Rapid Application Development with Python and Ultra96

Author: Fred Kellerman
Avnet Ultra96 PYNQ Evangelist
DeveloperWeek June 2019 New York NY

AVNET
Reach Further™

96Boards
Partner

XILINX

python™

ZYNQ UltraSCALE+

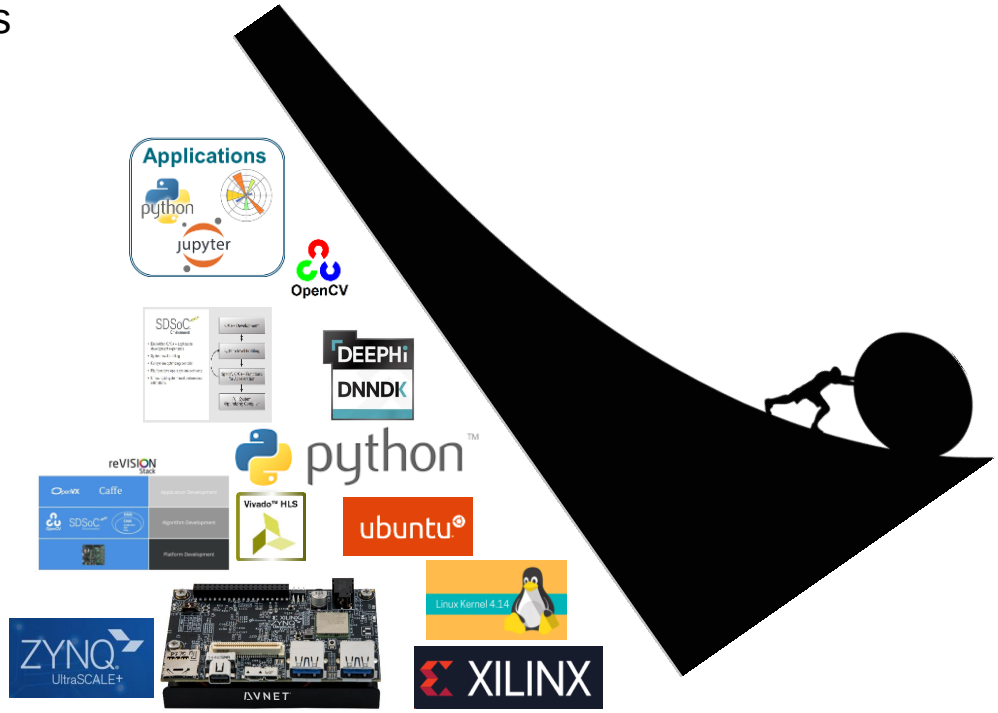
PYNQ™

Overview

- Intro to Ultra96 board
- Up and running with PYNQ in minutes
- Jupyter Notebooks and Labs
- Intro to programmable hardware
- Programming hardware with software
- Interfacing Python with hardware
- Working example: Image Resizer

Python for ZYNQ™

PYNQ™



AVNET

The board and the software

Ultra96 board features

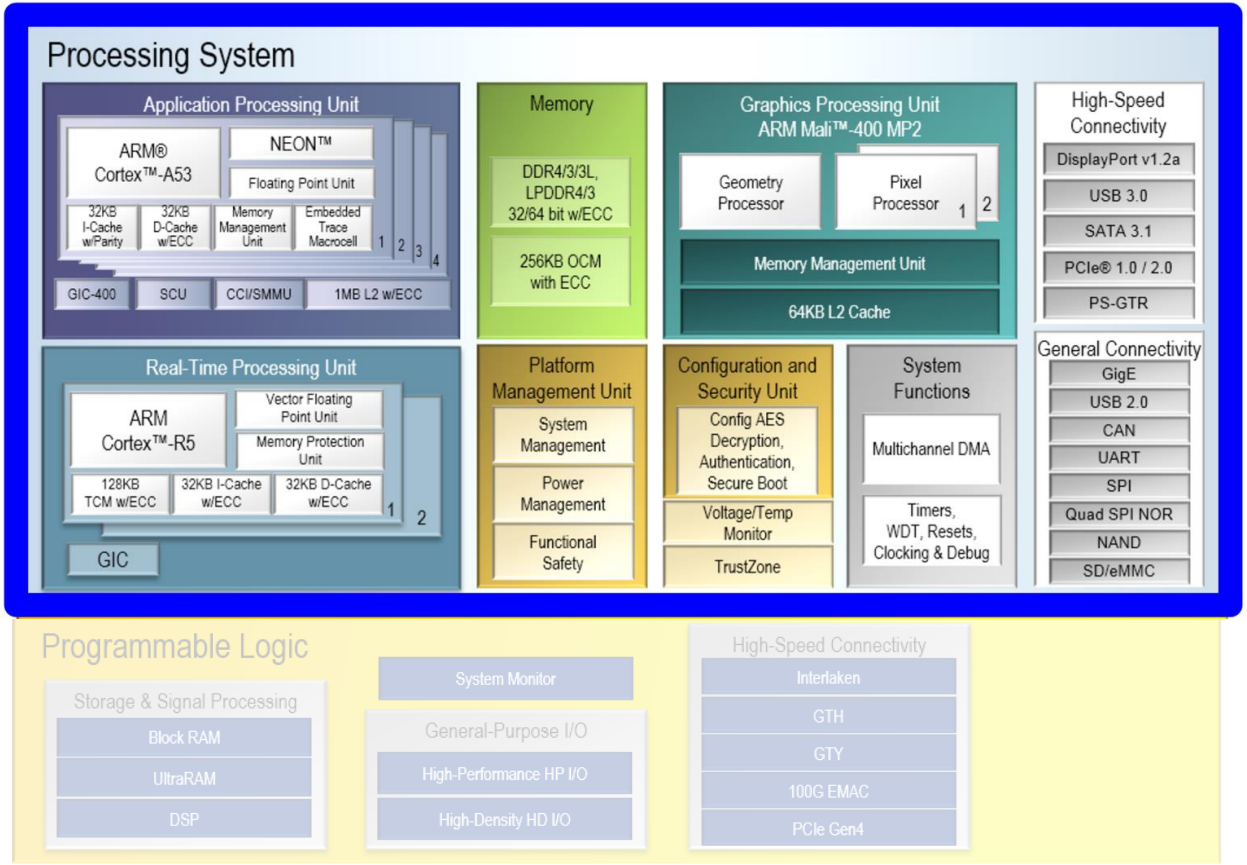
Ultra96
High performance edge computing

- Linaro 96Boards Consumer Edition compatible
- 85mm x 54mm form factor

Obtain Ultra96 board from Avnet or affiliate

AVNET

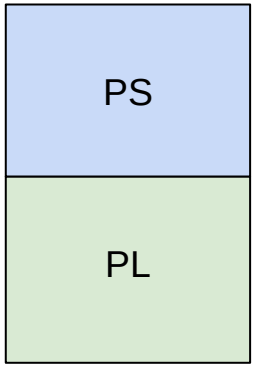
PS + PL = MPSoC Zynq



Ultra96's ZU3EG
ZYNQ UltraScale+ MPSoC

PS = Processing System

PL = Programmable Logic (FPGA)



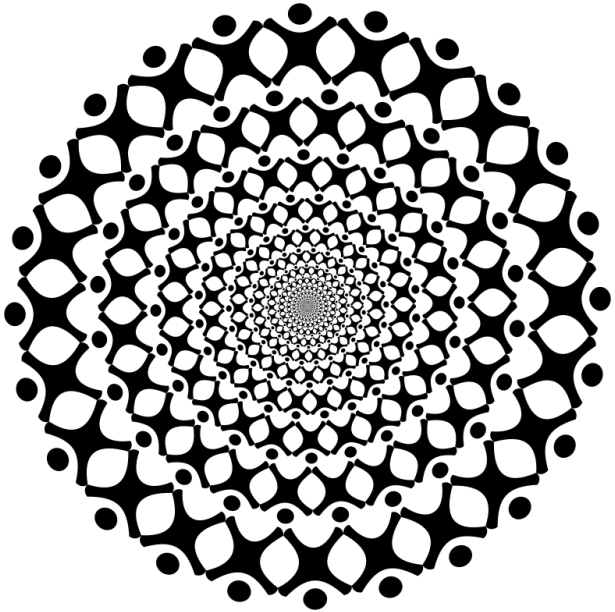
AVNET

What is and who is PYNQ for?

- Python developers who want to use the capabilities of Ultra96 and Xilinx programmable hardware
- Developers who need an Open Source Linux based rapid prototyping environment
- Anyone who finds a Raspberry Pi or the like useful should consider PYNQ for Ultra96

The PYNQ framework provides:

  - Built-in design presentation and interaction
  - Access and reuse of the Open Source universe
  - Internet and IoT interaction and connectivity
  - Machine learning
  - Parallel hardware access from Python
  - Video processing
  - Hardware accelerated algorithms
  - Real-time signal processing
  - High bandwidth IO
  - Low latency control
  - And much much more....

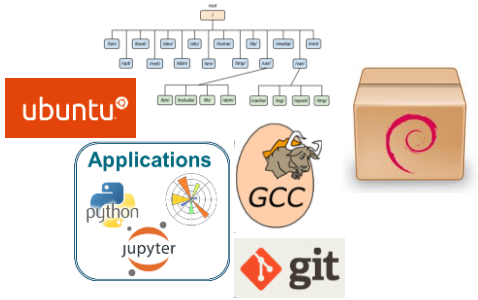


AVNET

What is Ultra96 PYNQ v2.4 made of?

It is a union of 3 major Open Source pieces that turn the Ultra96 into a complete Linux based distribution, palm of your hand computing system:

Ubuntu Bionic aarch64 Root FS with **Python 3.6**



\+

2018.3 Xilinx aarch64 PetaLinux

\+

Xilinx PYNQ Python module and classes

=

(.img)

\+ 100's more onboard + 1000's more available on the fly through "apt"

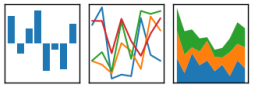
Debian Package Manager

v2.4 PYNQ works with Xilinx 2018.3 tool versions only!

AVNET

Use powerful Python toolboxes

pandas



matplotlib

SciPy

NumPy

plotly

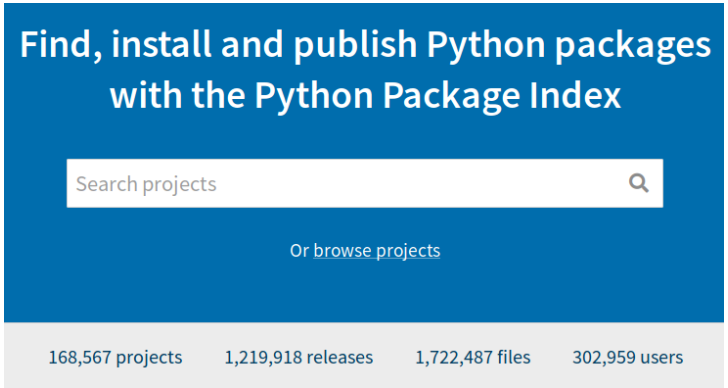
pillow

Twisted Matrix Labs
Building the engine of your Internet

IP[y]: IPython
Interactive Computing

python Package Index

https://pypi.org



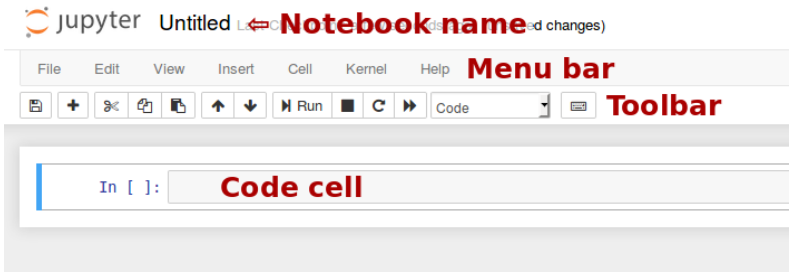
“sudo pip3 install <url, package>”
“sudo apt install python3-<package>”

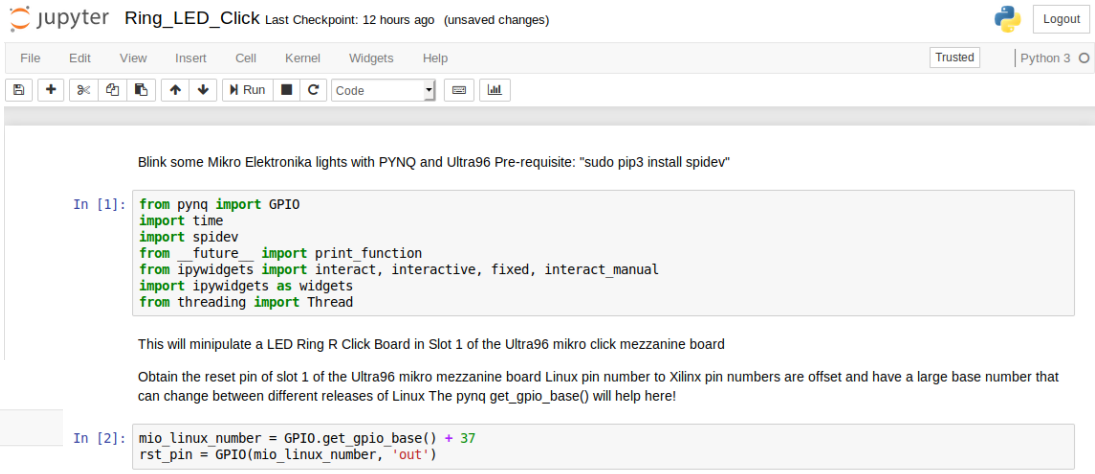
AVNET

Jupyter Notebook user web browser interface

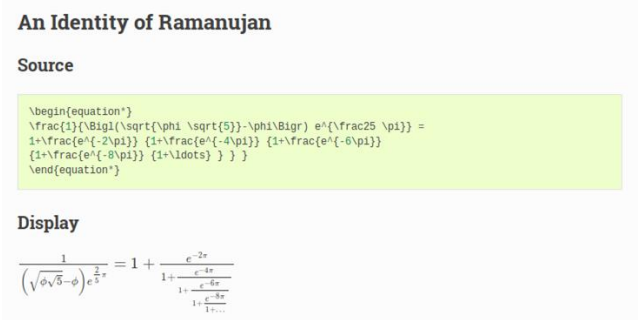
Cell Types:

- Python code
- Markdown (Github)
- iPython input and output





Markdown support for Math Equations LaTeX:



Recommended browsers:

- Firefox, Chrome and Safari

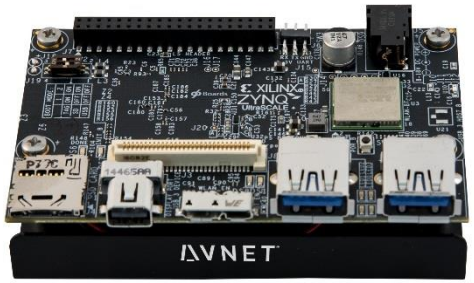
AVNET

Getting started: required parts

For Ultra96 V1:
http://avnet.me/ultra96-pynq-image-v2.4

For Ultra96 V2:
http://avnet.me/ultra96-pynq-image-v2.4_v2

(1.5GB download)



USB 2 (or 3)

>= 16GB
Delkin, Sandisk, others?
Samsung EVO won't work

balenaEtcher
(or Win32DiskImager)

240-100VAC 50/60Hz 12VDC 4A

AES-ACC-U96-4APWR

Host PC, Laptop:
Windows 10 (or 7),
Linux, OSX,
Chromebook

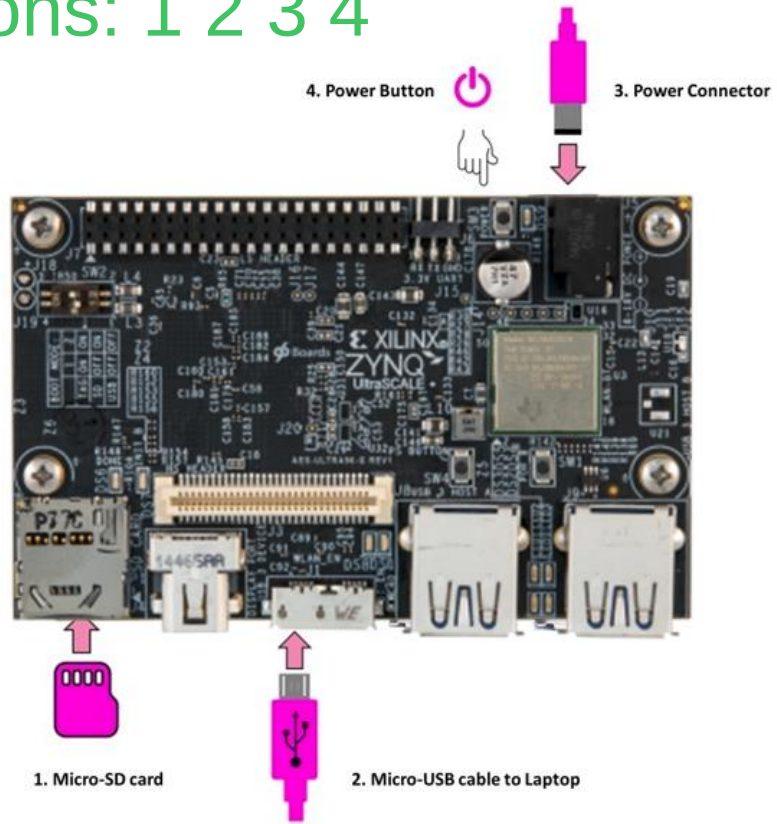
Microsoft ubuntu®

AVNET

Simple assembly instructions: 1 2 3 4

Please disconnect power supply from Ultra96:

1. Download ultra96_v2.4.zip, extract Ultra96_v2.4.img
   a. Copy .img to micro SD card with Etcher.
   b. Insert SD card in Ultra96
2. Connect USB cable between Ultra96 and host PC (Windows 7/10, Linux, MAC)
3. Attach power supply and plug-in
4. Press Ultra96 power on button



USB Cable to PC provides: Serial Console and 192.168.3.1 RNDIS + SAMBA share (xilinx)

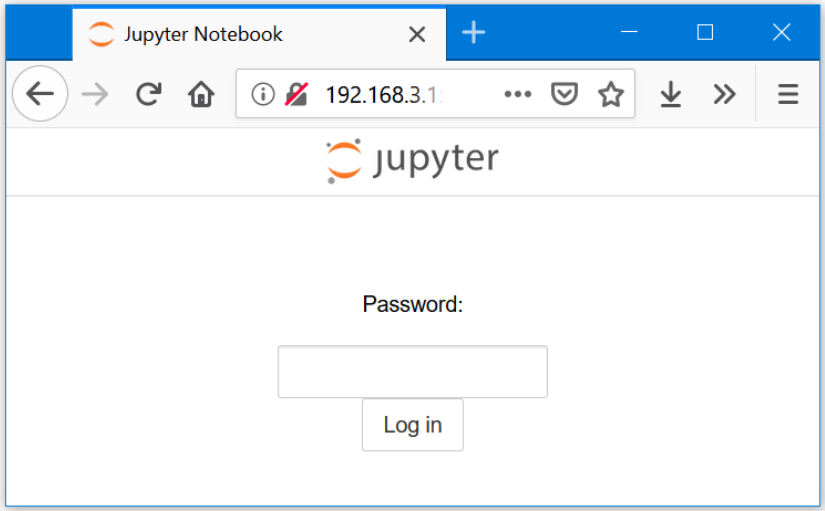
AVNET

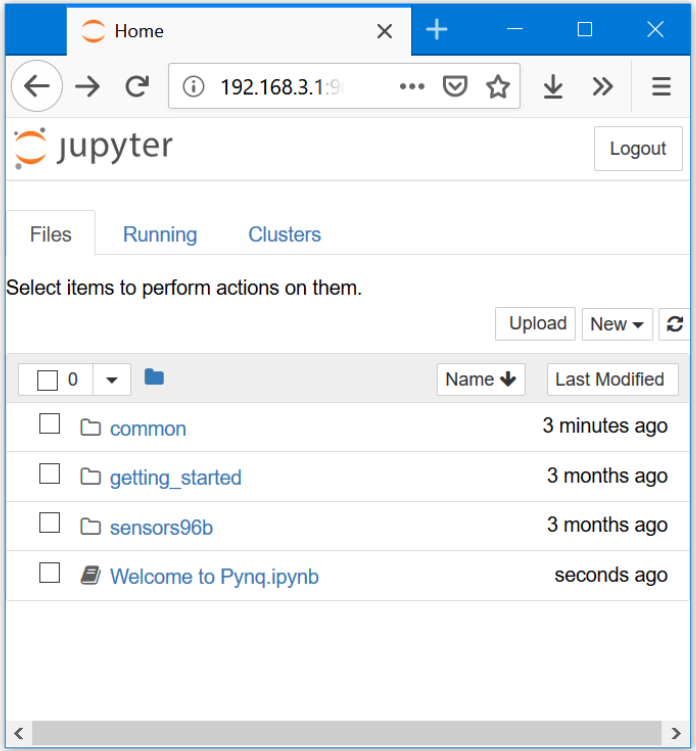
Lights, web-browser: Jupyter Notebook Action

On your host PC direct your web-browser to:

http://192.168.3.1

The password is: xilinx





See Appendix for tips on knowing if initial install is working properly!

AVNET

Example Python Jupyter Notebooks included:

PYNQ™

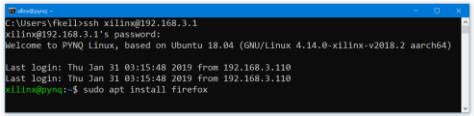
- Welcome to Pynq
- Connect to WLAN with Ultra96 WiFi
- Intro to Jupyter Notebooks
- PL Overlay Download
- Random Numbers
- Execute shell cmds from Python
- USB Camera capture / OpenCV
- Change PL SoC clocks from Python
- Python and PYNQ
- Advanced Jupyter Notebooks
- Grove mezzanine sensor demos

AVNET

Jupyter Labs: a development IDE in your browser

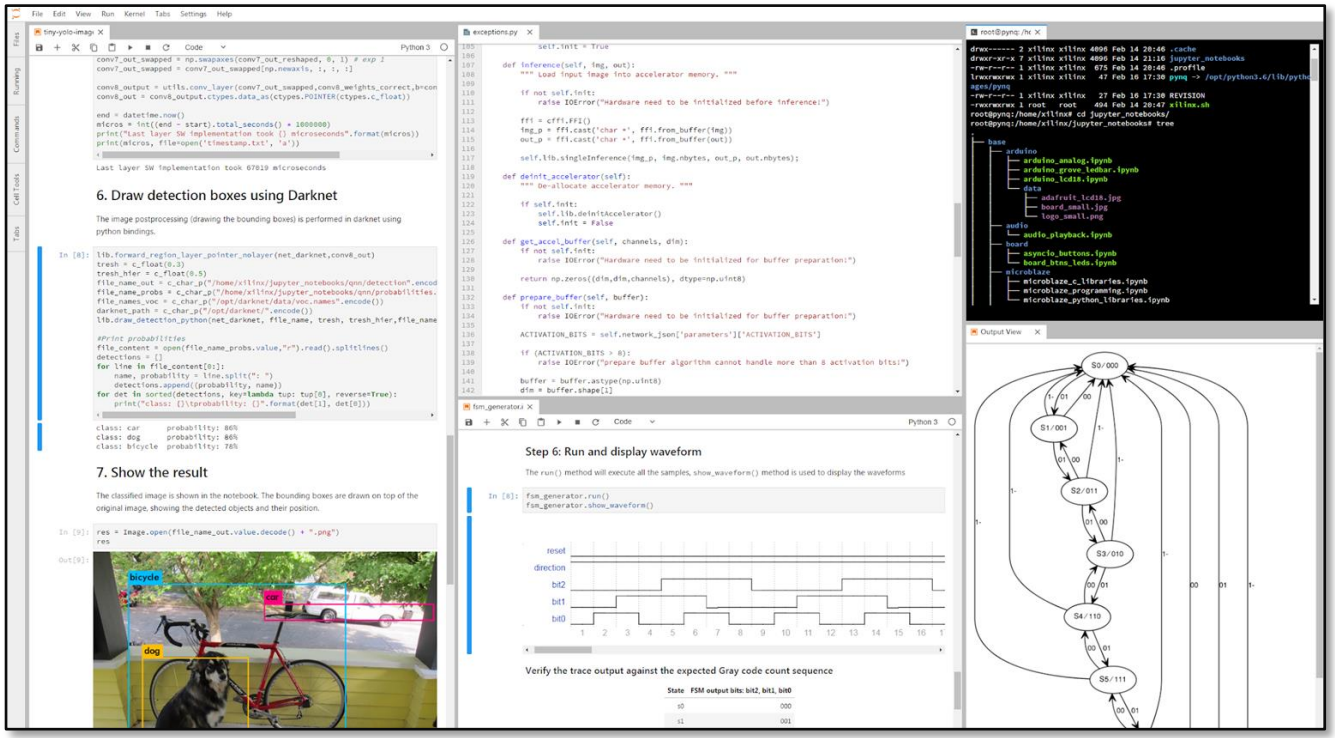
In your browser go to:
http://192.168.3.1/lab

(You can also ssh, user: xilinx, pass: xilinx)



Note: The image shown at left has had additional applications installed. Out of the box you will only have the example Jupyter Notebooks installed.

Jupyter Labs is written in JavaScript use Chrome or Firefox for best results. Yes there is a GUI debugger add-on available through a Python module from IBM: PixiDust



AVNET

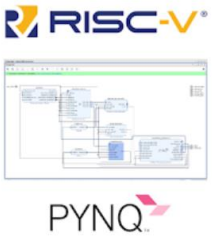
PYNQ Community: more Application Notebooks

PYNQ community projects

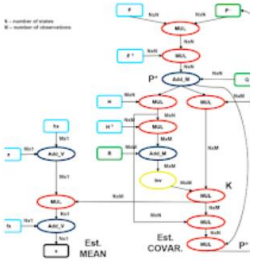
http://www.pynq.io/community.htm

PYNQ Hello World
Hardware accelerated image resizer example

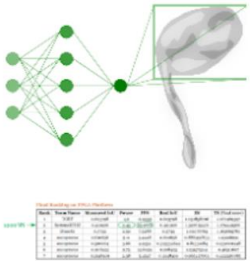
RISC-V on PYNQ
UCSD



Extended Kalman filter
University Sydney



spoonNN
ETH Zurich
FPGA-based neural network inference project



iSmart DNN
FPGA-based neural network inference for DAC 2018 contest

TGIIF
FPGA-based neural network inference for DAC 2018 contest

cv2PYNQ
FAU
Accelerated OpenCV image filtering library.

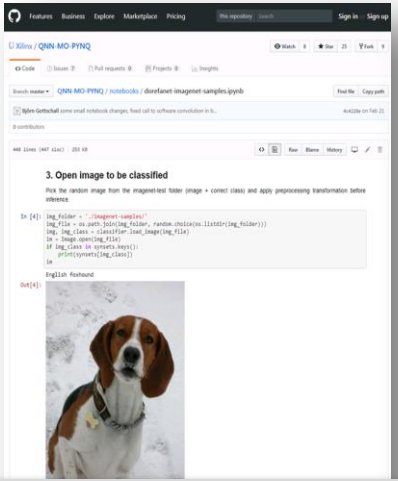
Video processing
KU Leuven
Hardware accelerated videoprocessing

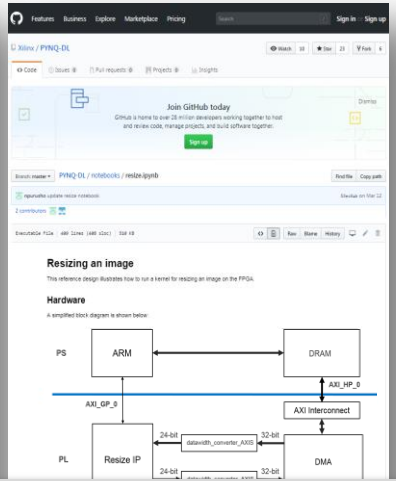
Jupyter Notebook
Python
PL Overlay/Bitstream

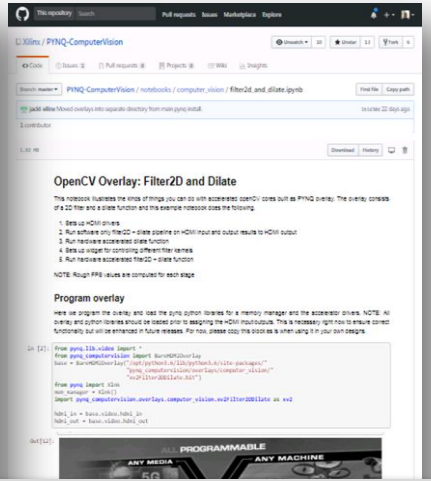
PYNQ Notebook

AVNET

Software-style Packaging & Distribution of Designs







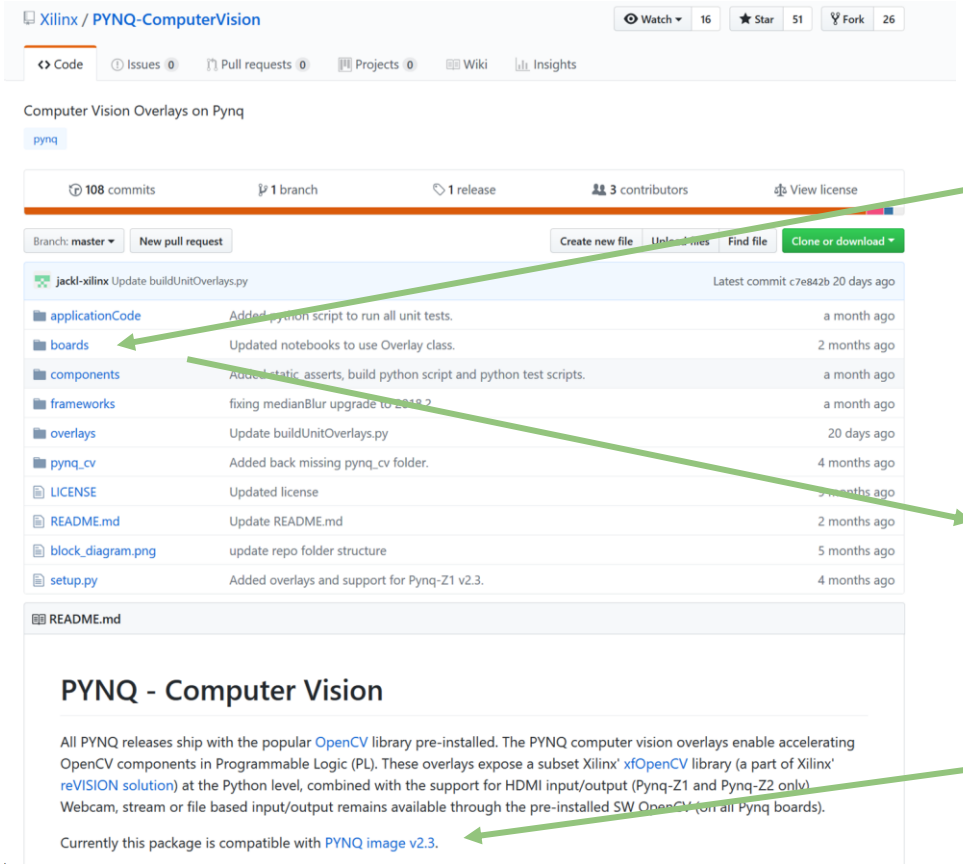
Install designs = Jupyter Notebook, Python, overlay/bitstream + support files from GitHub with a single shell command, example OpenCV demo cmd:

sudo pip3 install git+https://github.com/Xilinx/PYNQ-ComputerVision.git

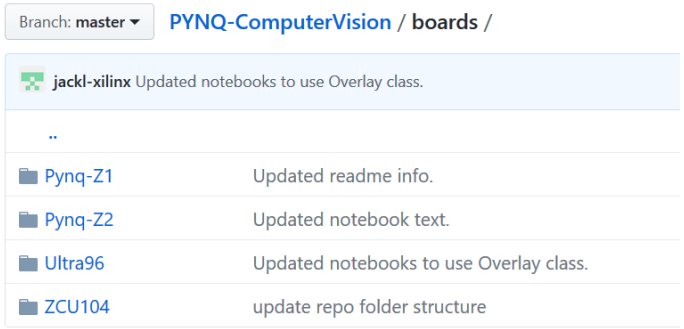
(Ultra96 *must* have internet access, see appendix)

AVNET

Checking for Notebook compatibility



Look here to see if Ultra96 is supported, not all notebooks have a board folder

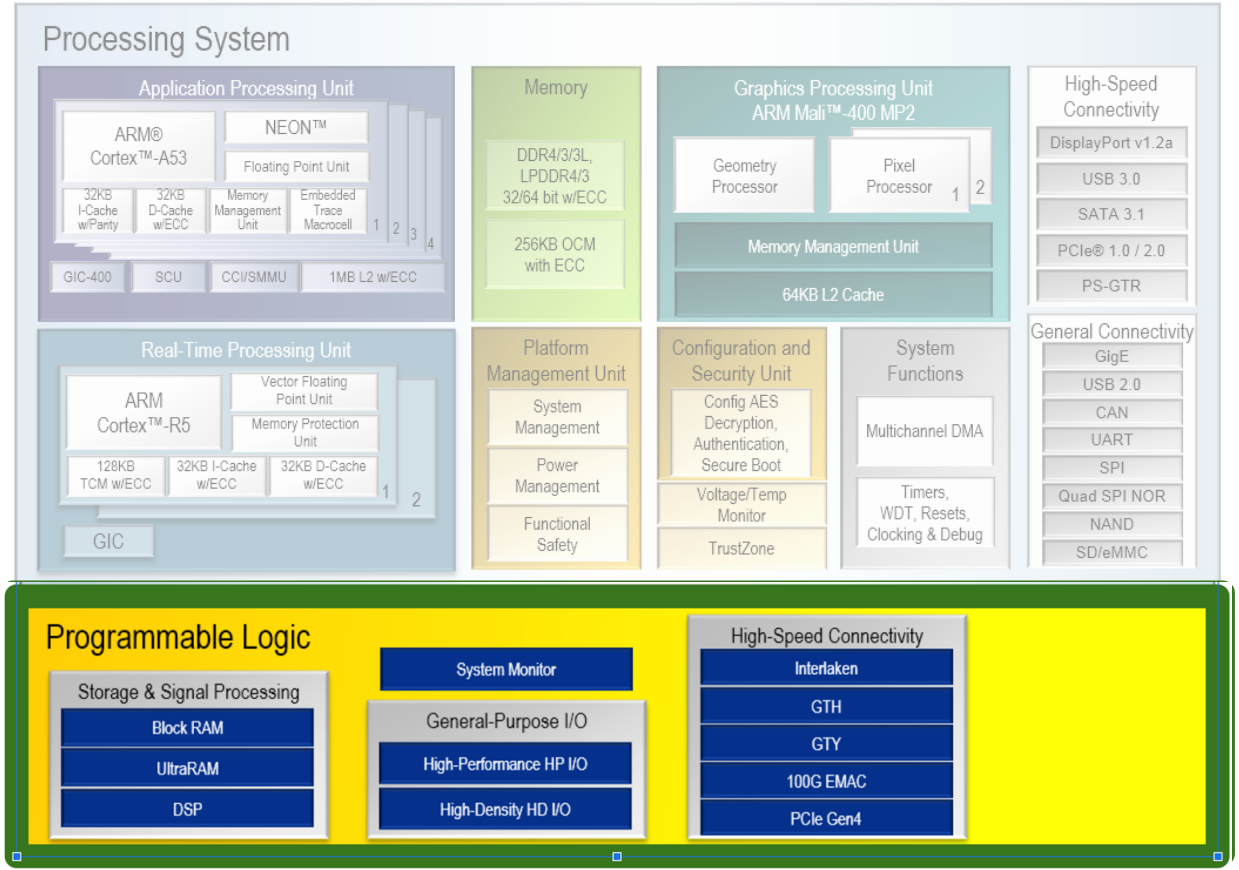


Double check PYNQ version as well!

AVNET

Through the Looking Glass: Programmable Hardware with Python

PS + PL = MPSoC Zynq



Ultra96's ZU3EG
ZYNQ UltraScale+ MPSoC

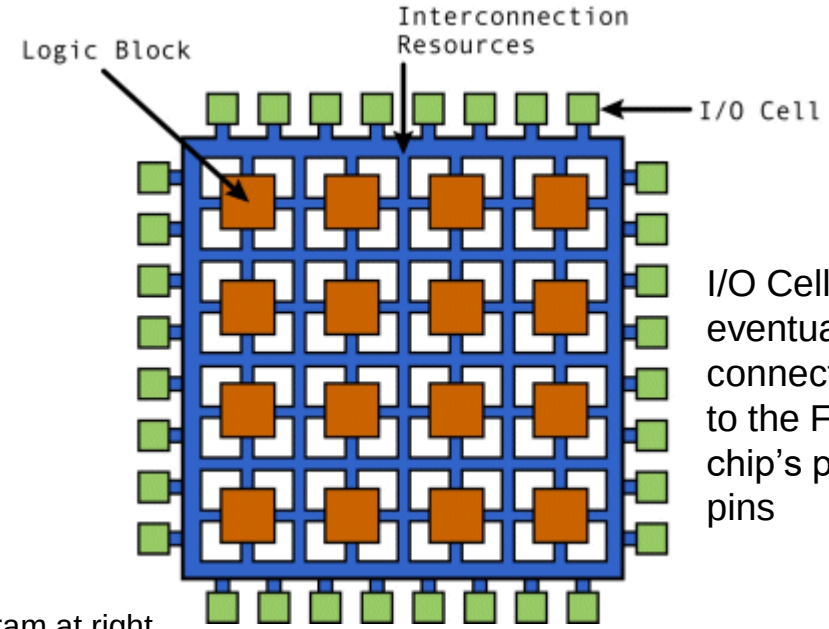
PS = Processing System

PL = Programmable Logic

Note: ZU3EG does not have UltraRAM or High-Speed Connectivity.

AVNET

PL/Field Programmable Gate Array, under the hood:



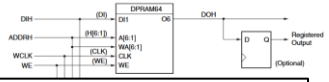
I/O Cells eventually connect directly to the FPGA chip's physical pins

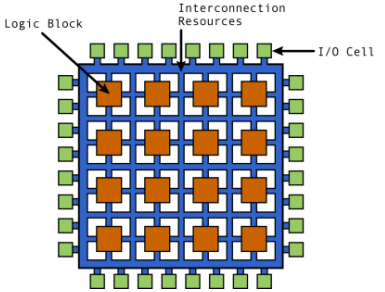
(The diagram at right is a simplification of what is inside the actual part!)

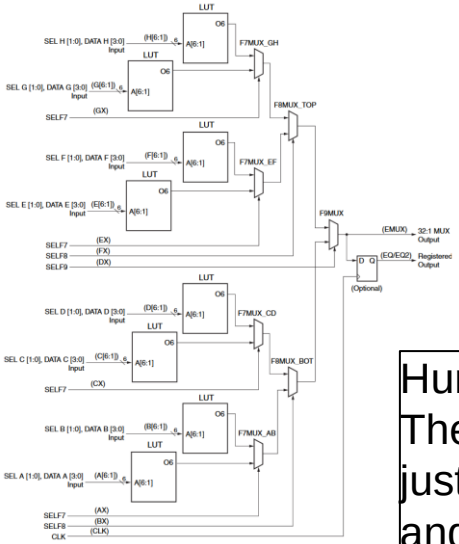
AVNET

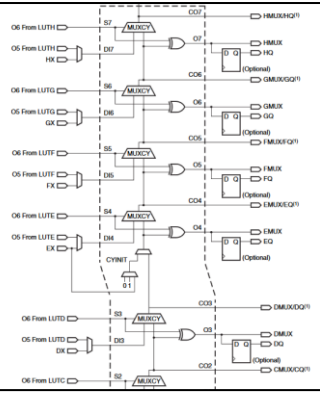
What's inside?

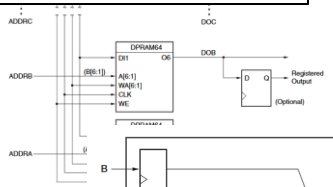
LUTs, Clocks, PLLs, Transceivers, Multiplexers, modulo 2 based Logic Functions, Interconnects, Arithmetic operators and Memories

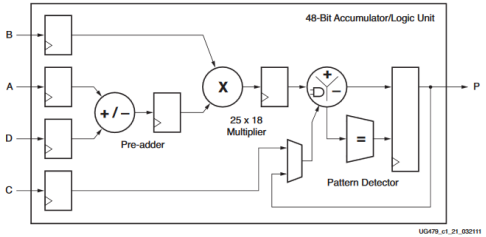












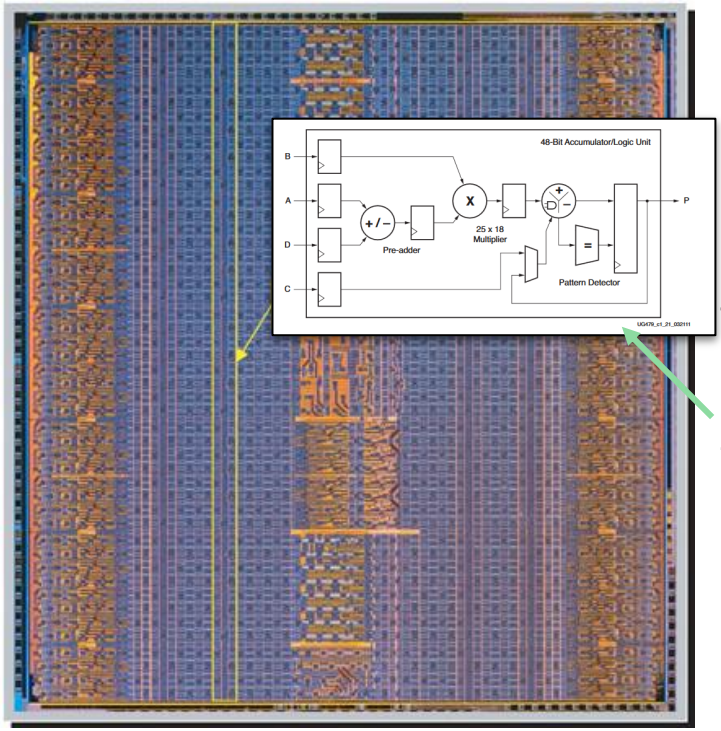
Hundreds of thousands of these programmable logic block widgets! They can be interconnected to create logic that can accomplish just about anything. Hardware Design Language source is synthesized and then place and routed (the tools do this for you). This is analogous to a compiler and linker.

Note: logic in ZU3eG is not exactly exactly as shown above but is similar

AVNET

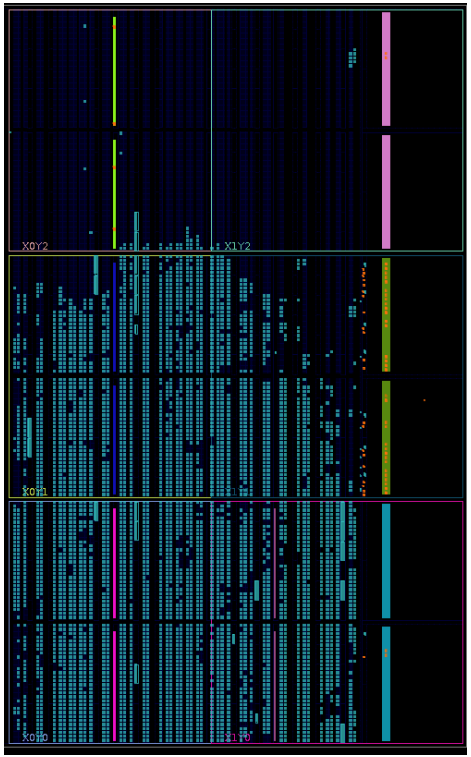
PL/FPGA innards

Raw chip die:
(not actual ZU3EG)



(360 parallel 48-bit MAC accelerators in the ZU3EG)

Compiler (synthesis) place and route output:



AVNET

Reasons to use a PL/FPGA

- For many solutions the PL will be orders of magnitude faster
- Precise timing capabilities (picoseconds jitter accuracy) for control of hardware
- Determinism of algorithmic execution (no cache, preemption, task switching, threads or interrupts if you design it that way)
- You can do things beyond what the PS can do with Python, even design your own CPU/GPU! See Xilinx's MicroBlaze™ for PYNQ: https://pynq.readthedocs.io/en/v2.0/pynq_libraries/pynq_microblaze_subsystem.html
- The art of designing hardware with software can be rewarding and enjoyable!

Example - machine learning computer vision on Ultra96:

- FINN Binary Neural Network (BNN) non-PYNQ demo
- http://www.wiki.xilinx.com/Zynq+UltraScale%EF%BC%8B+MPSoC+Accelerated+Image+Classification+via+Binary+Neural+Network+TechTip

Ultra96 FINN PYNQ Notebook:
https://github.com/Xilinx/BNN-PYNQ.git

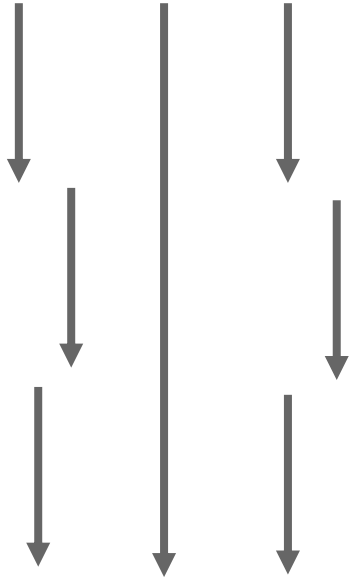
1080p Images per Second in HW: 66.3
1080p Images per Second in *SW: .01

HW Acceleration Factor: *6171

* The SW used for benchmark was running on the Ultra96 ARM Cortex™ A53 cores with same OS as the HW tests @ ~1.3GHz. Other platforms that have somewhat faster ARM cores could do a little better with just SW. Other platforms with their own hardware accelerators will also run faster than pure SW.

AVNET

Traditional cpu based programming



CPU programming is sequentially executed

Sure we have interrupts and soft processes and threads
(virtual parallelism)

And hardware threads and processes and multiple cores/CPUs
(true parallelism)

Interprocess communications are sequential execution strands
ordered through some kind of semaphore for synchronization.

Some parallelism to exploit and cope with for CPUs? Yes A relative lot? No

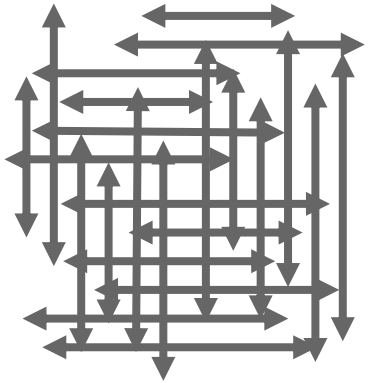
Easy to create and comprehend, can read top to bottom and follow the code...

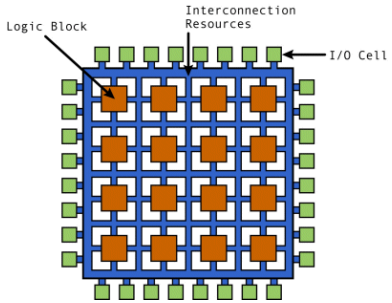
Parallel hardware programming mindshift

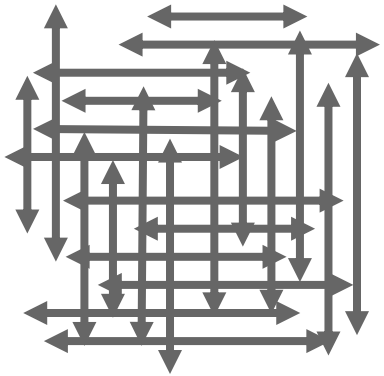
- Hardware “executes” all at once: extreme parallelism
  - What could you do with 360 multipliers simultaneously?

- “Interprocess communication” occurs at the speed of electrical signal propagation (which still gets in the way even at near the speed of light)

- Programmed with Hardware Design Languages (HDL): VHDL, Verilog







VHDL 4-bit decimal up counter example

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity bin_counter_A is
  port(                enable: in std_logic;
               clk: in std_logic;
               reset: in std_logic;
               output: out std_logic_vector(0 to 3));
end bin_counter_A;

architecture behavioral of bin_counter_A is
  signal count: std_logic_vector(0 to 3);
begin   process(enable, clk, reset)
  begin
    if reset='1' then
      count <= "0000";
    elsif(rising_edge(clk)) then
      if enable='0' then
        if count="1001" then
          count <="0000";
        else
          count <= count + 1;
        end if;
      end if;
    end if;
  end process;
  output <= count;
end behavioral;
```

Both programs give the same results, lines of code are not time ordered top to bottom !?!

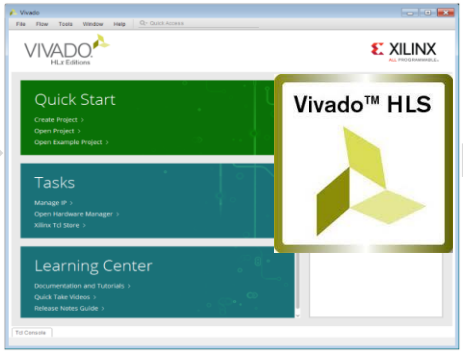
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity bin_counter_B is
  port(                enable: in std_logic;
               clk: in std_logic;
               reset: in std_logic;
               output: out std_logic_vector(0 to 3));
end bin_counter_B;

architecture behavioral of bin_counter_B is
  signal count: std_logic_vector(0 to 3);
begin   process(enable, clk, reset)
  begin
    output <= count;
    if reset='1' then
      count <= "0000";
    elsif(rising_edge(clk)) then
      if enable='0' then
        if count="1001" then
          count <="0000";
        else
          count <= count + 1;
        end if;
      end if;
    end if;
  end process;
end behavioral;
```

AVNET

Xilinx tools for software programmers to the rescue

Don’t have the time for the learning curve of HDL?
Then program PL hardware with
C/C++ using Xilinx HLS or SDSoC

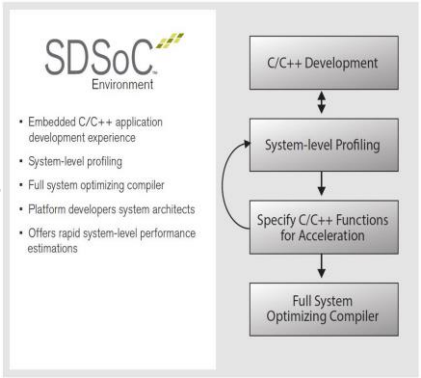
C/C++/OpenCL



PL Overlay/ bitstream

C/C++/OpenCL

(also requires Device Support Archive file, .dsa)



PL bitstream + .exe, .so

Note: Overlay/bitstream is a file that configures the PL hardware

AVNET

HLS/SDSoC hardware design

- Excellent fit for accelerator type architecture, also very good for other hardware-isms
- Some paradigms cannot be exploited (can intermix with HDL to overcome)
- Programmer is still responsible for exploitation of parallelism but to a lesser degree
- #pragma are used to tell compiler how to synthesize your code

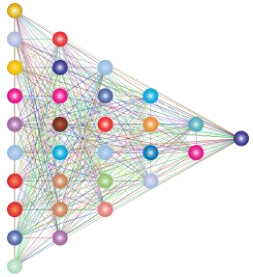
HLS Sobel Edge Detector fragment:

```
for(i = 0; i < height; i++){
     for(j = 0; j < width; j++){
          x_dir = 0;
          y_dir = 0;
          if((i > 0) && (i < (height-1)) && (j > 0) && (j<(width-1))){
          for(rowOffset = -1; rowOffset <= 1; rowOffset++){
             for(colOffset= -1; colOffset <= 1; colOffset++){
                x_dir = x_dir + input_image[i+rowOffset][j+colOffset] * Gx[1+rowOffset][1+colOffset];
                y_dir = y_dir + input_image[i+rowOffset][j+colOffset] * Gy[1+rowOffset][1+colOffset];
             }
          }
     edge_weight = ABS(x_dir) + ABS(y_dir);
     output_image[i][j] = edge_weight;
}
```

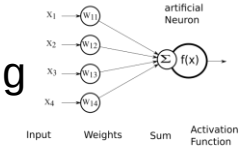
```
for(i = 0; i < height; i++){
     for(j = 0; j < width; j++){
#pragma HLS PIPELINE
          x_dir = 0;
          y_dir = 0;
          if((i > 0) && (i < (height-1)) && (j > 0) && (j<(width-1))){
          for(rowOffset = -1; rowOffset <= 1; rowOffset++){
              for(colOffset= -1; colOffset <= 1; colOffset++){
                 x_dir = x_dir + input_image[i+rowOffset][j+colOffset] * Gx[1+rowOffset][1+colOffs
                 y_dir = y_dir + input_image[i+rowOffset][j+colOffset] * Gy[1+rowOffset][1+colOffs
              }
          }
     edge_weight =ABS(x_dir) + ABS(y_dir);
     output_image[i][j] = edge_weight;
}
```

AVNET

Re-use: some useful Xilinx hardware libraries



- Xilinx Open Computer Vision functions accelerated in hardware
  - SDSoC/HLS C++ Xilinx xfOpenCV library
  - https://github.com/Xilinx/xfopencv
- Machine Learning (v2.4 of PYNQ using 2018.3)
  - Xilinx DeePhi DPU, an SDSoC C++ callable AI Inference engine
  - https://www.xilinx.com/support/documentation/user_guides/ug1331-dnndk-sdsoc-ug.pdf
- Xilinx Re-Vision stack (various support for 2018.3, DeePhi for 2018.3)
  - C/C++ callable amalgamation of computer vision and machine learning
  - https://www.xilinx.com/products/design-tools/embedded-vision-zone.html

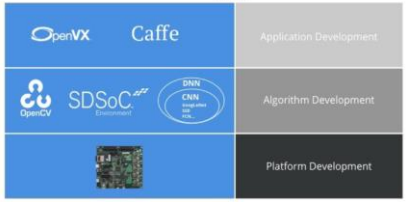


OpenCV

DEEPHi DNNDK

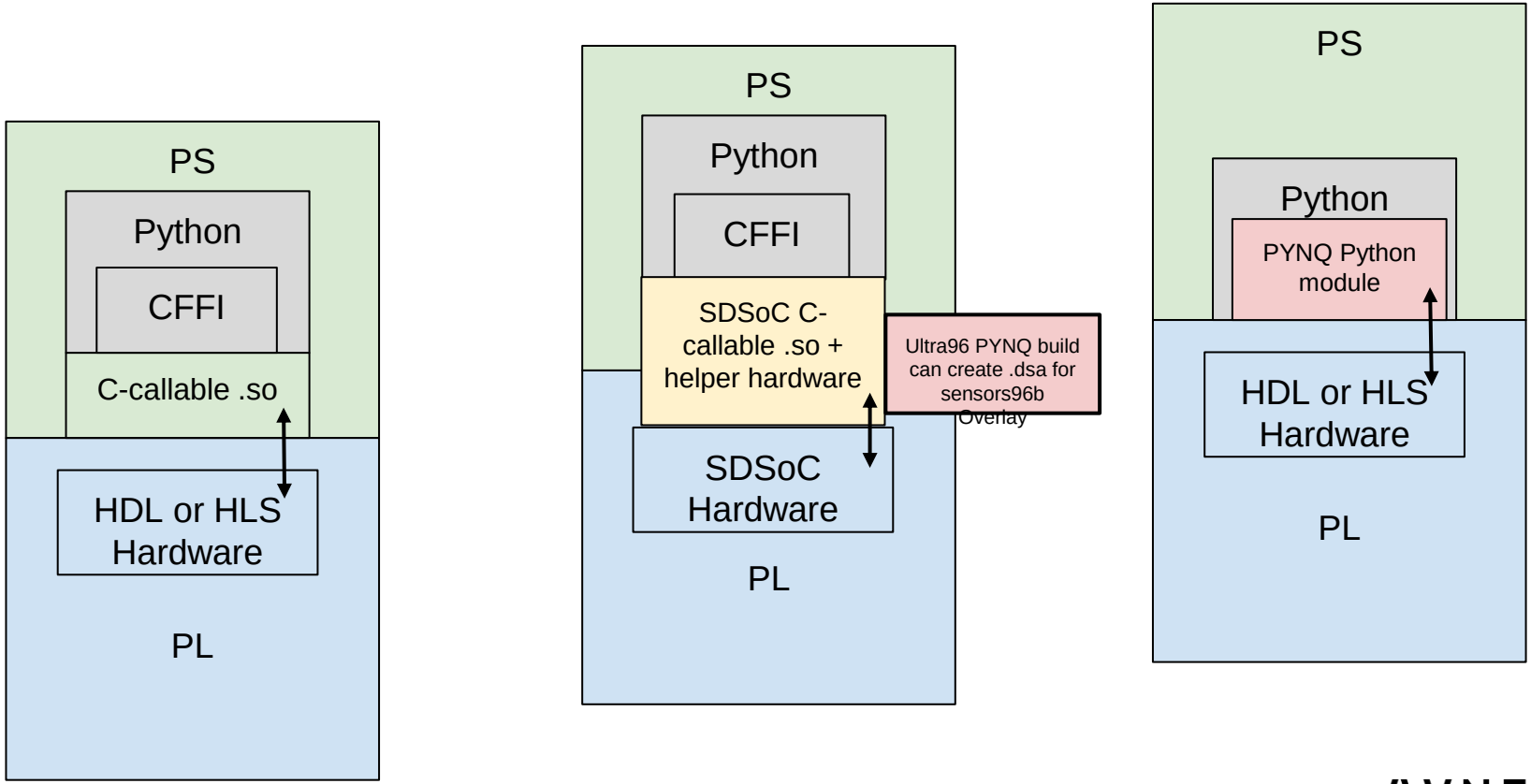
XILINX

reVISION Stack



AVNET

Python data exchange and control of the PL



Python CFFI Example

- Common Foreign Function Interface is a simple and popular means to interface Python with C code
- A c-like method to execute and exchange data with 2 languages whose types are not the same

Simple example:

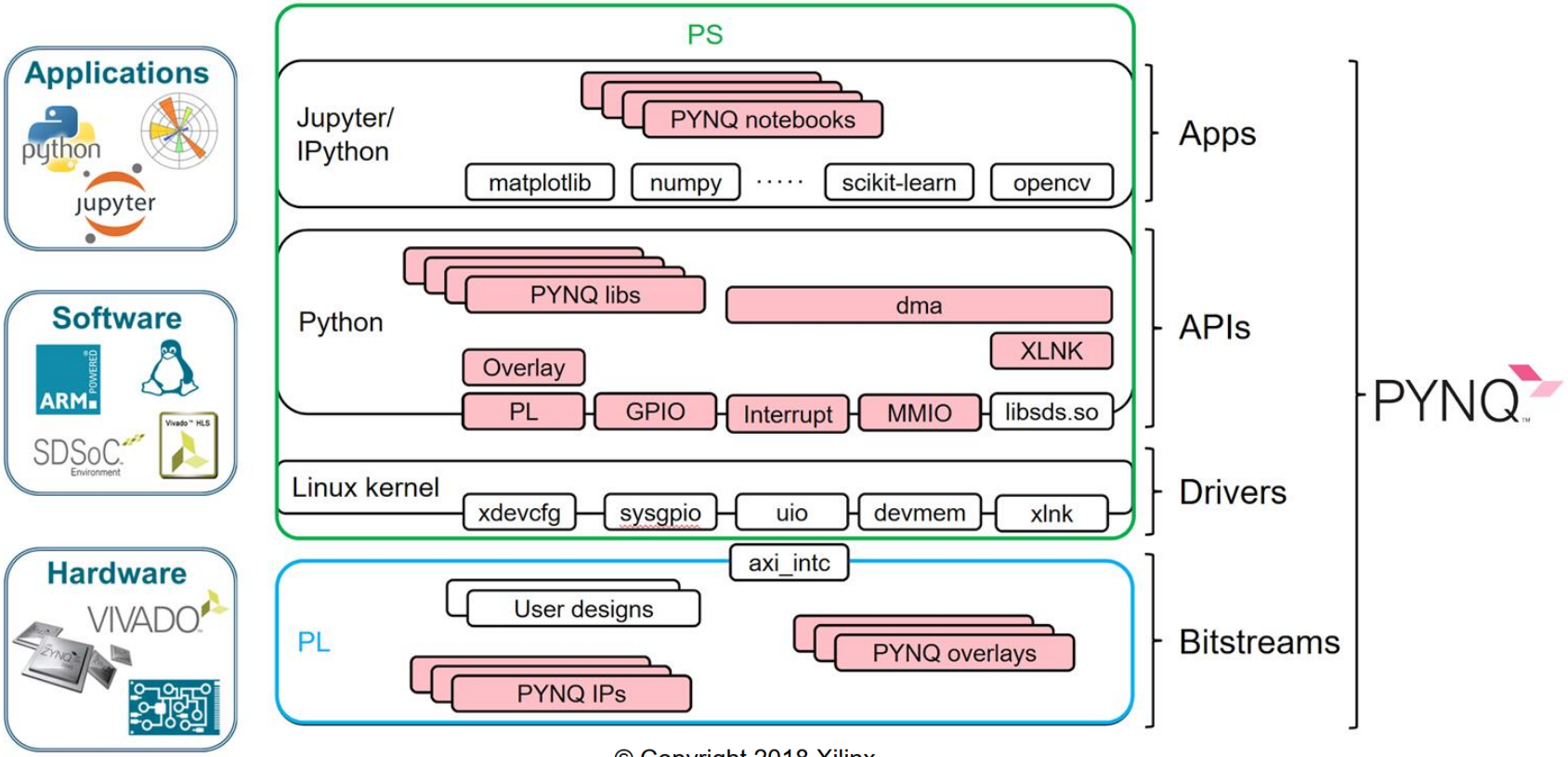
```
from cffi import FFI
ffi = FFI()
ffi.cdef("""
   int main_like(int argv, char *argv[]);
""")
lib = ffi.dlopen("your_library.so")

lib.main_like(2, [ffi.new("char[]", "arg0"),
                  ffi.new("char[]", "arg1")])
```

More info:

https://cffi.readthedocs.io/en/latest/

AVNET

PYNQ Open Source framework



AVNET

Ultra96 PYNQ Python module features

Grove board and soon Mikro Elektronika sensor access

Control PL MicroBlaze™ and other IOPs

Classes for creating and auto-creation of Python hardware drivers

Memory management accelerations for numpy

Re-configure and load the PL (load Overlays)

Zero-copy DMA memory transfers

Python control of GPIO, Interrupts, Clocks, SPI, I2C and more

Event driven multi-threaded template classes

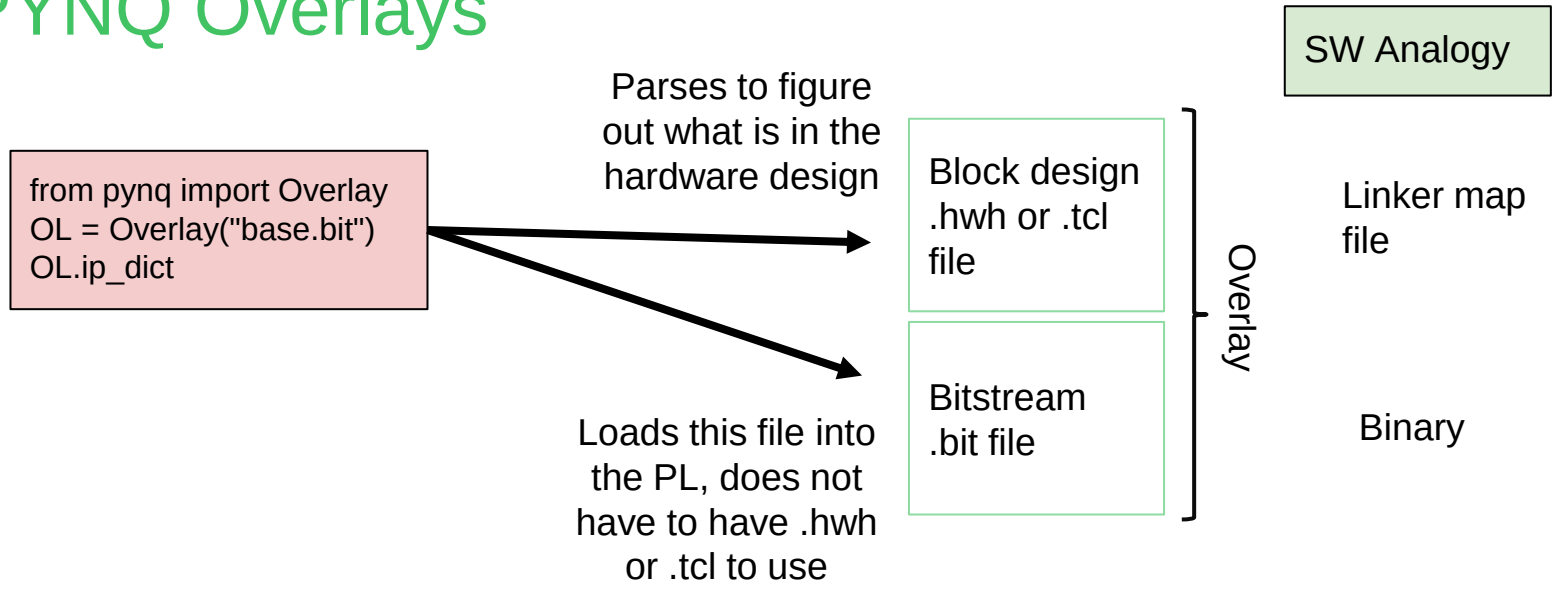
Additional IP and much much more...

For precise API details, examples and how-to, Please READ THE MANUAL:

https://pynq.readthedocs.io/en/v2.4/

AVNET

PYNQ Overlays



The Block design .tcl file is created by the Vivado hardware tools with “File > Export > Block Design”, .hwh is based on .dsa which are also created with the Vivado tool.

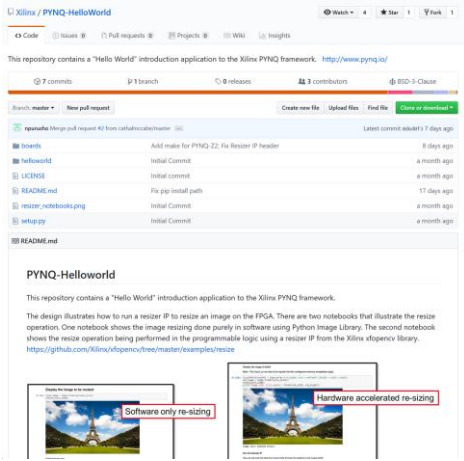
https://pynq.readthedocs.io/en/v1.3/10_creating_overlays.html

Definition of Bitstream file:
A bitstream is a file that contains the programming information for the PL. A Xilinx PL device must be programmed using a specific bitstream in order for it to behave as an embedded hardware platform.

AVNET

Working example: Ultra96 image resizer

- Python uses PYNQ library to interface Python to the PL Resizer Hardware
- Hardware was designed using C++ HLS and xfOpenCV for image processing
- 2 Notebooks: Resizer_PL.ipynb and Resizer_PS.ipynb
- Hardware resizer round-trip is about 4 times faster than SW only



Fetch Notebook from here:
sudo pip3 install git+https://github.com/Xilinx/PYNQ-HelloWorld.git

Includes HLS source code (Please look at this source):
https://github.com/Xilinx/PYNQ-HelloWorld/tree/master/boards/ip/hls

You can study this example and modify it to suit your own purpose.
You don't have to understand every detail to use it as a template for acceleration

AVNET

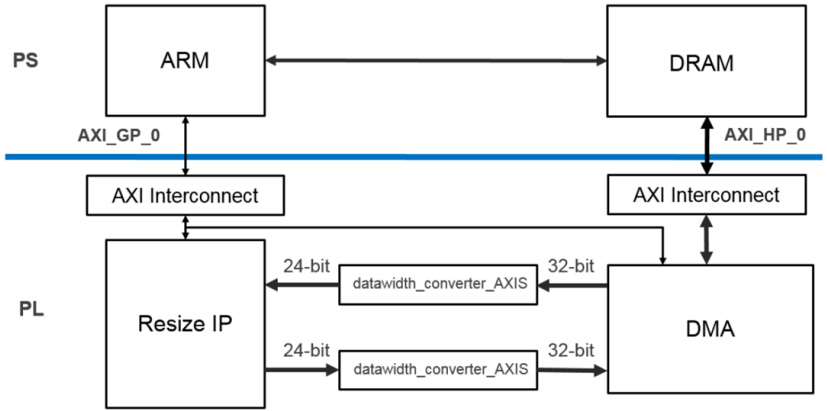
Resizer in Programmable Logic (PL) - Application Notebook

This reference design illustrates how to run a resizer IP on the Programmable Logic (PL) using Jupyter Notebooks and Python

> The resize IP transforms the source image to the size of the destination image. Different types of interpolation techniques can be used in resize function, namely: Nearest-neighbor, Bilinear, and Area interpolation. The type of interpolation can be passed as a parameter to the Python API.
> https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug1233-xilinx-opencv-user-guide.pdf

Contents

Block diagram



AVNET

Import libraries

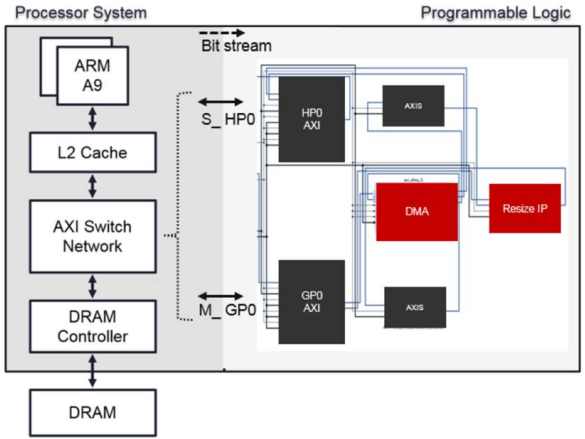
In [1]:
```
from PIL import Image
import numpy as np
from IPython.display import display
from pynq import Xlnk
from pynq import Overlay
```

Download the Resize IP bitstream

In [2]:
```
resize_design = Overlay("/usr/local/lib/python3.6/dist-packages/helloworld/bitstream/resizer_wrapper.bit")
```

Create DMA and Resizer IP objects

In [3]:
```
dma = resize_design.axi_dma_0
resizer = resize_design.resize_accel_0
```



Create an Image object using PIL in SW

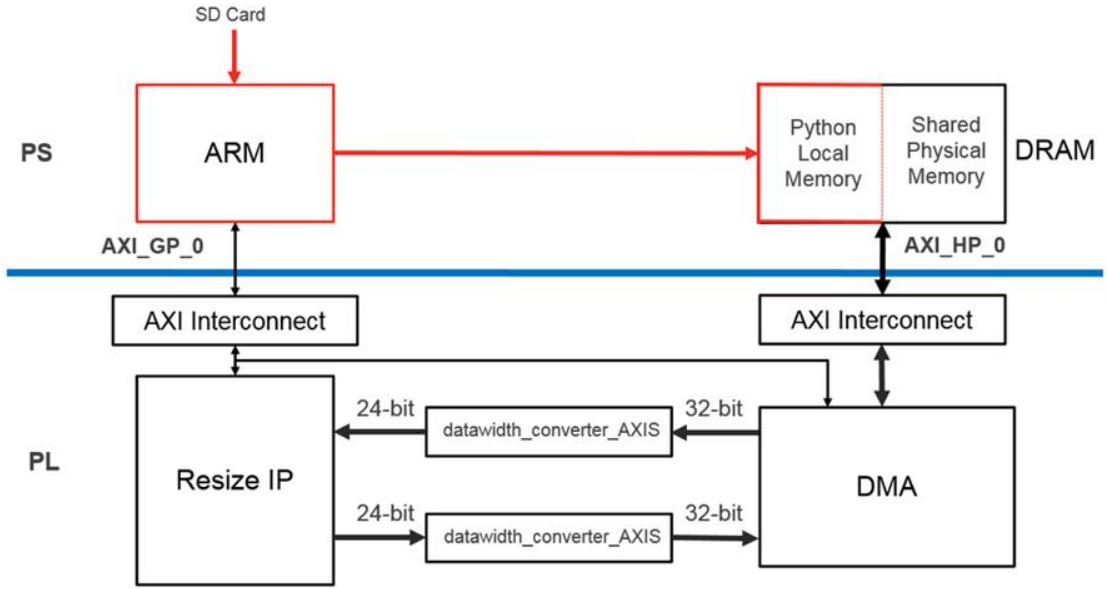
Load image from the SD card and create a PIL Image object

In [4]:
```
image_path = "./images/iceland.jpg"
original_image = Image.open(image_path)
original_image.load();
```

AVNET

Create a numpy array of the pixels

In [5]:
```
input_array = np.array(original_image)
```



AVNET

Display the image to be resized

In [6]:
```
input_image = Image.fromarray(input_array)
display(input_image)
```

Original image size

In [7]:
```
old_width, old_height = original_image.size
print("Image size: {}x{} pixels.".format(old_width, old_height))
```

```
Image size: 640x427 pixels.
```

AVNET

Resize in Programmable Logic

Set resize dimensions

Note:

1. Downscale factor range: 1 to 7 (by design of the resize IP)
2. Max input and output size of Image is 640x360 for this version of resize IP

In [8]:
```
resize_factor = 2
new_width, new_height = int(old_width/resize_factor), int(old_height/resize_factor)
```

Allocating memory to process data on PL

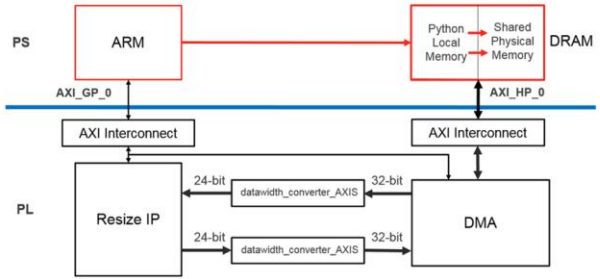
Data is provided as contiguous memory blocks.

The size of the buffer depends on the size of the input or output data.
The image dimensions extracted from the read image are used to allocate contiguous memory blocks.
We will call `cma_array()` to perform the allocation.

In [9]:
```
xlnk = Xlnk()
in_buffer = xlnk.cma_array(shape=(old_height, old_width, 3), dtype=np.uint8, cacheable=1)
out_buffer = xlnk.cma_array(shape=(new_height, new_width, 3), dtype=np.uint8, cacheable=1)
```

Note : Documentation snippet for **`xlnk.cma_array`** function:

```
xlnk.cma_array(shape, dtype=<class 'numpy.uint32'>)

Get a contiguously allocated numpy array

Parameters
----------
shape : int or tuple of int
    The dimensions of the array to construct - We use (height, width, depth)
dtype : numpy.dtype or str
    The data type to construct - We use 8-bit unsigned int
```



AVNET

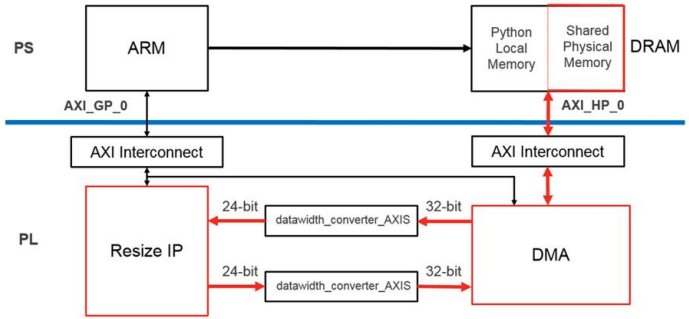
Display the image in buffer

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

In [10]:
```
in_buffer[0:819840] = 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))
```

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))
```

AVNET


```
Resized in Hardware(PL): 320x213 pixels.
```

Timing the resize in PL operation

In [12]:
```
# 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()
```

In [13]:
```
%%timeit
run_kernel()
```

```
100 loops, best of 3: 3.01 ms per loop
```

Reset XInk

In [14]:
```
xlnk.xlnk_reset()
```

AVNET

The End

Appendix / Extra info to help get started with Ultra96 PYNQ v2.4

Additional Information:

Main PYNQ site:

http://www.pynq.io

PYNQ distribution source code (build PYNQ for Ultra96 from scratch):

https://github.com/Xilinx/github & https://github.com/Avnet/Ultra96-PYNQ

You can find the base Overlay and a makefile to create an Ultra96 DSA for 2018.2 SDSoC under Avnet/Ultra96-PYNQ/sensors96b/

Additional design examples and tutorial videos:

http://www.pynq.io/community.html

Official Ultra96 PYNQ getting started guide:

https://ultra96-pynq.readthedocs.io/en/latest/

Getting started video:

https://www.youtube.com/watch?time_continue=3&v=is34FBOlDJE

Avnet Ultra96, PYNQ downloads and more:

http://zedboard.org/product/ultra96

White paper: “The Value of Python Productivity: Extreme Edge Analytics on Xilinx Zynq Portfolio

https://www.xilinx.com/support/documentation/white_papers/wp502-python.pdf

AVNET

HLS, SDSoC and general PL Intros

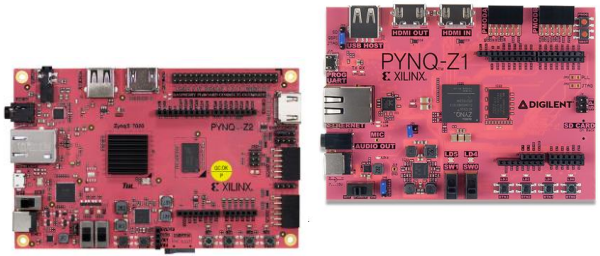
- Intro to HLS design: https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf
- Intro to SDSoC design: https://github.com/Xilinx/SDSoC-Tutorials/tree/master/getting-started-tutorial
- Avnet Ultra96 SDSoC Platform Tutorial v2018.2: http://zedboard.org/support/design/24166/156
- https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug902-vivado-high-level-synthesis.pdf
- https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0012-vivado-high-level-synthesis-hub.html
- Older but still useful for HLS: https://github.com/Xilinx/HLx_Examples

The growing PYNQ family

- Z1, Z2, ZCU104, Ultra96 v1, v2 and RFSoC ZCU111
- PYNQ can be ported to any Zynq or ZynqMP PetaLinux supported platform

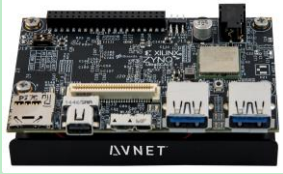
ZCU104 with ZU7EV

(Most powerful part with powerful video codec, also the most expensive, least amount of available public PYNQ designs)



PYNQ Z1,Z2 with XC7Z020

(Least inexpensive, has the most existing PYNQ designs but is least powerful of these)



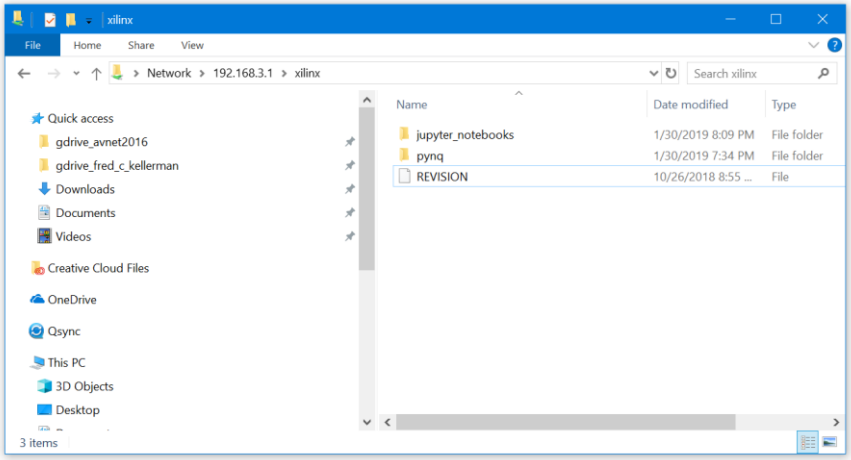
ULTRA96 with ZU3EG

AVNET

Drag and drop file transfer through USB:

- After boot up, wait about 15-20 seconds, a bit longer the 1st time (see Appendix for more info)
- Open Window's File Explorer (if your PC Admin allows share mapping) open and enter in search bar: \\192.168.3.1\xilinx
- On Linux you must install and configure Samba or cifs
- Simply drag and drop files to transfer them between the Ultra96 file system and the host PC

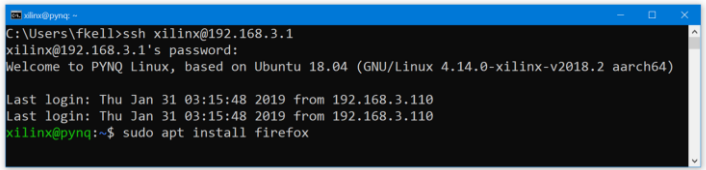
Password: xilinx



Debian Package Manager

- This PYNQ feature is **fundamentally** different than a typical PetaLinux distribution:
  - It enables installation of features and applications without re-building PetaLinux
- A few examples of 1000's of programs that can be installed:
  - Firefox, Chrome, Python Libraries, C/C++ Libraries, OpenGL, different graphical desktop utilities
- All package manager commands start with 'sudo apt', update, upgrade, install, remove
- How to use it see: https://wiki.debian.org/DebianPackageManagement

Internet
Server/Cloud



#!/bin/bash console ~$ through USB:

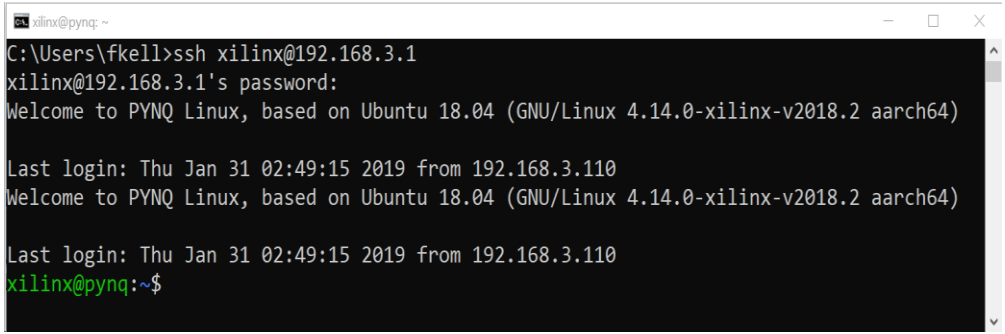
User: xilinx Password: xilinx

Serial and networking ssh connections, Windows, OSX or Linux

ssh xilinx@192.168.3.1

COMx :115200, 8,N,1

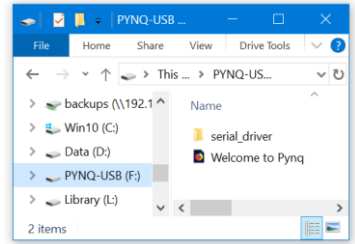
PuTTY
(for serial or SSH)



AVNET

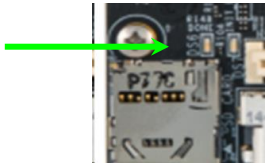
PYNQ booting troubleshooting:

- Wait, it takes about 20 seconds to start up but longer the 1st time, the host PC has to install some drivers for the USB functionality (RNDIS, Samba, Serial COM port)
- On Windows 7 the serial drivers may need to be installed from the PYNQ-USB drive “serial_driver” folder. Windows 10 built-in drivers worked out of the box for me
- You know it is working if a removable drive named “PYNQ-USB” mounts on your host OS, Jupyter Server takes a little longer to start up shortly after.



Troubleshooting:

- Make sure USB cable is working properly
- Make sure your OS is setup to allow removable drives to auto-mount (if not you’ll have to figure out how to mount it)
- If no lights on on the board: check power supply and re-press power button
- If **ds6** LED is not green: something is not right with the SD card image or the SD card is incompatible with the Ultra96 (Try again and/or a different card, make sure >= 16GB)



Connecting Ultra96 to the internet:

- For WiFi run Jupyter Notebook: /common/wifi.ipynb
- I prefer a plug-and-play USB to Ethernet adapter and wired LAN
  (Anker and TPLINK 1GB USB 3.0 have worked for me, others supported)
- Connections beyond USB IP are setup for automatic DHCP: board will have multiple IP addresses
- 192.168.3.1 is still maintained for USB connections
- Can use either IP address, can also disconnect USB and use just DHCP address
- It is not officially supported but if your PC has admin privileges you may be able to share PC internet access through just the Ultra96 USB cable:
    - Configure host PC appropriately
    - You will have to use dhclient on the Ultra96 to grab new IP.

Test Internet access:

- On the host PC use a serial comms program like TeraTerm and connect it to the PYNQ USB COM device
- Login user: xilinx, password: xilinx
- ping www.avnet.com
- You can obtain the local lan IP by finding the ip list with the ifconfig cmd
- Assuming external firewalls are not impeding your access you should now have full LAN and WAN networking access
- SSH, scp, sftp, ftp, http, pip3 and the apt Debian package manager can now be used to install additional software
- Viva la “apt update, upgrade, install” Enjoy!

AVNET

Acknowledgments:

Slide content marked with Xilinx Copyright were borrowed with permission from the Ultra96 PYNQ presentation given by Forrest Pickett at the San Jose October 2018 XDF conference

Hello world PYNQ resizer demo created by Xilinx and is open source

Overview graphic:
https://openclipart.org/detail/275842/sisyphus-overcoming-silhouette