XILINX
ALL PROGRAMMABLE™

PYNQ™ Python Productivity for Zynq

Patrick Lysaght, CTO, San Jose

XILINX
ALL PROGRAMMABLE™

Agenda

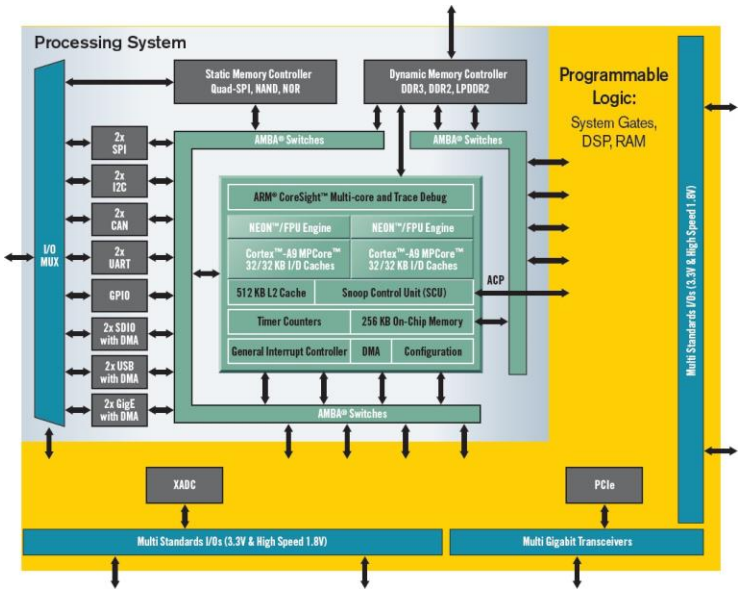
- Vision
- Inspiration: software
- Open source
- PYNQ™
- Base overlay
- PYNQ Community Projects
- *logictools* overlay
- Edge computing

Vision

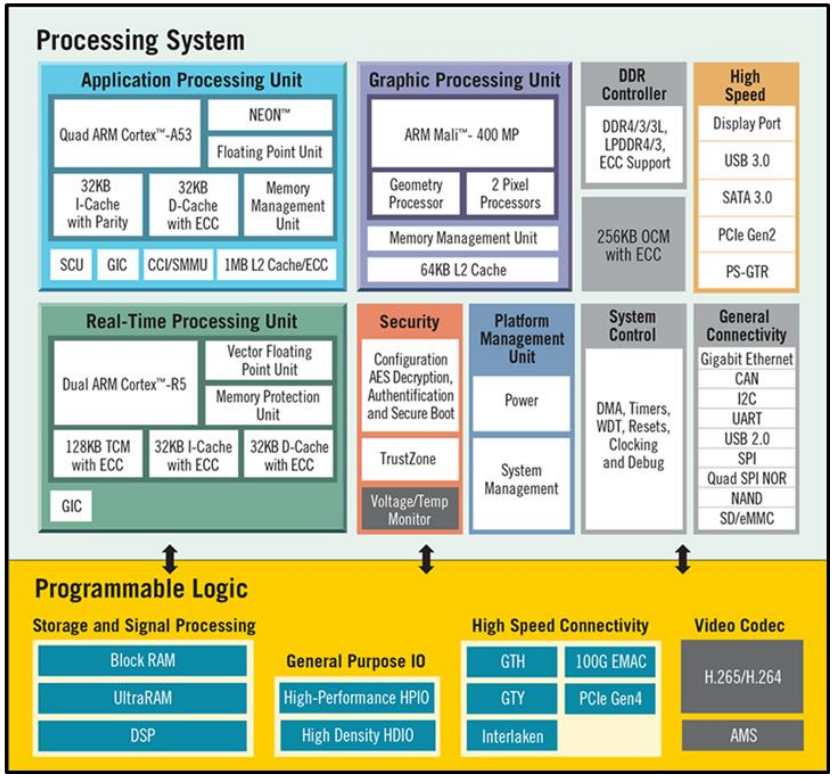
ZYNQ and ZYNQ UltraSCALE+

Best-in-class, All Programmable SoCs

ZYNQ 7000



ZYNQ UltraSCALE+

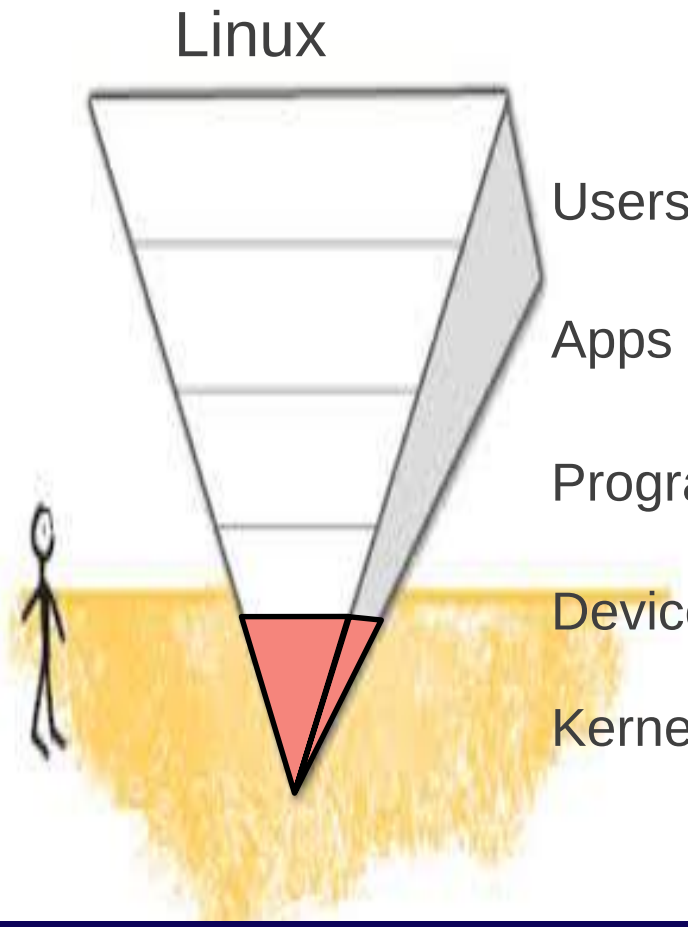


FPGAs and tightly-integrated CPUs enable entirely new opportunities

Make All Programmable technologies so easy-to-use that programmers can access the benefits of Zynq APSoCs without learning advanced digital design skills

Inspiration from the software community

Hundreds of people maintain the Linux kernel, tens of millions of people use it!



Users:	Tens of millions
Apps Programmers:	Hundreds of thousands
Programmers(embedded):	Tens of thousands
Device driver writers:	Thousands
Kernel developers:	Hundreds

How can we support hundreds of thousands of Zynq programmers with hundreds of bitstream developers?

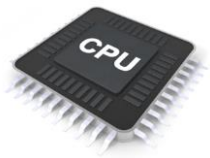
The rise of productivity languages

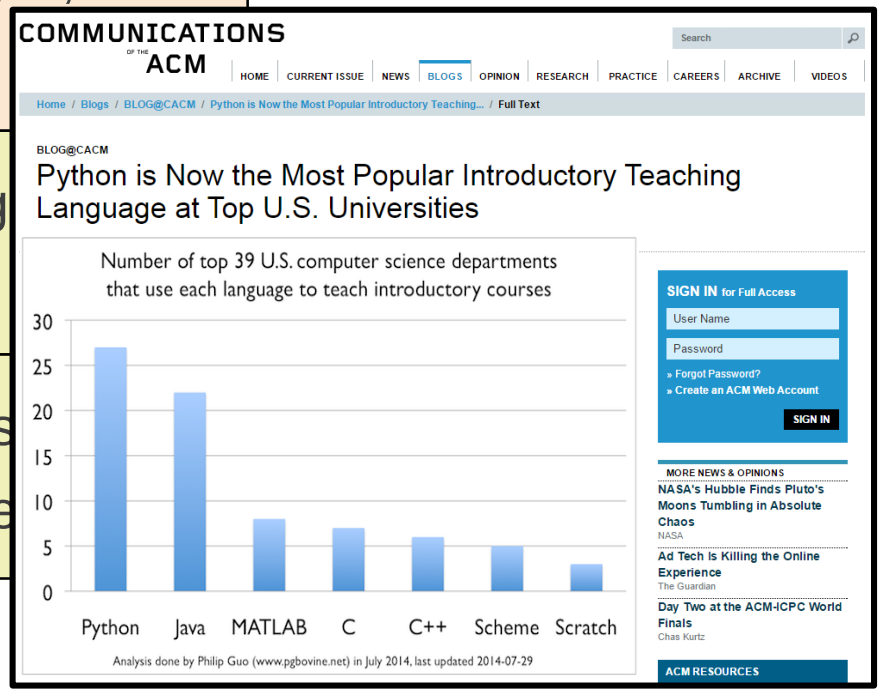
Applications,
Programming Frameworks

Productivity Languages,
Python, Scala, ..

Systems/Efficiency Lang
C, C++, OpenCL

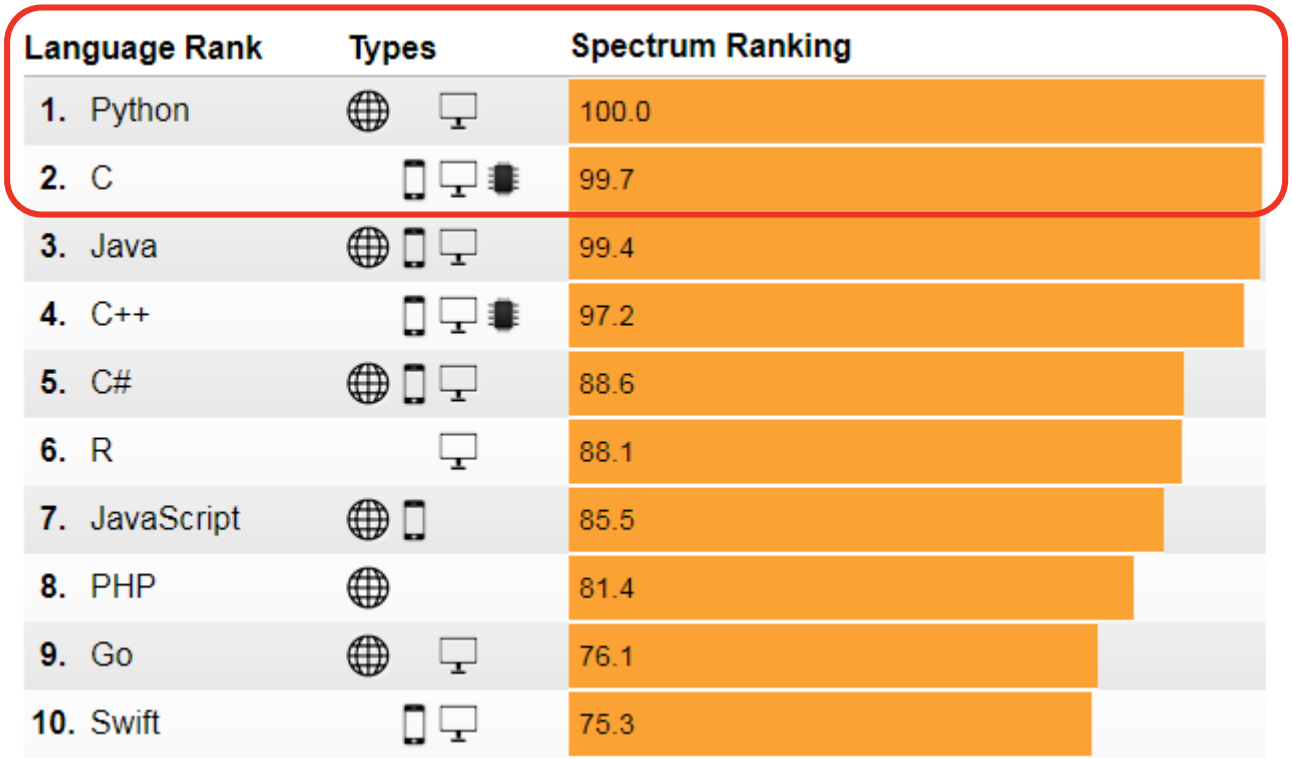
Hardware Systems
OS, hypervisor, drive





How can we program ZYNQ in a productivity language?

IEEE Spectrum's Top Programming Languages, July 2017

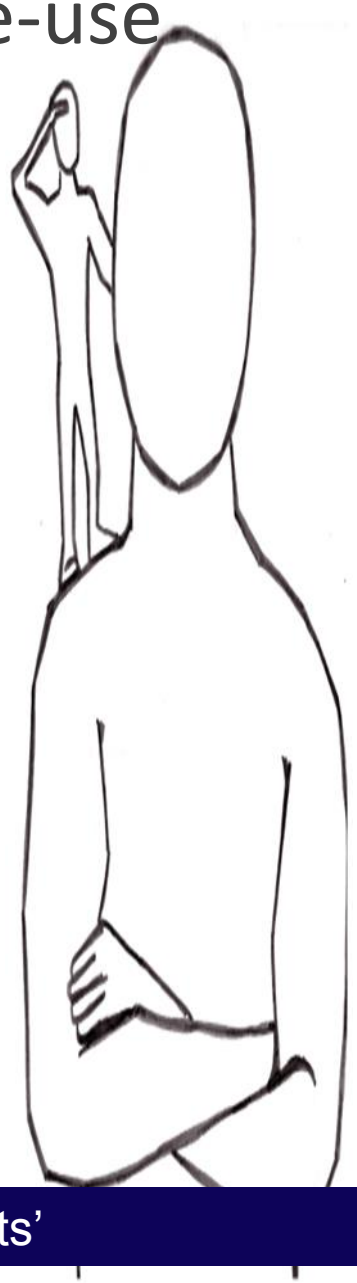


Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C		99.7
3. Java		99.4
4. C++		97.2
5. C#		88.6
6. R		88.1
7. JavaScript		85.5
8. PHP		81.4
9. Go		76.1
10. Swift		75.3

CPython gives us the best of Python and C

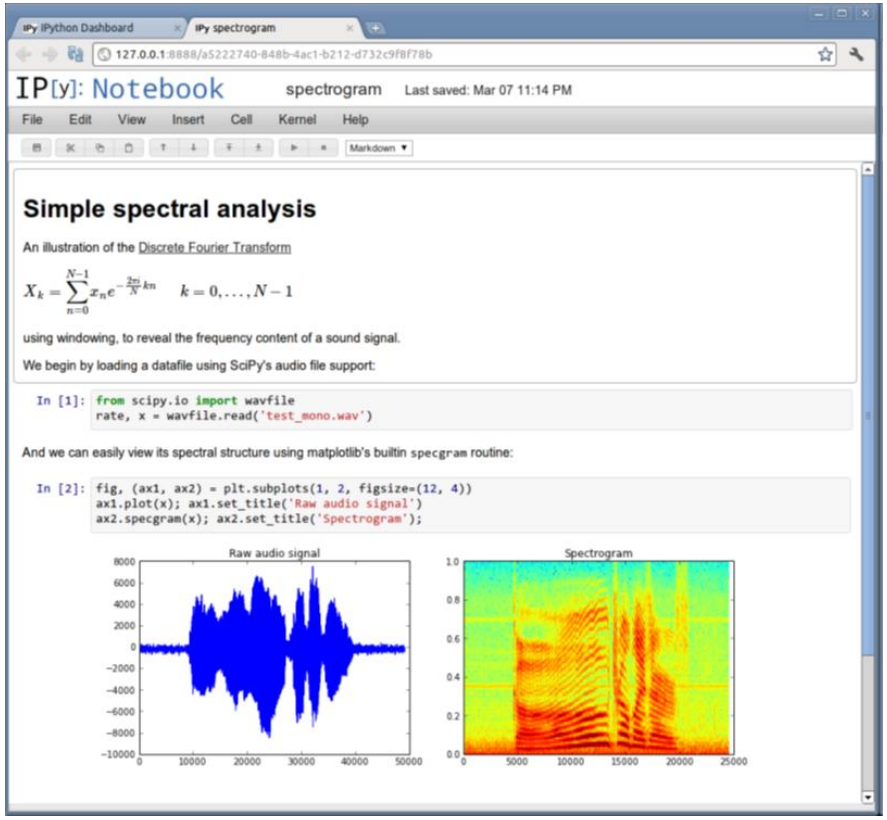
The rise of open source and knowledge re-use

- **GitHub: world's leading software repository**
  - https://github.com/
- **Stack Overflow: communities helping each other**
  - http://stackoverflow.com/
- **Python Package Index (PyPI): 113,000+ packages**
  - https://pypi.python.org
- **JupyterLab & Jupyter Notebook: web-based IDEs**
  - http://jupyter.org/
- **Readthedocs: software documentation repository**
  - https://readthedocs.org
- **Plus Linux, and Python itself, of course!**



We see further when 'we stand on the shoulders of giants'

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

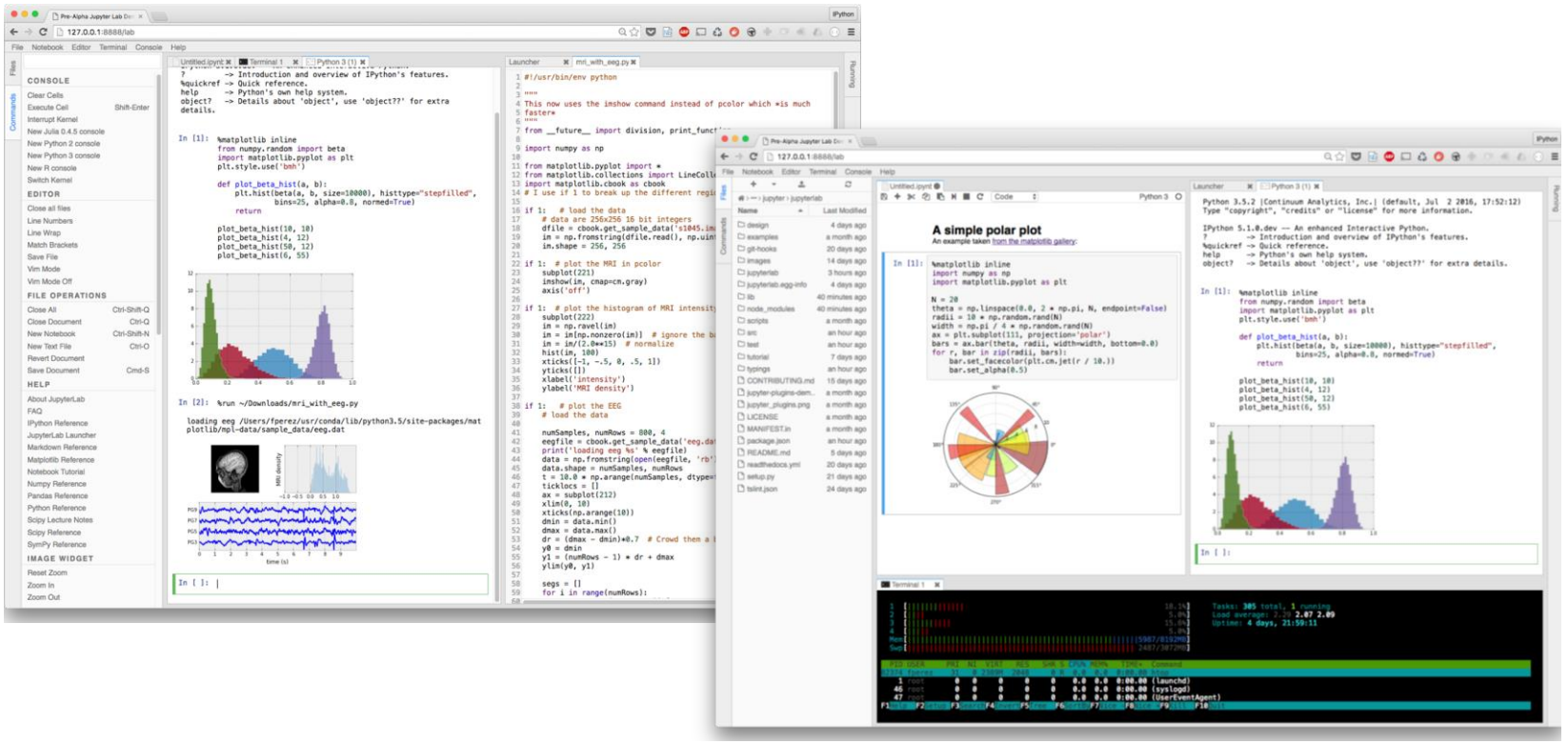


- Designed for
  - Interactive, exploratory computing
  - Reproducible results
- Ideal for
  - Teaching and learning
  - Projects and research
- Rapid adoption
  - Across multiple disciplines

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

Part of a broader trend towards new, web-based IDEs

JupyterLab: web-based IDE incl. Notebooks



Jupyter Notebook is now one of many plug-ins within the JupyterLab integrated development environment

JupyterLab - an open-source, extensible IDE in a browser

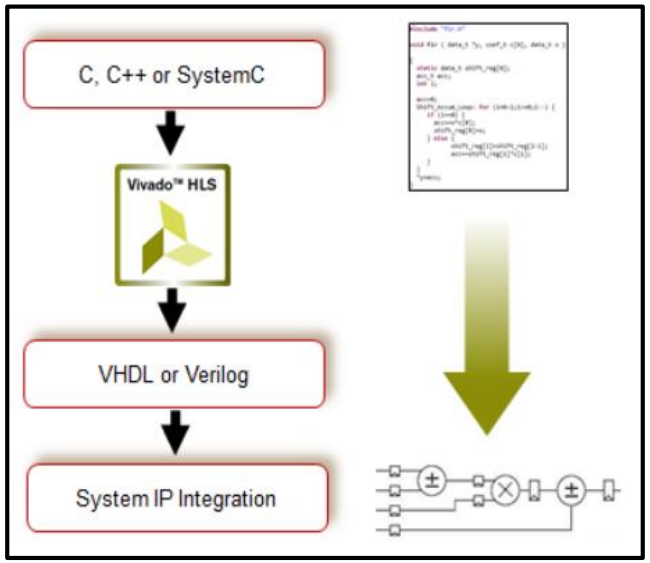
Python Productivity for Zynq

Today Zynq users use systems/efficiency languages

Could Zynq flows exploit productivity languages?

Productivity languages

Application oriented



Cross-compiled C/C++ code

FreeRTOS/Linux

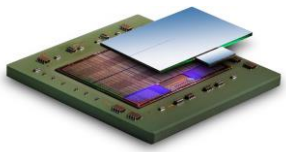
Bare metal IP drivers in C

Zynq PS & PL Hardware

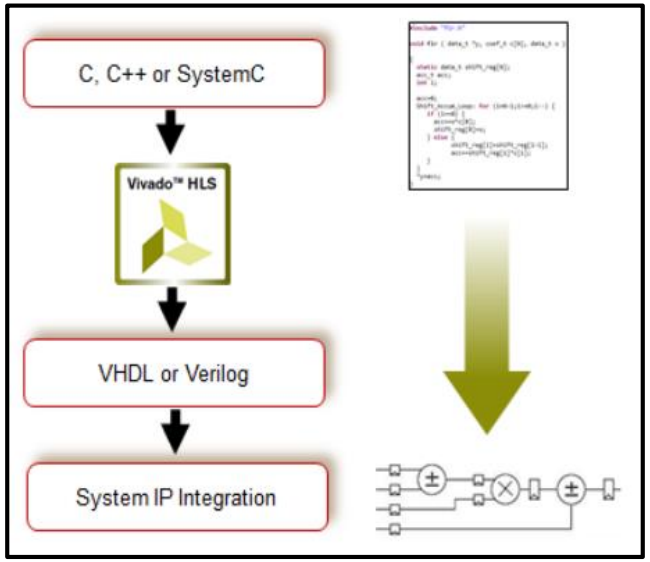
Systems/ efficiency languages

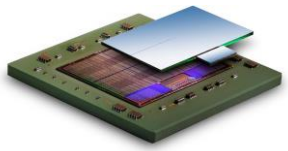
H/W flow

S/W flow



Productivity level tools for Zynq





Python scripts & programs

Interactive console | Python Libraries

CPython Virtual Machine | Python-C bindings

Ubuntu core | for IP drivers

IP drivers in C

Post-bitstream-programmable overlay

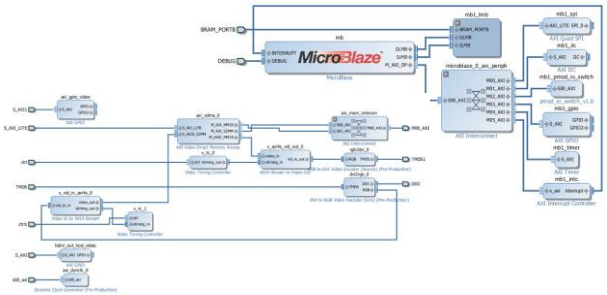
Overlays are analogous to ‘design patterns’

- Idiom
  - how we write code in a particular language to do this particular type of thing, eg the way that you code the process of stepping through an array in C
- Specific Design
  - the solution that we came up with to solve this particular problem. This might be a clever design, but it makes no attempt to be general
- Standard Design
  - a way to solve this kind of problem. A design that has become more general, typically through reuse
- Design Pattern
  - how to solve an entire class of similar problems. This usually only appears after applying a standard design a number of times, and then seeing a common pattern throughout these applications

Overlays are special FPGA designs – they each solve a class of problems

Modified from http://docs.linuxtone.org/ebooks/Python/Thinking_In_Python.pdf

Overlays aka hardware libraries



Step 1:

Create an FPGA design for a class of related applications

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

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

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

Step 2:

Export the bitstream and a C API for programming the design

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

void setInverseDisplay(){
    sendCommand(OLED_Inverse_Display_Cmd);
}

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

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

Step 3:

Wrap the C API to create a Python library

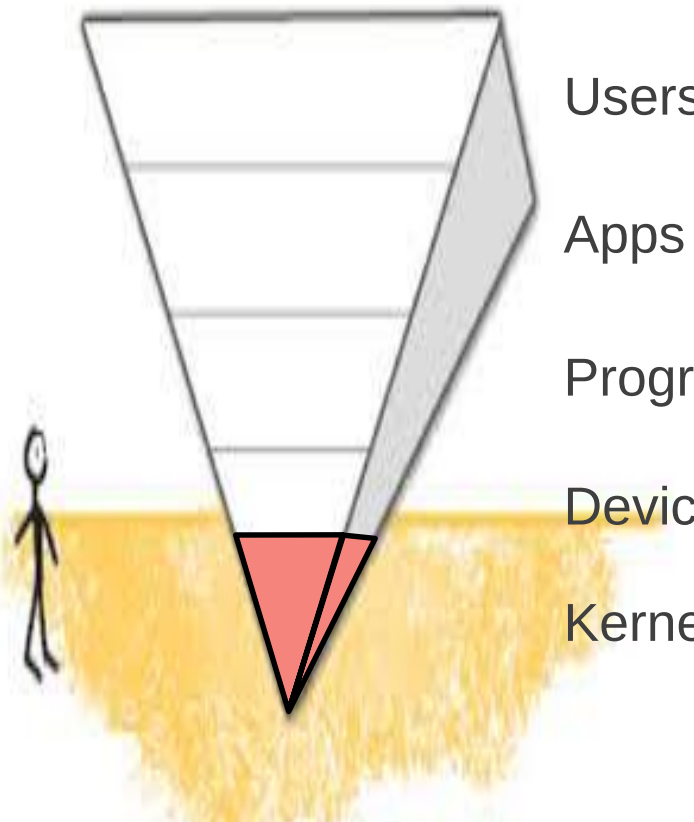
```
from time import sleep
from pynq import Overlay
from pynq.iop import PMOD_ADC, PMOD_DAC

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

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

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

Step 4:

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

Productivity languages & hardware overlays

All Programmable Devices



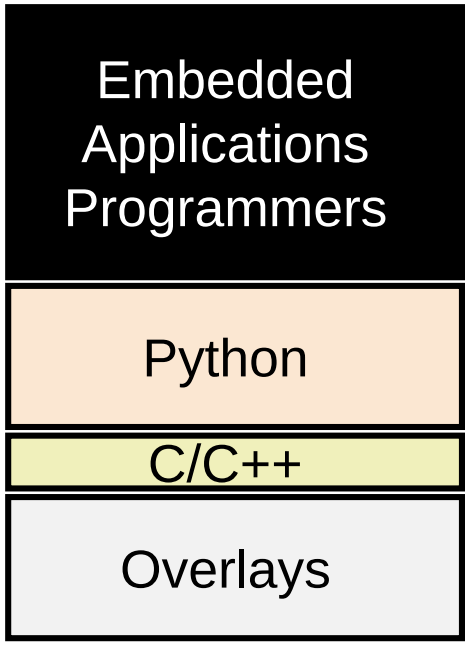
Users:

Apps Programmers:

Programmers(embedded):

Device driver writers:

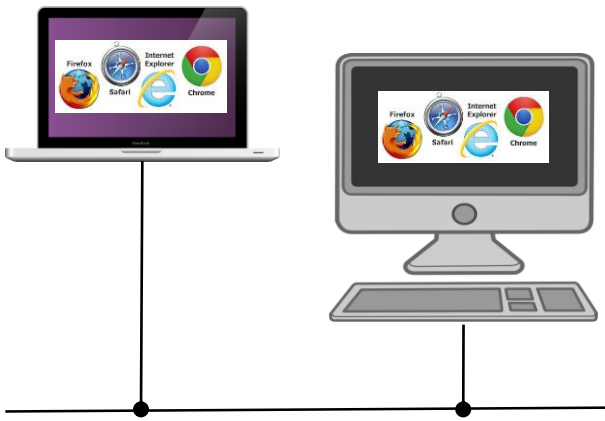
Kernel developers:



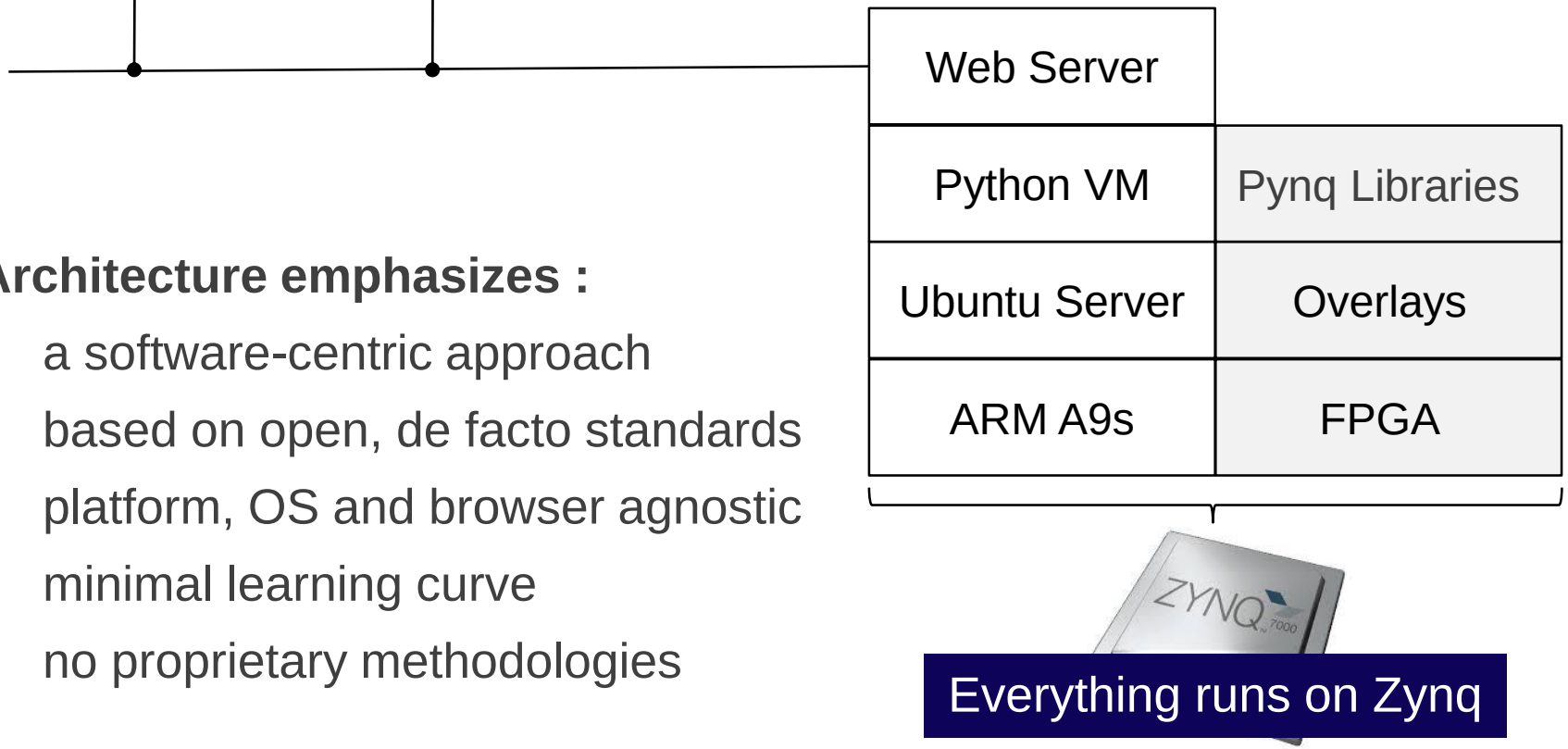
Small group of experts create APSoC overlays and C API/drivers
Many more users build applications in C/Python

Jupyter Notebooks on Zynq

Web & Python-based, open source architecture



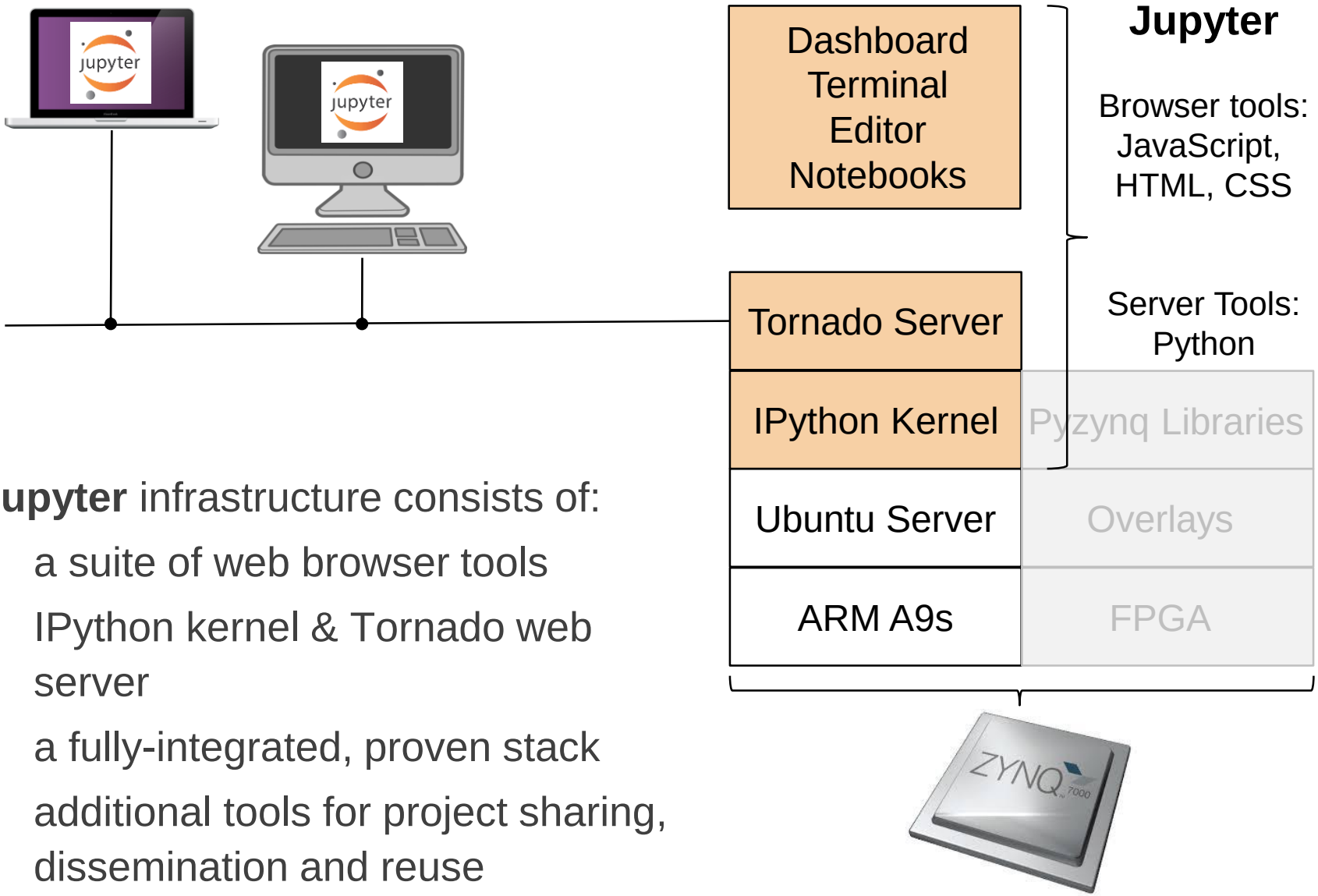
Develop on the target, access via the browser



Architecture emphasizes :

- a software-centric approach
- based on open, de facto standards
- platform, OS and browser agnostic
- minimal learning curve
- no proprietary methodologies

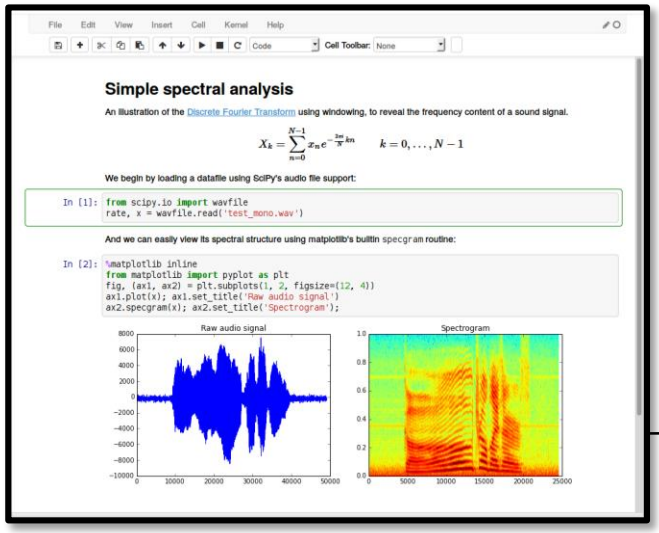
Jupyter on Zynq

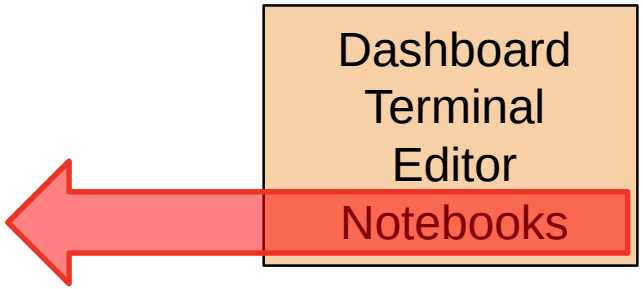


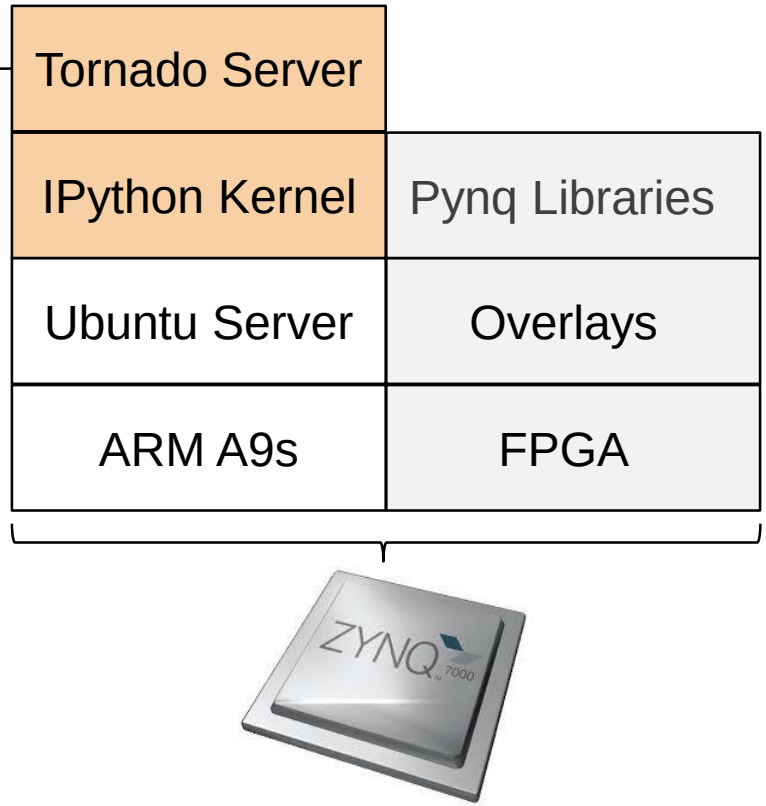
Jupyter infrastructure consists of:

- a suite of web browser tools
- IPython kernel & Tornado web server
- a fully-integrated, proven stack
- additional tools for project sharing, dissemination and reuse

Jupyter on Zynq







Jupyter Notebooks

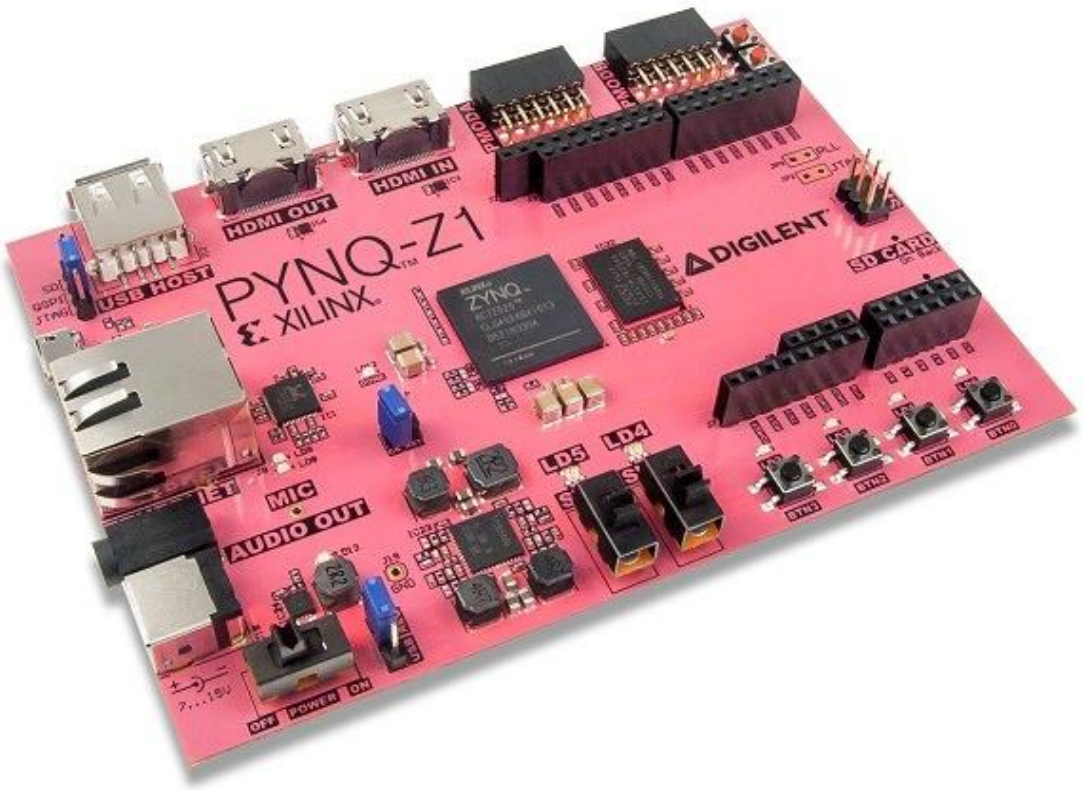
Interactive design with Zynq:

- highly productive, easy-to-use tools
- extensive third-party libraries
- integrated multi-media
- open source JSON format
- application-oriented perspective

With **PYNQ** embedded programmers **get** …

- **Productivity-level programming**
  - Interactive console & scripting
  - Fast compile and debug times
- **Hardware performance in a software environment**
  - Hardware parallelism
  - Function independence
  - Deterministic performance
  - Hard, real-time latencies
  - Accelerated algorithms
- **Portable libraries & open source community**

Low-cost PYNQ-Z1 Board

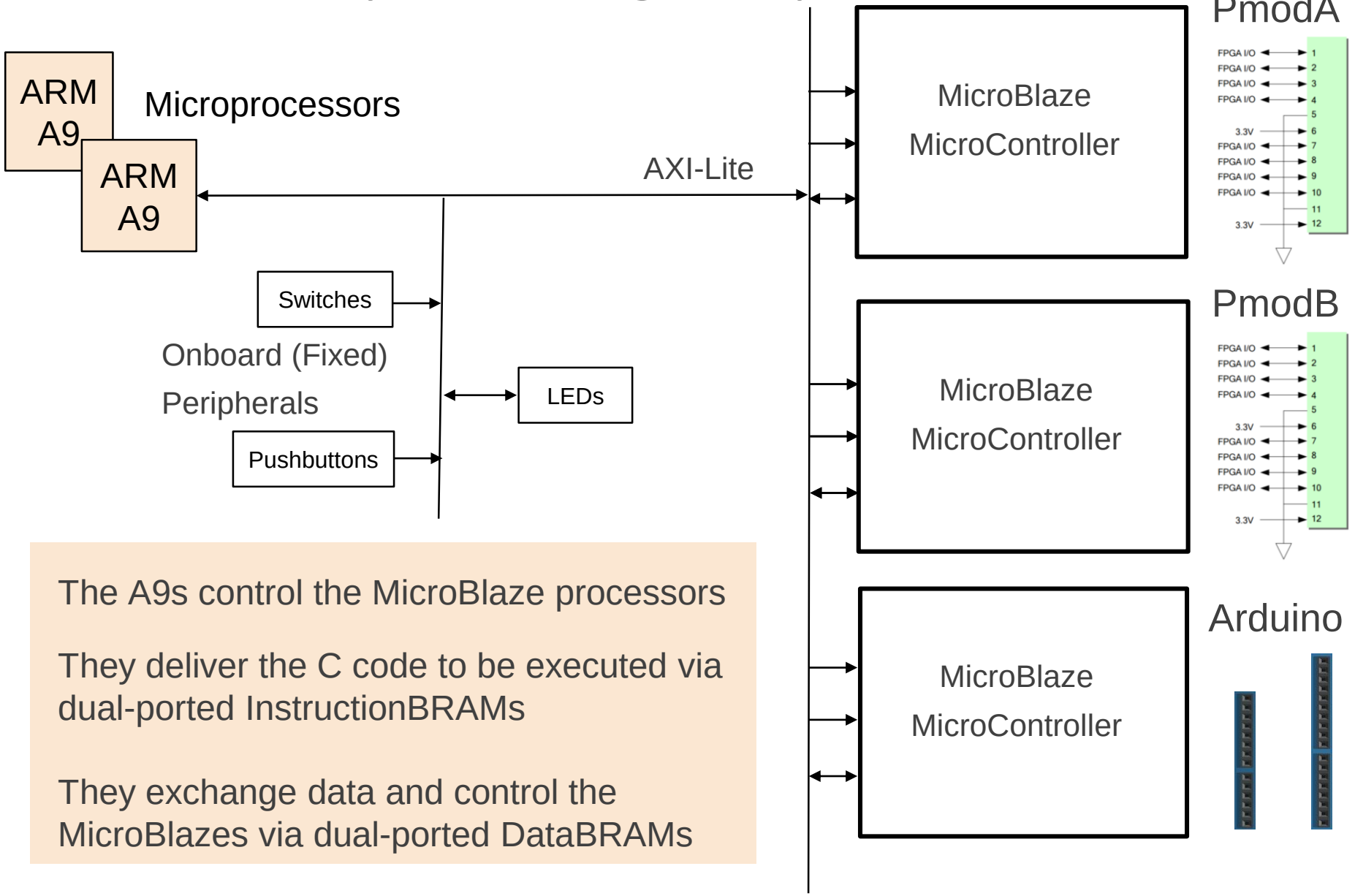


- Zynq XC7Z020 CLG400
- 512 MB DRAM
- HDMI input, HDMI out
- Microphone input
- Audio output
- 10/100/1G Ethernet
- MicroSD slot
- USB 2.0
- 4 LEDs, 4 buttons, 2 switches
- Arduino connector
  - With additional IO
- 2 Pmod 8-pin GPIOs

Available from Digilent for 65$ for approved academic customers

‘base’ overlay design and operation

`base` overlay block diagram (partial)



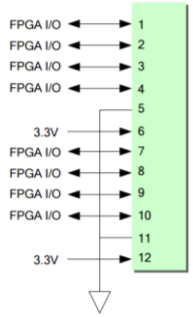
The A9s control the MicroBlaze processors

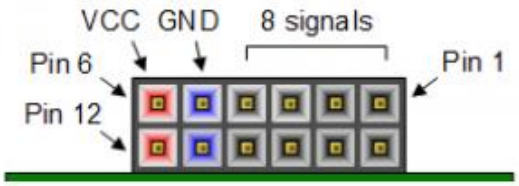
They deliver the C code to be executed via dual-ported InstructionBRAMs

They exchange data and control the MicroBlazes via dual-ported DataBRAMs

Typical Pmod IO physical and electrical interfaces

Pmod





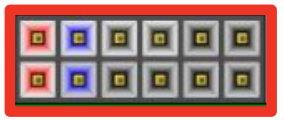
Pin	Signal	Description
1 & 5	SCL	Serial Clock
2 & 6	SDA	Serial Data
3 & 7	GND	Power Supply Ground
4 & 8	VCC	Power Supply (3.3V/5V)

Connector J1		
Pin	**Signal**	**Description**
1	CS	SPI Chip Select (Slave Select)
2	SDIN	SPI Data In (MOSI)
3	None	Unused Pin
4	SCLK	SPI Clock
7	D/C	Data/Command Control
8	RES	Power Reset
9	VBATC	V_{BAT} Battery Voltage Control
10	VDDC	V_{DD} Logic Voltage Control
5, 11	GND	Power Supply Ground
6, 12	VCC	Power Supply

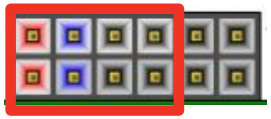
Pin	Signal	Description
1	~CS	Chip Select
2	MOSI	Master-Out-Slave-In
3	(NC)	Not Connected
4	SCLK	Serial Clock
5	GND	Power Supply Ground
6	VCC	Power Supply (3.3V/5V)

ADC

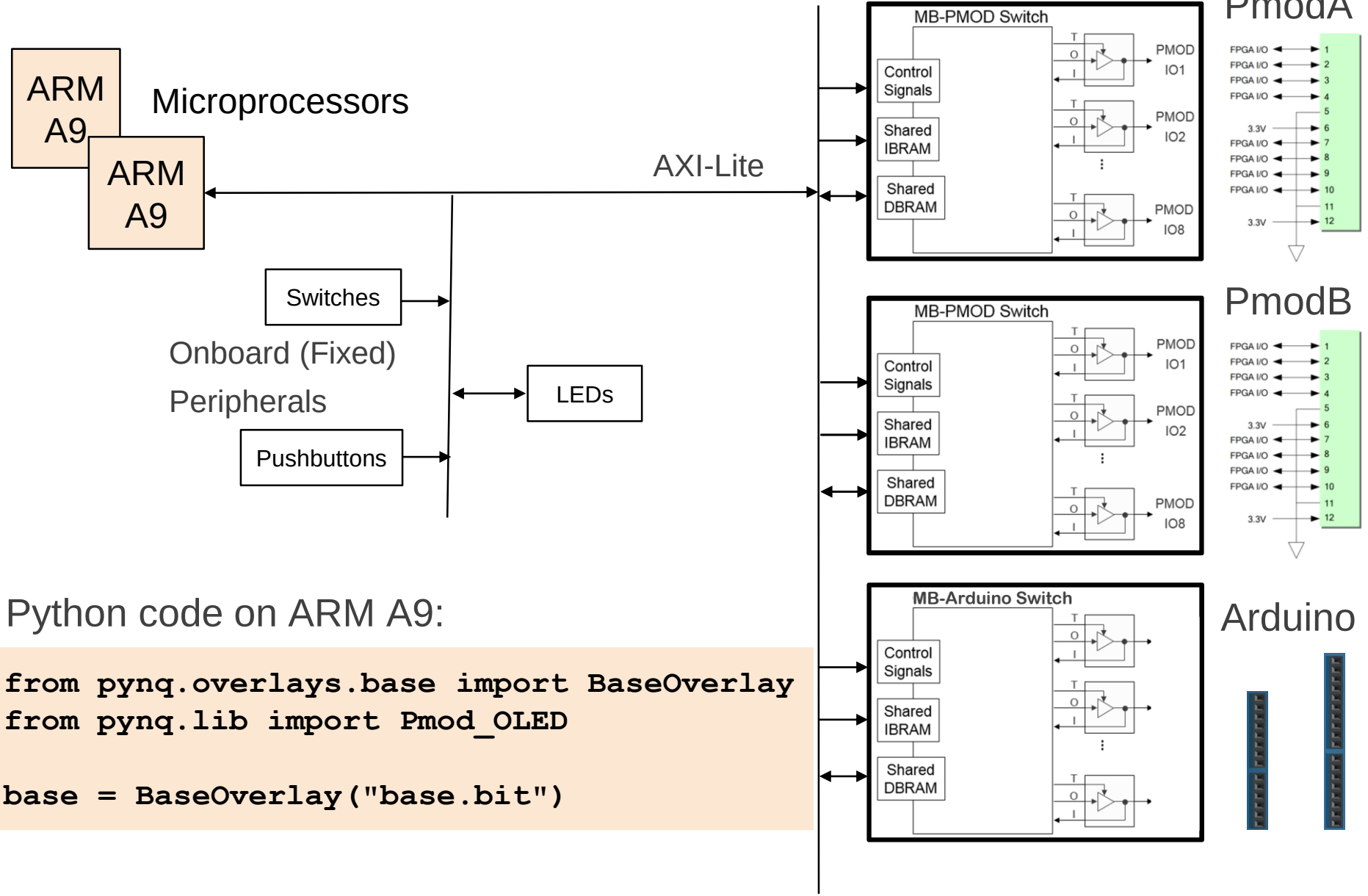
OLED



DAC



Load ‘base’overlay on PL

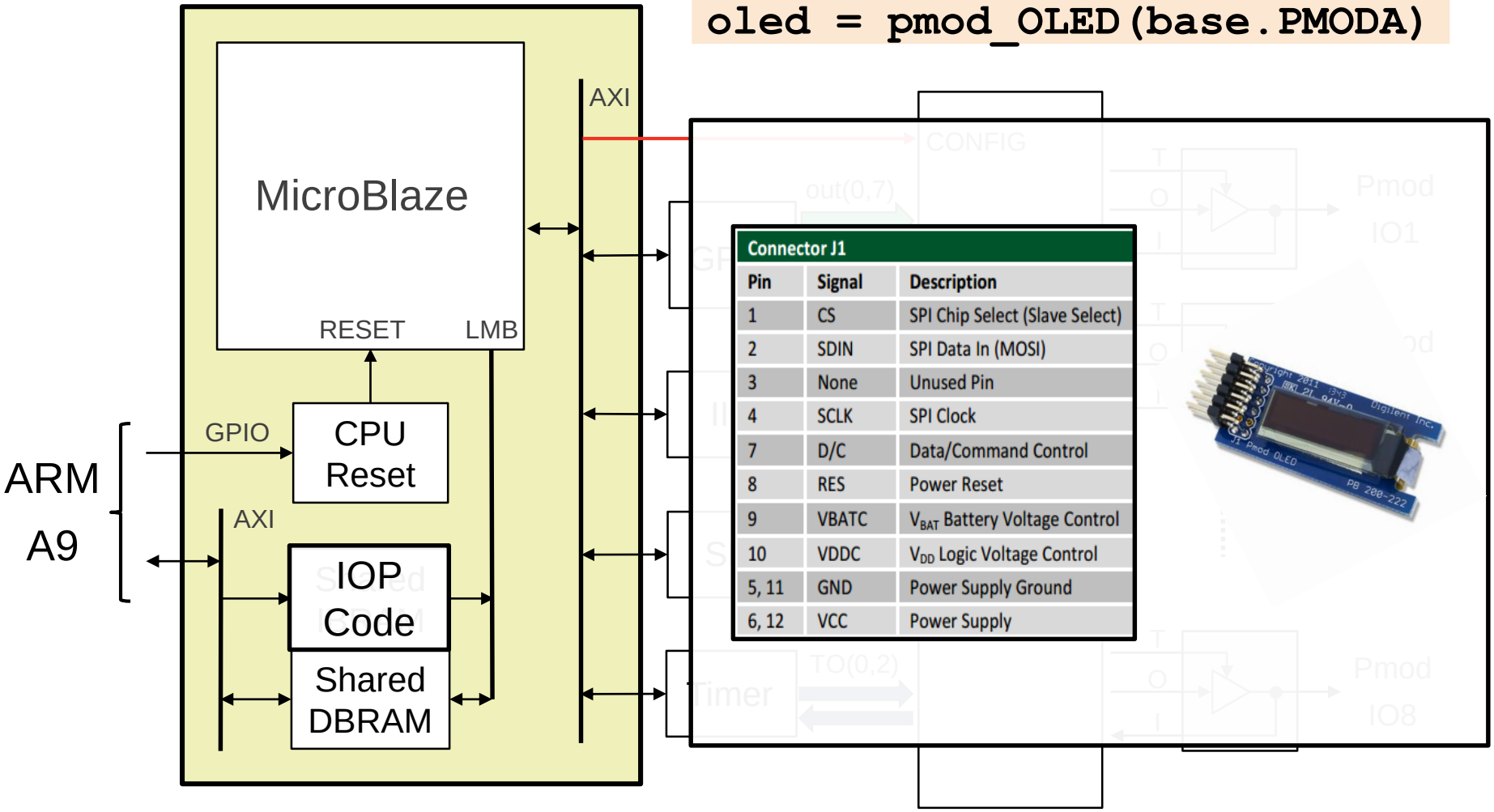


Python code on ARM A9:

```
from pynq.overlays.base import BaseOverlay
from pynq.lib import Pmod_OLED

base = BaseOverlay("base.bit")
```

Configure IO Processor for PmodA

```
oled = pmod_OLED(base.PMODA)
```



Connector J1		
Pin	Signal	Description
1	CS	SPI Chip Select (Slave Select)
2	SDIN	SPI Data In (MOSI)
3	None	Unused Pin
4	SCLK	SPI Clock
7	D/C	Data/Command Control
8	RES	Power Reset
9	VBATC	V_{BAT} Battery Voltage Control
10	VDDC	V_{DD} Logic Voltage Control
5, 11	GND	Power Supply Ground
6, 12	VCC	Power Supply


```
oled.write("1 2 3 4 5 6")
```

Jupyter Notebook

In [1]:

```
1 from pynq.overlays.base import BaseOverlay
2 from pynq.lib import Pmod_OLED
3
4
5 base = BaseOverlay("base.bit")
6 oled = Pmod_OLED(base.PMODA)
7 oled.write('1 2 3 4 5 6')
```

A total of 5 lines of user code

… thanks to Python, FPGA overlays, abstraction & re-use

Community Projects

Machine Learning on PYNQ cluster

Spark acceleration on FPGAs: A use case on machine learning in Pynq

Elias Koromilas, Ioannis Stamelos
Department of Electrical and Computer Engineering,
National Technical University of Athens
Athens, Greece

Christoforos Kachris
Institute of Communication and Computer Systems (ICCS/NTUA)
Athens, Greece

Dimitrios Soudris
Department of Electrical and Computer Engineering,
National Technical University of Athens
Athens, Greece

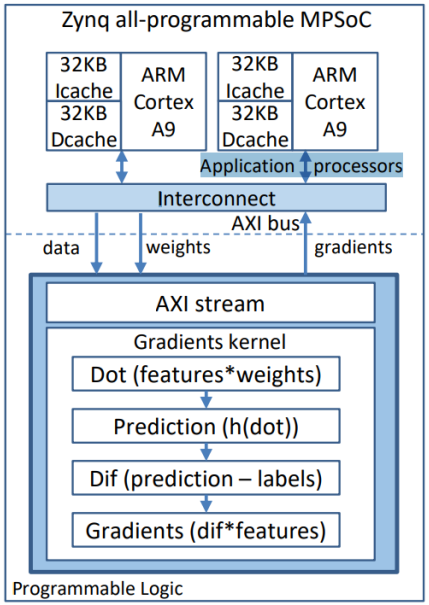
***Abstract*—Spark is one of the most widely used frameworks for data analytics. Spark allows fast development for several applications like machine learning, graph computations, etc. In this paper, we present Spynq: A framework for the efficient deployment of data analytics on embedded systems that are based on the heterogeneous MPSoC FPGA called Pynq. The mapping of Spark on Pynq allows that fast deployment of embedded and cyber-physical systems that are used in edge and fog computing. The proposed platform is evaluated in a typical machine learning application based on logistic regression. The performance evaluation shows that the heterogeneous FPGA-based MPSoC can achieve up to 11x speedup compared to the execution time in the ARM cores and can reduce significantly the development time of embedded and cyber-physical systems on Spark applications.**

powered down in order to comply with thermal constraints [3]. One way to address this problem is through the utilization of hardware accelerators. Hardware accelerators can be used to offload the processor, increase the total throughput and reduce the energy consumption.

In this paper we present a framework for the seamlessly utilization of hardware accelerators in heterogeneous SoCs that are used to speedup the processing of Spark data analytics applications.

The main contributions of this paper are the followings:

- An efficient framework for the seamlessly utilization of hardware accelerators for Spark applications



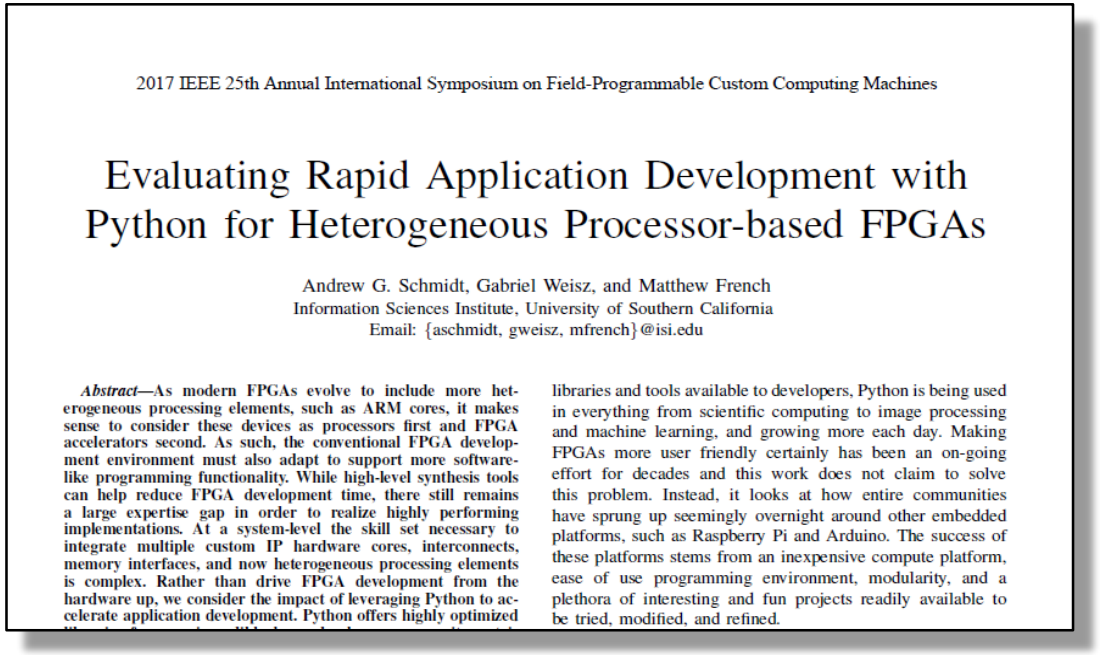
Best student paper award, MODCAST 2017

National Technical University of Athens

SPYNQ: Spark PYNQ cluster

https://www.youtube.com/watch?v=ix1NI7yq8O4

Evaluating PYNQ for image processing

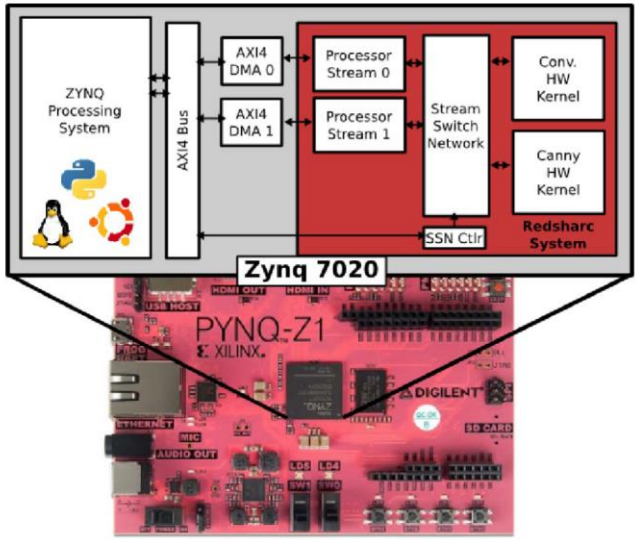


2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines

Evaluating Rapid Application Development with Python for Heterogeneous Processor-based FPGAs

Andrew G. Schmidt, Gabriel Weisz, and Matthew French
Information Sciences Institute, University of Southern California
Email: {aschmidt, gweisz, mfrench}@isi.edu

***Abstract*—As modern FPGAs evolve to include more heterogeneous processing elements, such as ARM cores, it makes sense to consider these devices as processors first and FPGA accelerators second. As such, the conventional FPGA development environment must also adapt to support more software-like programming functionality. While high-level synthesis tools can help reduce FPGA development time, there still remains a large expertise gap in order to realize highly performing implementations. At a system-level the skill set necessary to integrate multiple custom IP hardware cores, interconnects, memory interfaces, and now heterogeneous processing elements is complex. Rather than drive FPGA development from the hardware up, we consider the impact of leveraging Python to accelerate application development. Python offers highly optimized libraries and tools available to developers, Python is being used in everything from scientific computing to image processing and machine learning, and growing more each day. Making FPGAs more user friendly certainly has been an on-going effort for decades and this work does not claim to solve this problem. Instead, it looks at how entire communities have sprung up seemingly overnight around other embedded platforms, such as Raspberry Pi and Arduino. The success of these platforms stems from an inexpensive compute platform, ease of use programming environment, modularity, and a plethora of interesting and fun projects readily available to be tried, modified, and refined.**

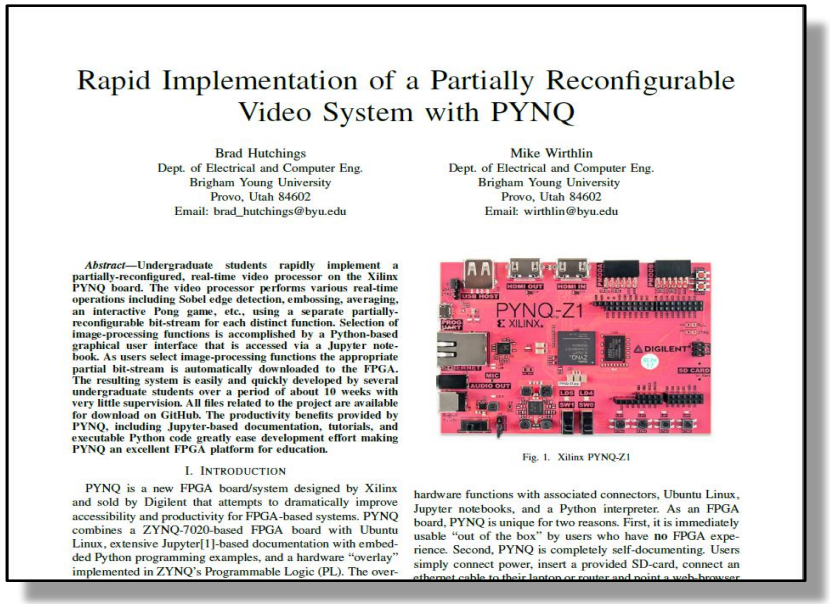


Best short paper award, FCCM 2017

USC Viterbi School of Engineering *Information Sciences Institute*

"The results are highly promising , with the ability to match and exceed performances from C implementations, up to 30x speedup."

“PYNQ is a game-changer”



Rapid Implementation of a Partially Reconfigurable Video System with PYNQ

Brad Hutchings
Dept. of Electrical and Computer Eng.
Brigham Young University
Provo, Utah 84602
Email: brad_hutchings@byu.edu

Mike Wirthlin
Dept. of Electrical and Computer Eng.
Brigham Young University
Provo, Utah 84602
Email: wirthlin@byu.edu

***Abstract*—Undergraduate students rapidly implement a partially-reconfigured, real-time video processor on the Xilinx PYNQ board. The video processor performs various real-time operations including Sobel edge detection, embossing, averaging, an interactive Pong game, etc., using a separate partially-reconfigurable bit-stream for each distinct function. Selection of image-processing functions is accomplished by a Python-based graphical user interface that is accessed via a Jupyter notebook. As users select image-processing functions the appropriate partial bit-stream is automatically downloaded to the FPGA. The resulting system is easily and quickly developed by several undergraduate students over a period of about 10 weeks with very little supervision. All files related to the project are available for download on GitHub. The productivity benefits provided by PYNQ, including Jupyter-based documentation, tutorials, and executable Python code greatly ease development effort making PYNQ an excellent FPGA platform for education.**

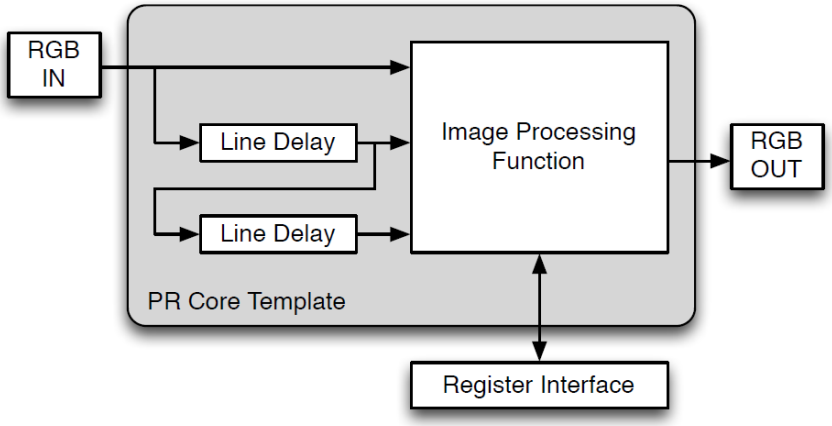
Fig. 1. Xilinx PYNQ-Z1

I. Introduction

PYNQ is a new FPGA board/system designed by Xilinx and sold by Digilent that attempts to dramatically improve accessibility and productivity for FPGA-based systems. PYNQ combines a ZYNQ-7020-based FPGA board with Ubuntu Linux, extensive Jupyter[1]-based documentation with embedded Python programming examples, and a hardware “overlay” implemented in ZYNQ’s Programmable Logic (PL). The overhardware functions with associated connectors, Ubuntu Linux, Jupyter notebooks, and a Python interpreter. As an FPGA board, PYNQ is unique for two reasons. First, it is immediately usable “out of the box” by users who have **no** FPGA experience. Second, PYNQ is completely self-documenting. Users simply connect power, insert a provided SD-card, connect an ethernet cable to their laptop or router and point a web-browser

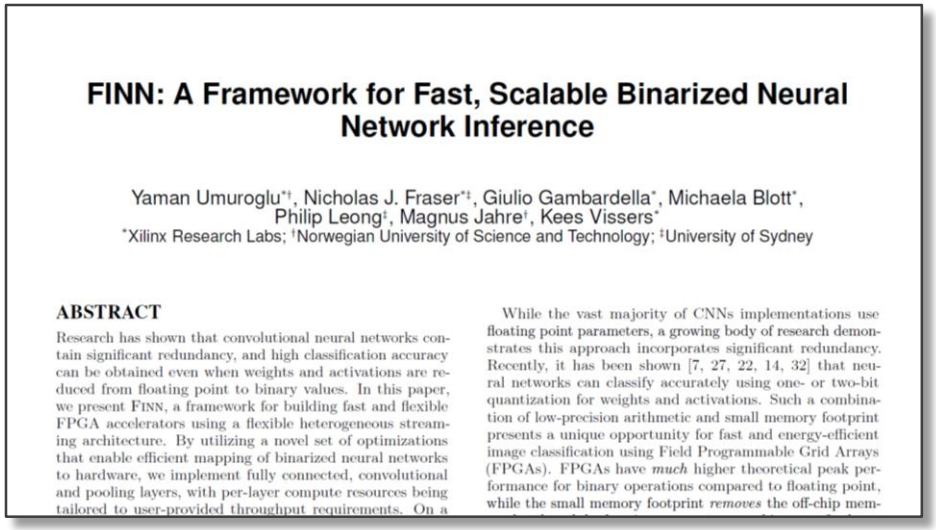
To appear in FPL2017

Partially reconfigurable video overlay



“After teaching students how to use FPGA-based systems for over 20 years, this is the first time we simply handed the boards to the students and **all** of them were able to use and/or add functionality to the system with no supervision.”

FINN: Binary Neural Network Overlay on PYNQ



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

Yaman Umuroglu*†, Nicholas J. Fraser*‡, Giulio Gambardella*, Michaela Blott*, Philip Leong‡, Magnus Jahre†, Kees Vissers*

*Xilinx Research Labs; †Norwegian University of Science and Technology; ‡University of Sydney

ABSTRACT

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

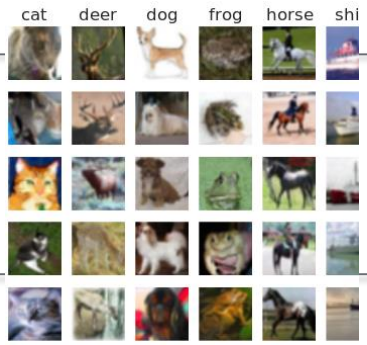
While the vast majority of CNNs implementations use floating point parameters, a growing body of research demonstrates this approach incorporates significant redundancy. Recently, it has been shown [7, 27, 22, 14, 32] that neural networks can classify accurately using one- or two-bit quantization for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

Int. Symposium on FPGAs, Feb. 2017

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

NTNU
Norwegian University of Science and Technology

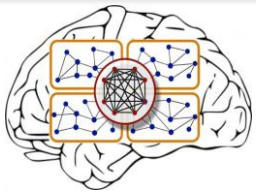
THE UNIVERSITY OF SYDNEY



SVHN,
Road signs,
CIFAR-10

python

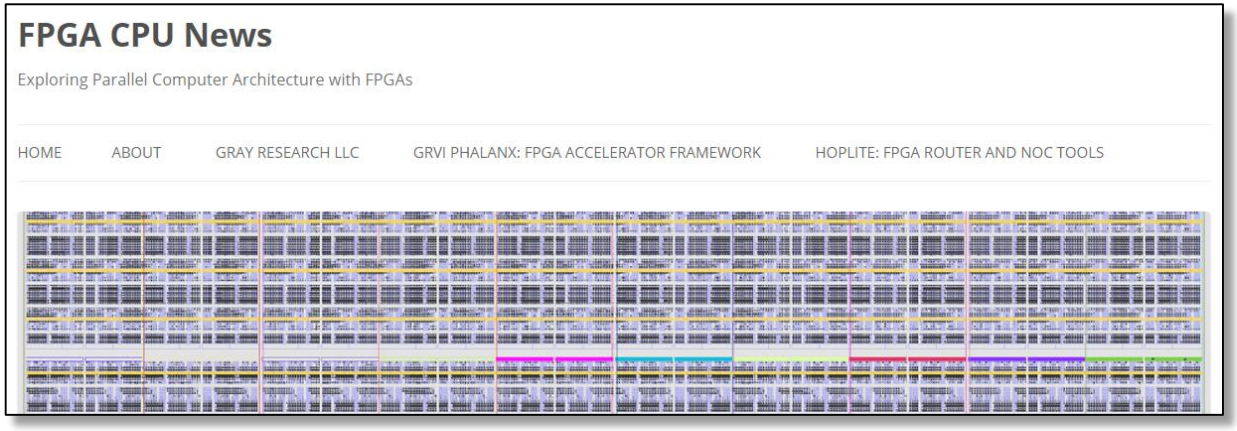
Image pre-processing in Python



Binary Neural Network in FPGA & ARM CPU

“cat”

“A high productivity tool for experienced FPGA designers”



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

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

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

PYNQ: PYTHON PRODUCTIVITY ON ZYNQ

Home | Get Started | PYNQ-Z1 Bo

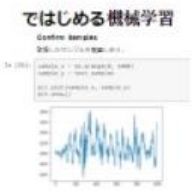
Community Projects

A selection of projects from the PYNQ community is shown below. Note that some examples are built on different versions of the PYNQ image.

Binary Neural Network
Xilinx Labs, NTNU, University Sydney

SPynq:
NTUA Greece

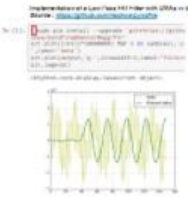
Processing noisy filters
PYNQ Japan user group



Soft GPU for ZC706
Ruhr University Bochum

Video Processing
Vectorblox

FIR filter example
CU Boulder



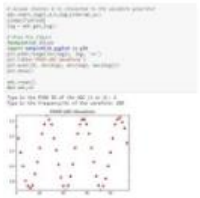
DMA and stream Tutorial

CNN Example
Imperial College London

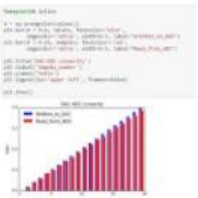
Example Notebooks

A selection of notebook examples are shown below that are included in the PYNQ image. The notebooks contain live code, and generated output from the code can be saved in the notebook. Notebooks can be viewed as webpages, or opened on a Pynq enabled board where the code cells in a notebook can be executed.

ADC waveforms



DAC ADC example



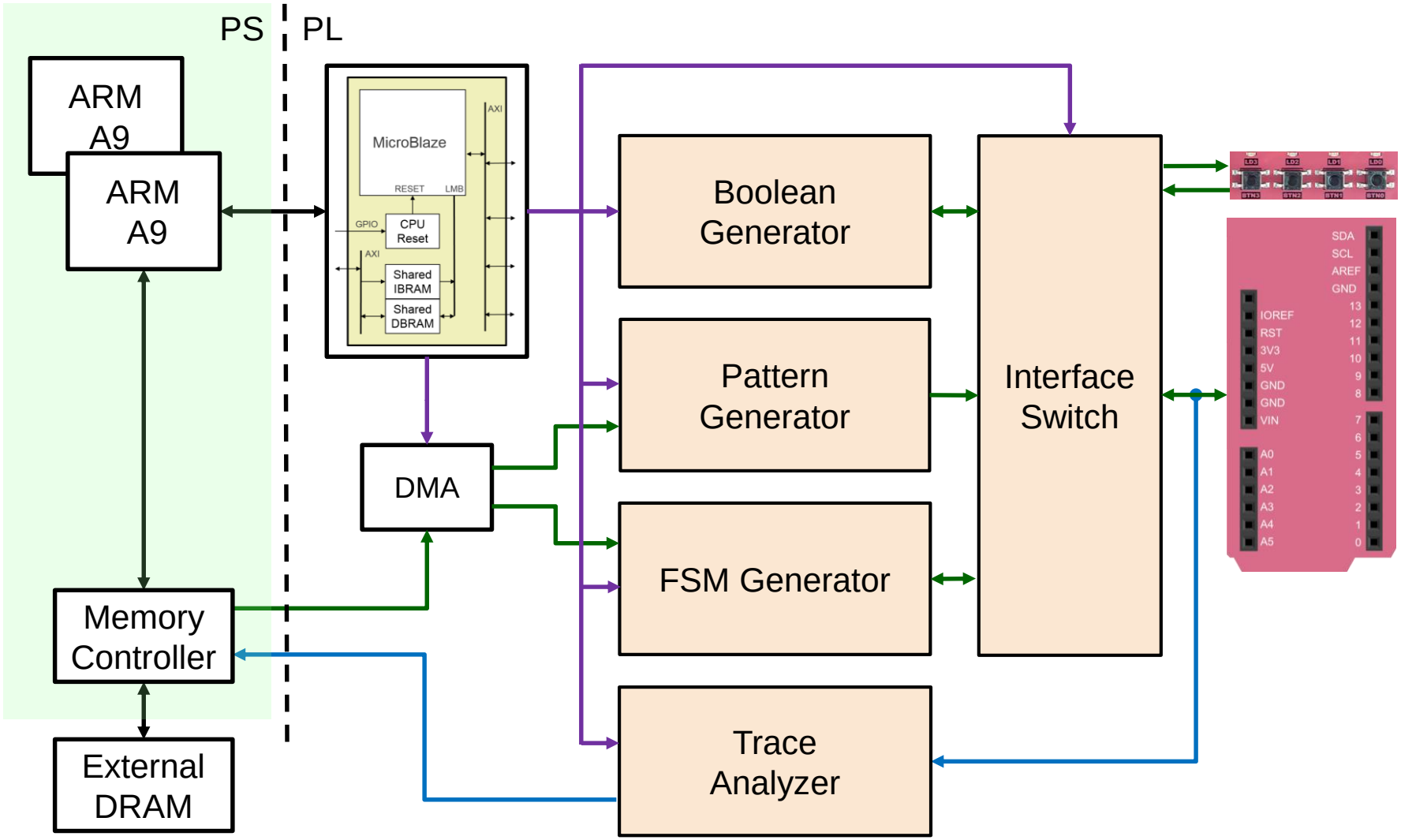
Downloading overlays

Grove ADC

Selection of projects and notebooks

New *Logictools* Overlay

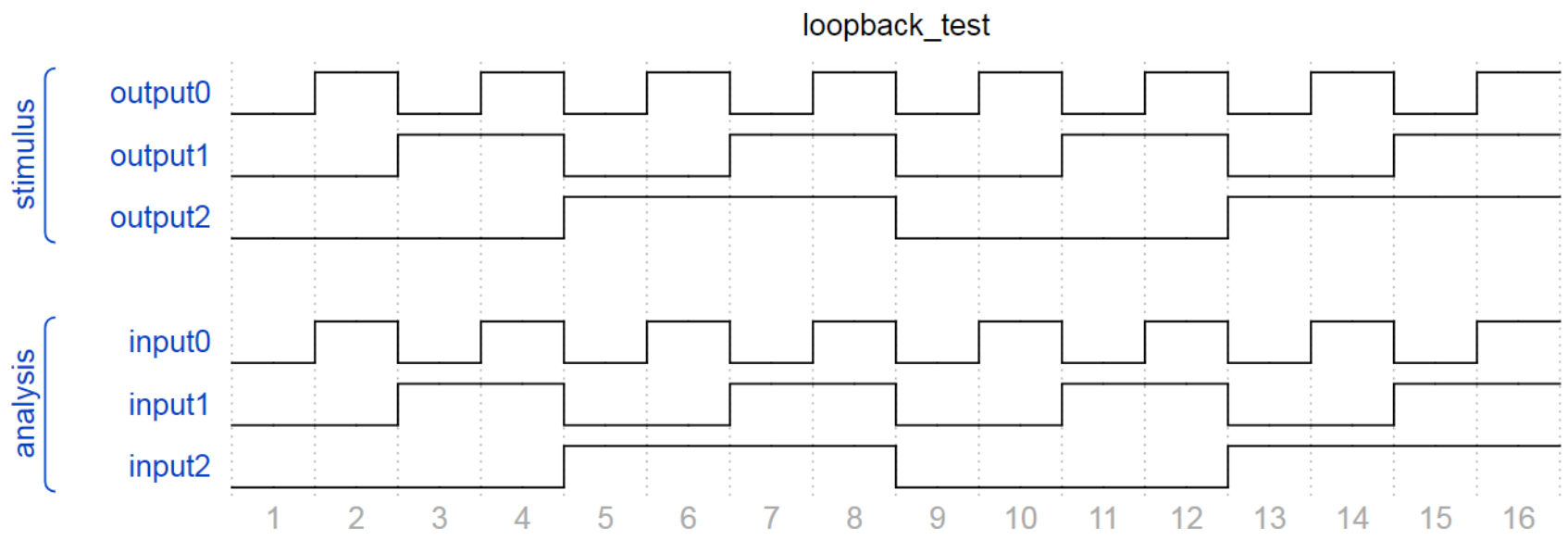
Logictools Overlay



Pattern Generator Example

```
from pynq.lib.logictools import Waveform
from pynq.overlays.logictools import LogicToolsOverlay
from pynq.lib.logictools import PatternGenerator

logictools_olay = LogicToolsOverlay('logictools.bit')
```

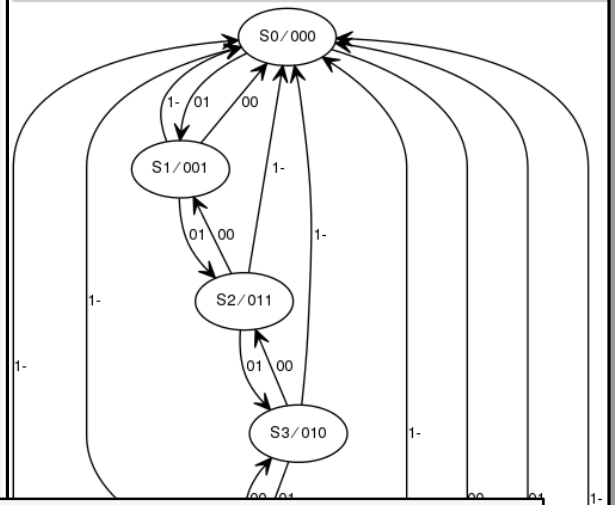
```
pattern_generator = logictools_olay.pattern_generator

pattern_generator.trace(num_analyzer_samples=16)
pattern_generator.setup(loopback_test)

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



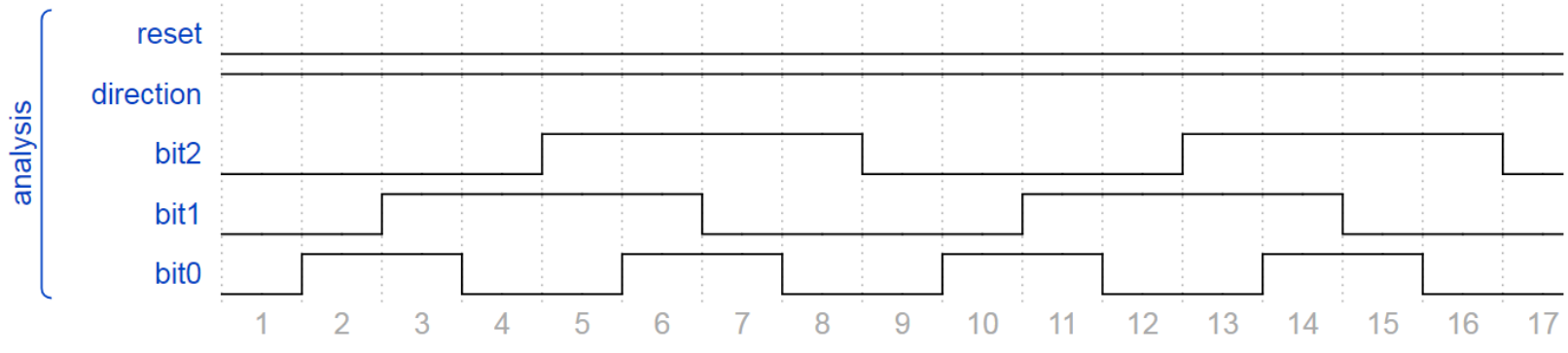
FSM Generator Example

```
fsm_spec = {'inputs': [('reset','D0'), ('direction','D1')],
            'outputs': [('bit2','D3'), ('bit1','D4'), ('bit0','D5')],
            'states': ['S0', 'S1', 'S2', 'S3', 'S4', 'S5', 'S6', 'S7'],
            'transitions': [['01', 'S0', 'S1', '000'],
                            ['00', 'S0', 'S7', '000'],
                            ['01', 'S1', 'S2', '001'],
                            ['00', 'S1', 'S0', '001'],
                            ['01', 'S2', 'S3', '011'],
                            ['00', 'S2', 'S1', '011'],
                            ['01', 'S3', 'S4', '010'],
                            ['00', 'S3', 'S2', '010'],
                            ['01', 'S4', 'S5', '110'],
                            ['00', 'S4', 'S3', '110'],
                            ['01', 'S5', 'S6', '111'],
                            ['00', 'S5', 'S4', '111'],
                            ['01', 'S6', 'S7', '101'],
```

```
fsm_generator.show_state_diagram()
```




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

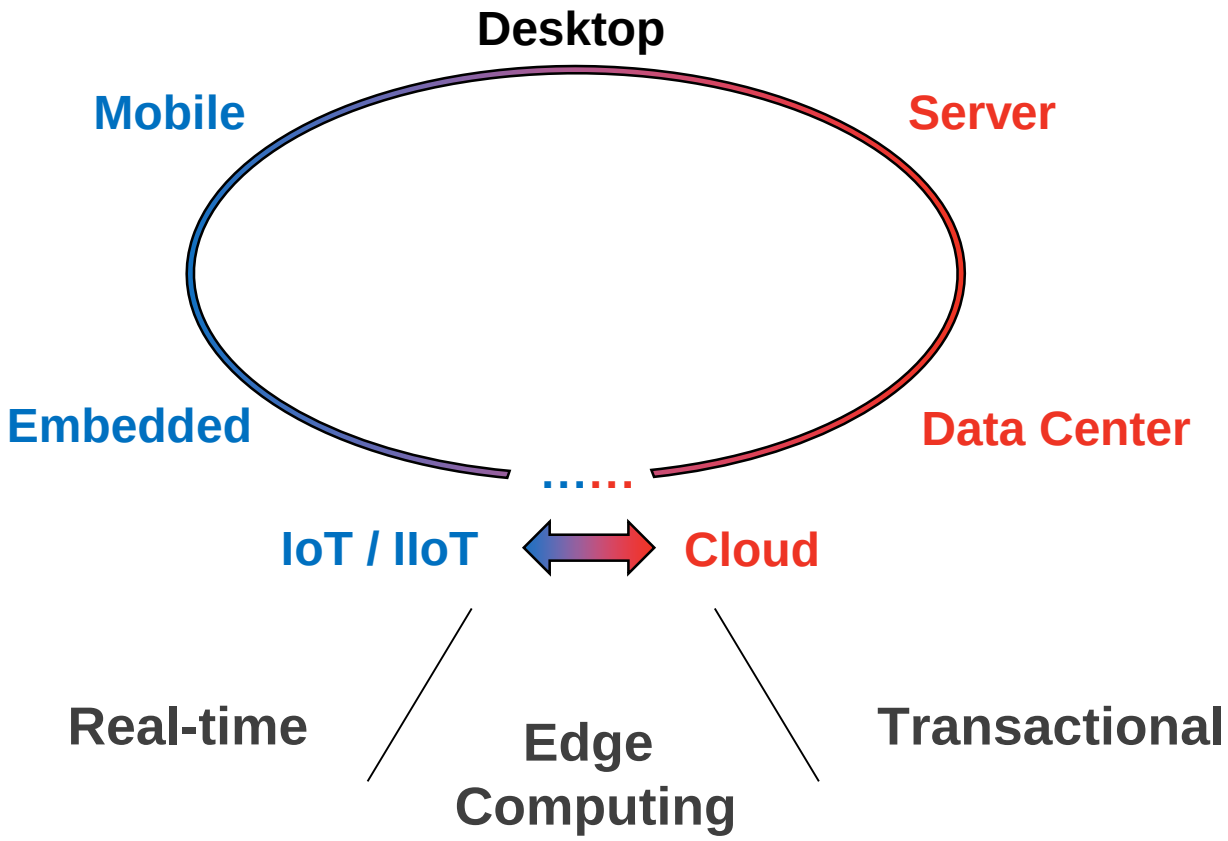


logictools: digital instrument on a chip

- Controlled from web browser
- Declarative
  - Specify the functionality that you want, and where you want it connected
    - Boolean functions, state machines, digital patterns, trace analysis
- Instantaneous realization
  - *logictools* creates it for you … instantaneously
- Reusable software and hardware libraries
  - Library of novel IP available for reuse by overlay developers
    - Useful also to non-Pynq users

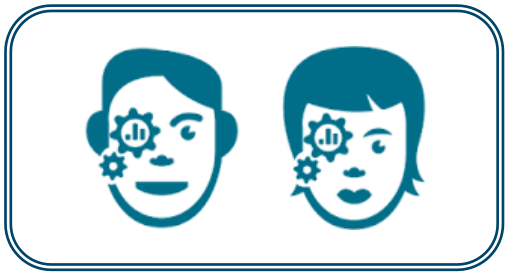
Edge Computing

Computing Landscape … a continuum

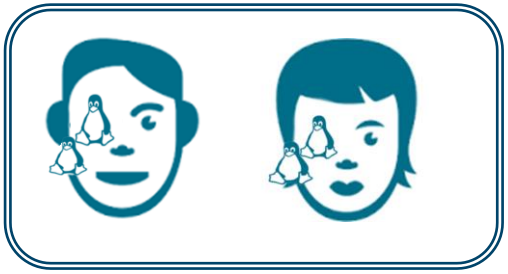


Cisco predicts that 40 percent of IoT data processing will be in the “fog” (aka the edge) by 2018

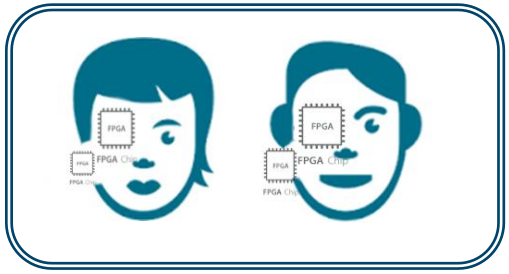
PYNQ enables hardware, software and analytics



Data Scientists



Embedded Engineers



Hardware Engineers

PYNQ™

Analytics

Machine Learning with Scikit-Learn

Abstract sensors

Software

O.S.

sensors

Hardware

sensors

Summary

- CPython & Jupyter make Zynq devices easier to use
- Overlay strategy: experts design the complex parts and export user-friendly APIs to community
- Everything runs on Zynq itself
- PYNQ leverages open source, higher levels of abstraction and systemic re-use
- PYNQ enables embedded hardware, software and analytics
- Open source is the Moore's Law of software

Your feedback is essential for improving PYNQ !