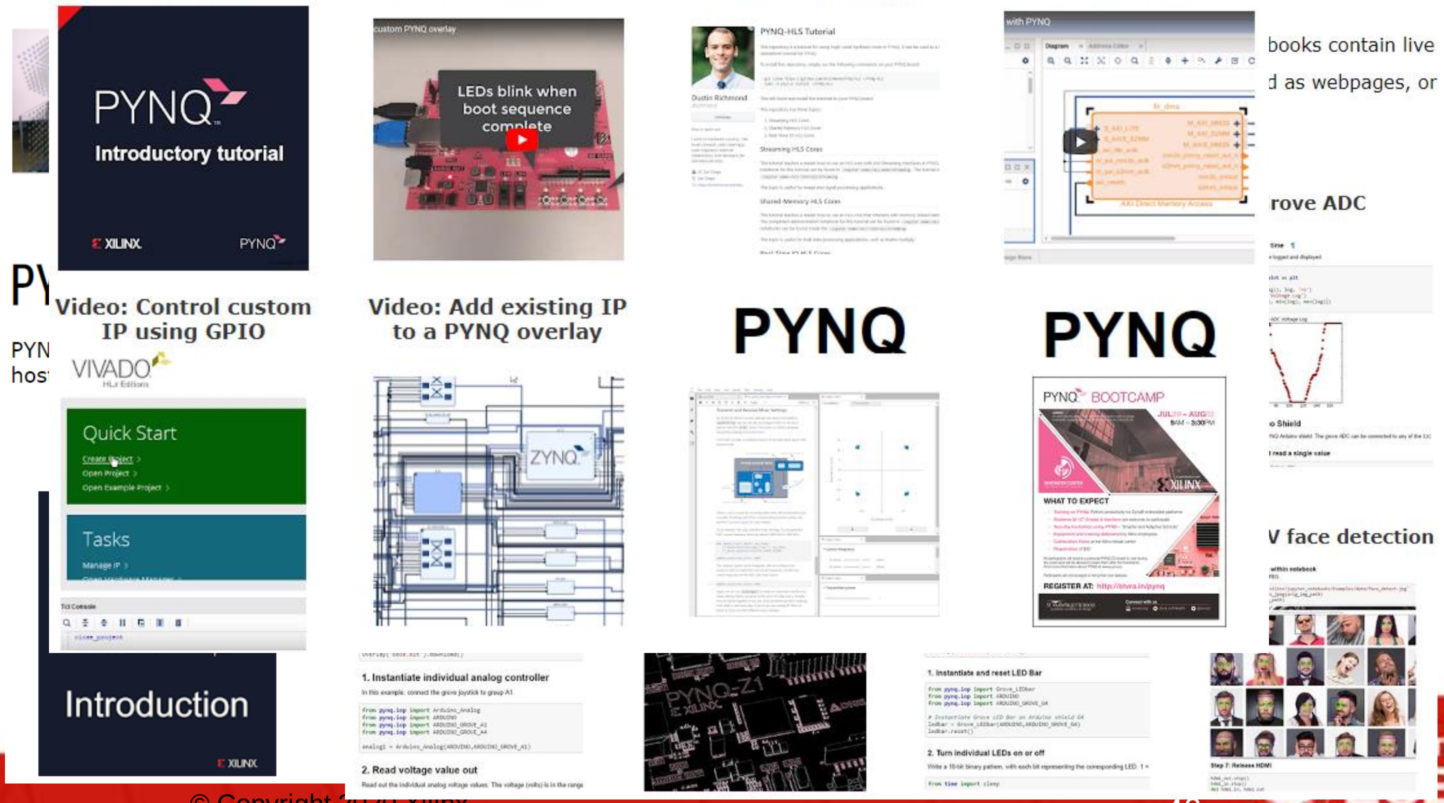
Home | Get Started | PYNQ-Z1 Board | Community | Source Code | Support

Community Projects

Tutorials and other resources

Example Notebooks

Tutorials and other resources



PYNQ Tutorial workshop

HLS filter example (FPGA Developer)

PYNQ HLS tutorial (Dustin Richmond)

Accelerate FIR function (FPGA Developer)

Video: Control custom IP using GPIO

Video: Add existing IP to a PYNQ overlay

1. Instantiate individual analog controller

2. Read voltage value out

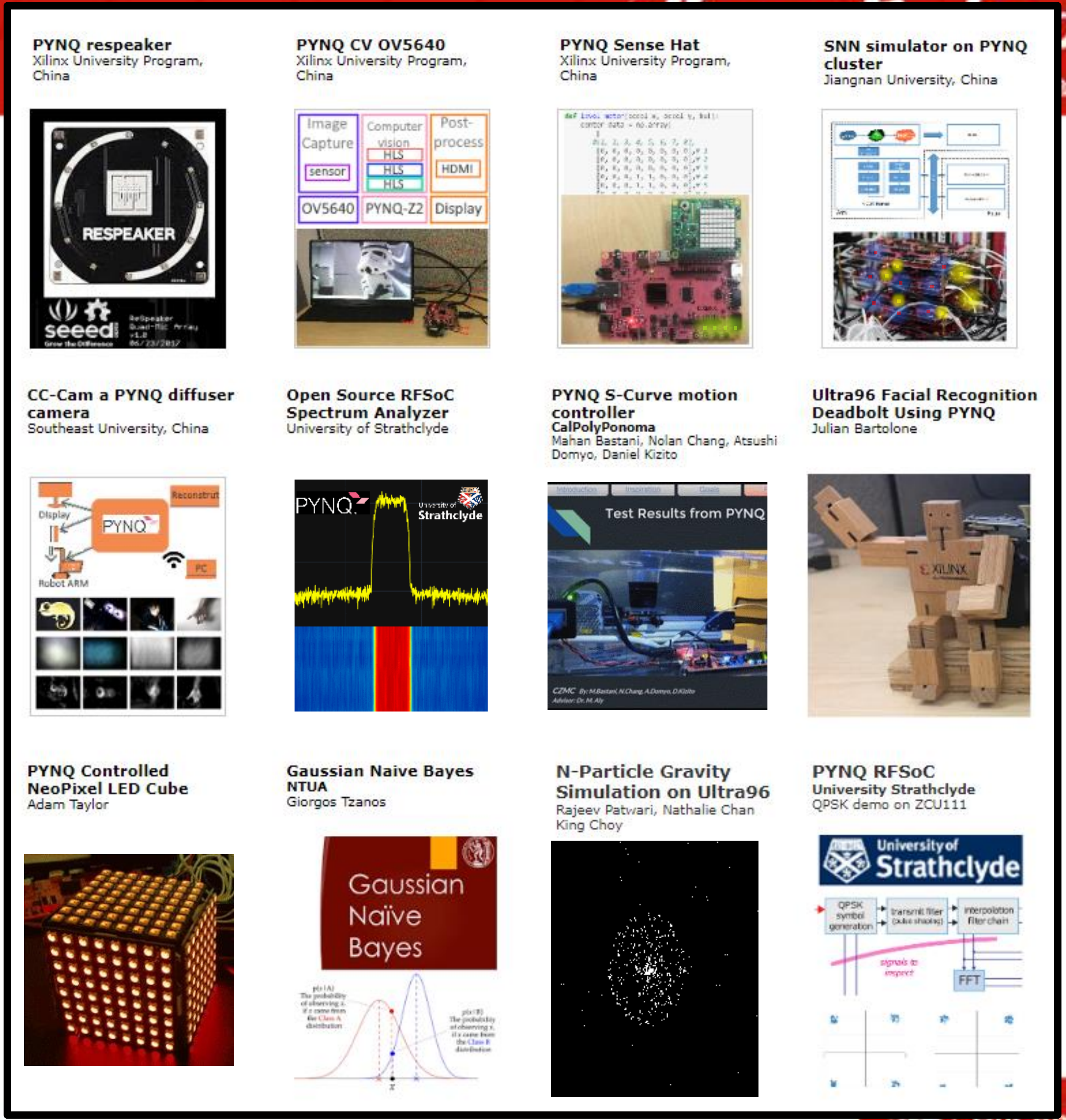
1. Instantiate and reset LED Bar

2. Turn individual LEDs on or off

© Copyright 2020 Xilinx

PYNQ Community

www.pynq.io/community.html



PYNQ respeaker
Xilinx University Program, China

PYNQ CV OV5640
Xilinx University Program, China

PYNQ Sense Hat
Xilinx University Program, China

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
CalPolyPonoma
Mahan Bastani, Nolan Chang, Atsushi Domyo, Daniel Kizito

Ultra96 Facial Recognition Deadbolt Using PYNQ
Julian Bartolone

PYNQ Controlled NeoPixel LED Cube
Adam Taylor

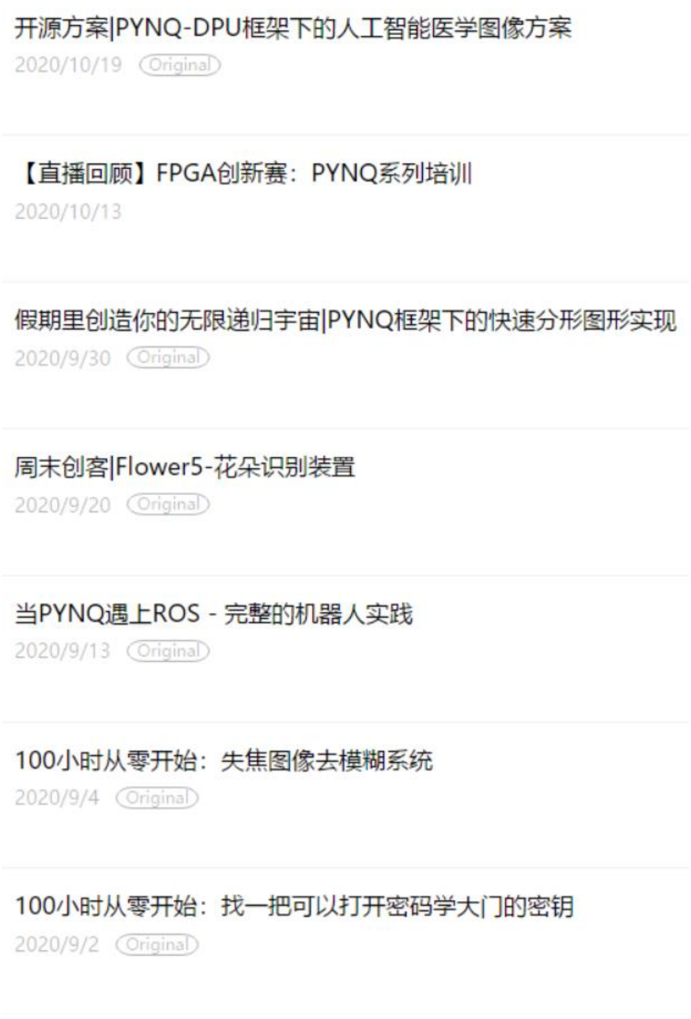
Gaussian Naive Bayes
NTUA
Giorgos Tzanos

N-Particle Gravity Simulation on Ultra96
Rajeev Patwari, Nathalie Chan King Choy

PYNQ RFSoC
University Strathclyde
QPSK demo on ZCU111

PYNQ开源社区 公众号

- Q1CY21 PYNQ 工作坊
  - 工作坊1：PYNQ框架初探
    - 尝试利用Python调用硬件资源
  - 工作坊2：PYNQ 物联网应用
    - 结合Azure IoT完成设计
  - 工作坊3：PYNQ 定制计算加速
    - 深入定制硬件加速设计



开源方案|PYNQ-DPU框架下的人工智能医学图像方案
2020/10/19 Original

【直播回顾】FPGA创新赛：PYNQ系列培训
2020/10/13

假期里创造你的无限递归宇宙|PYNQ框架下的快速分形图形实现
2020/9/30 Original

周末创客|Flower5-花朵识别装置
2020/9/20 Original

当PYNQ遇上ROS - 完整的机器人实践
2020/9/13 Original

100小时从零开始：失焦图像去模糊系统
2020/9/4 Original

100小时从零开始：找一把可以打开密码学大门的密钥
2020/9/2 Original

Wrap-up

- The use of Python in embedded and edge systems is growing
- Programmable platforms such as Zynq extend existing microprocessor, microcontroller and FPGA capabilities to enable more innovative and powerful designs
- PYNQ opens up programmable platforms to Python programmers in a Pythonic way
- PYNQ enhances the productivity of hardware designers and makes it possible for them to share their programable platform designs with the much larger Python community
- We invite you to explore the PYNQ framework and welcome contributions to the open source PYNQ community

THANK YOU