



赛灵思工业物联网研讨会
XILINX IIoT SEMINAR

融入Python生态的开源Zynq软硬件设计框架

PYNQ™



陆佳华

joshual@Xilinx.com

学术与创新生态

边缘计算带来IT、CT和OT的融合



IT向OT注入敏捷灵活的业务应用

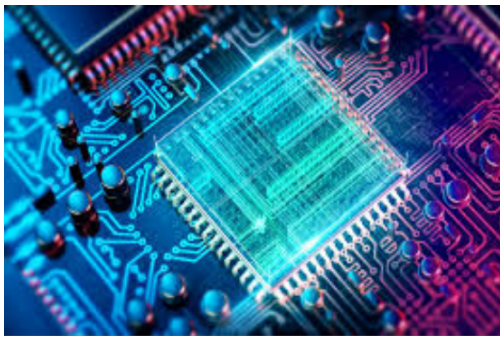
- MES、SCADA等工业软件系统
- 以大数据为基础的预测性分析和预防性维护
- 工业生产流程的数字化与信息化
- 基于机器学习、深度学习的人工智能在工业体系中的应用

CT向OT注入可靠可管的网络服务

- 匹配有线及无线网络的多元化需求
- 低时延, 高可靠、确定性的厂内外网络承载技术
- 云化架构和技术实现网络设备资源共享和多生态应用
- SDN实现网络灵活调度集中优化
- 网络切片实现网络资源的隔离和专享

CT和OT的融合, 是信息、网络和自动化技术的有机结合, 为智能化工业互联网奠定技术基础

来源: 崔爱国@华为 “边缘计算: 对基础软件提出的新挑战” 第三届边缘计算技术研讨会, 杭州, 2019年5月



Compute Acceleration



Computational Storage



**SmartNICs &
Network Acceleration**



Automotive



Wireless Infrastructure



Wired Communications



**Audio, Video, &
Broadcast**



Aerospace & Defense



Industrial, Scientific & Medical



**Test & Measurement,
and Emulation**



Consumer

Three Big Trends

1

Explosion of Data

- > 90% unstructured
- > Video & image content
- > Needs higher throughput & real-time computing

2

Dawn of AI

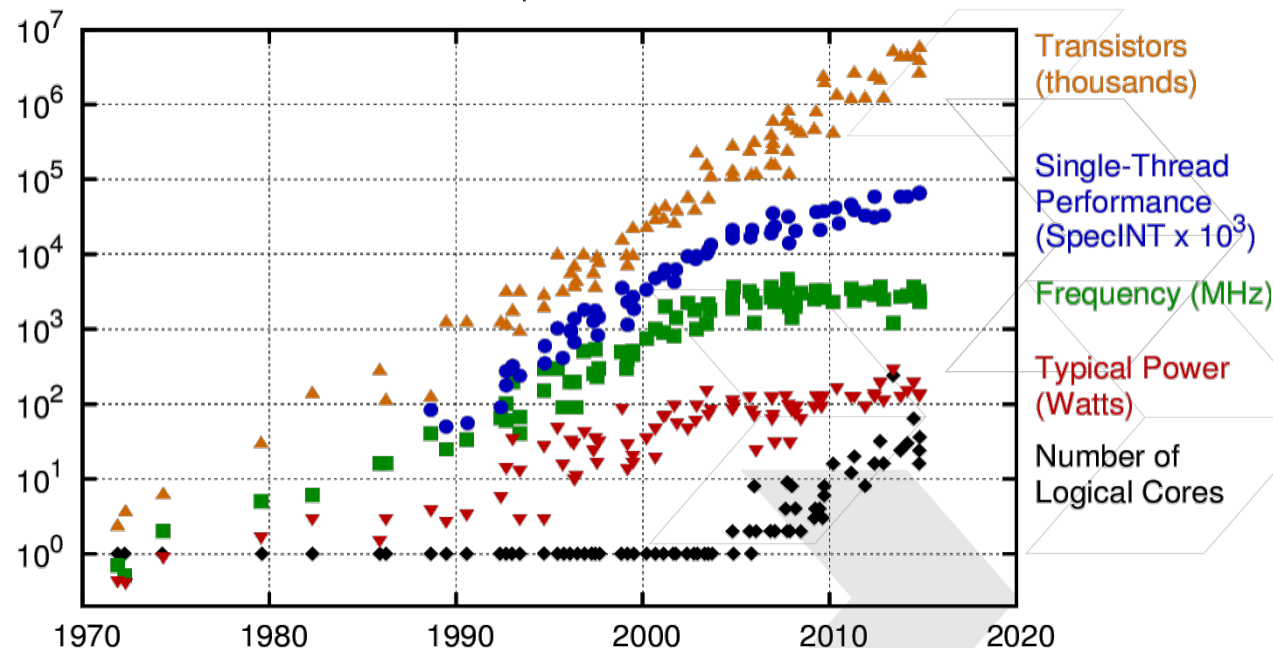
- > Adoption across all industries
- > Injecting new intelligence into apps
- > From endpoints to edge to cloud

3

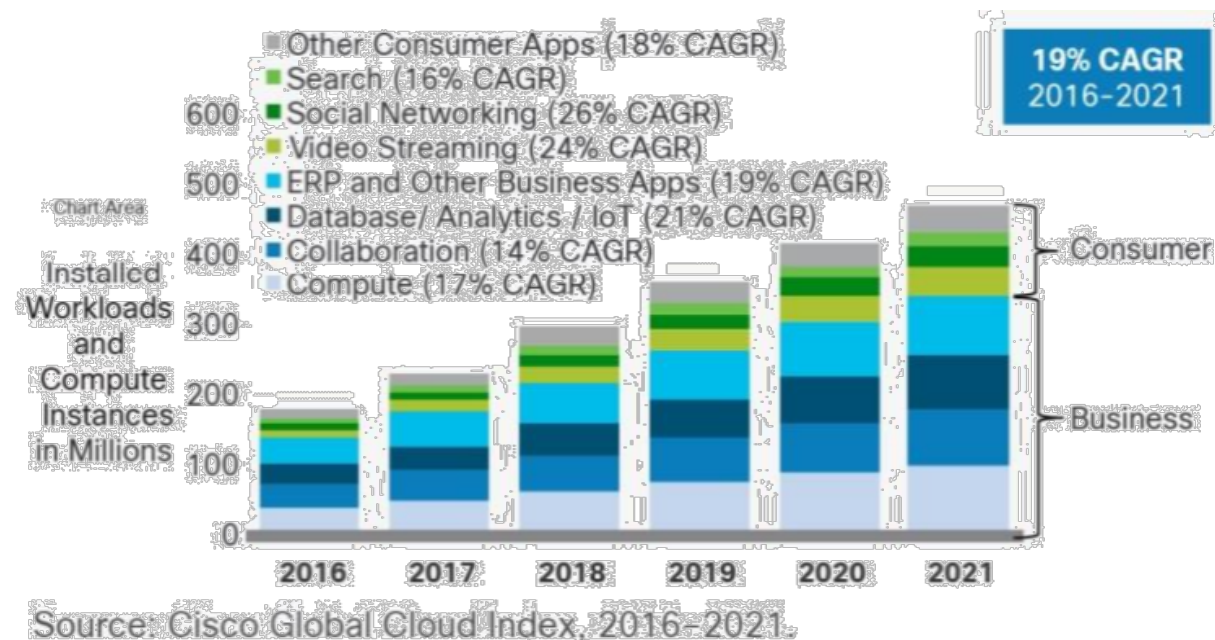
Computing After Moore's law

- > Heterogeneous computing with accelerators
- > Breadth of apps require different architectures
- > Speed of innovation outpacing silicon cycles

40 Years of Microprocessor Trend Data



Diverse and Evolving Applications Needs



- > Top 50 hottest applications only ~60% of data centre compute cycles
 - >> No killer applications
- > Large number of diverse and evolving workloads → **Adaptable Acceleration**
 - >> Compute (Database, Analytics, AI, Video ...), Storage, Networking, Security, etc.
- > AI infusion into traditional workloads driving additional need for adaptable accel


Hennessy & Patterson: A New Golden Age for Computer Architecture

- > Hardware/Software Co-Design for High-Level and Domain-Specific Languages
- > Enhancing Security
- > Free and Open Architectures and Open-Source Implementations
- > Agile Chip Development

02:35 41/42
Time Page

Summary

- There is a data tsunami in science
- Machine learning offers promise to science problems, but high standards of interpretability
- Computing performance will always be needed in science (and elsewhere)
- It's a golden age for computer architectures
- A platinum age for programming systems research
- And at least a bronze age for algorithms



2017 Electronics Resurgence Initiative

- > **Software Defined Hardware (SDH)**
- > **Domain-specific System on Chip (DSSoC)**
- > **Intelligent Design of Electronic Assets (IDEA)**
- > **Push Open Source Hardware (POSH)**
- > **Three Dimensional Monolithic System-on-a-Chip (3DSoC)**
- > **Foundations Required for Novel Compute (FRANC)**

注：互联网、半导体、个人计算机操作系统[UNIX](#)、激光器、[全球定位系统](#)（GPS）等都源于DARPA



HotChip 30 Years



Hot Chips: A Symposium on High Performance Chips

Conference Sponsor IEEE Technical Committee on Microprocessors and Microcomputers (TC-MICRO) and the IEEE Computer Society (SIGARCH).



Transformation Through Innovation

World's
First FPGA



1980

First
Virtex FPGA



1990

Virtex-2 Pro



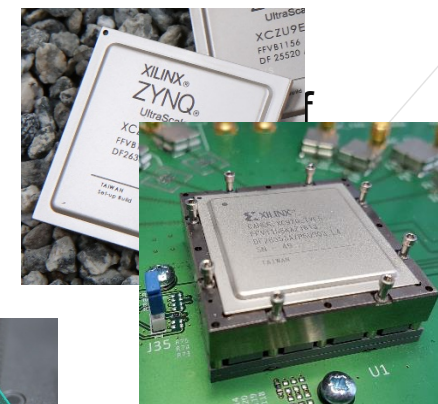
2000

First 3D FPGA
& HW/SW
Programmable SoC



2010

First
MPSoC & RFSoc



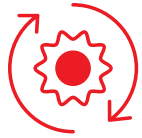
2020

ACAP



Driving Adaptive Computing with Versal

The World's First Adaptive Compute Acceleration Platform



Scalar Processing Engines



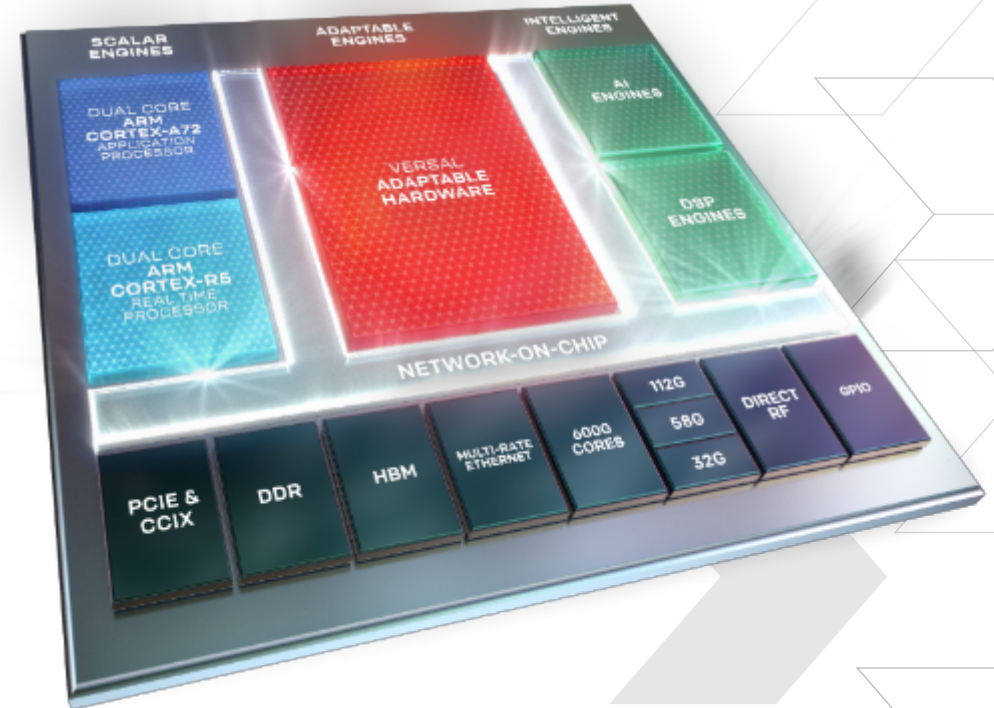
Adaptable Hardware Engines



Intelligent Engines
SW Programmable, HW Adaptable



Breakout Integration of Advanced
Protocol Engines



Harnessing the Flexibility of Programmable Hardware with Design Tools

Increasing Productivity

VIVADO

State-of-the-Art Tools for Traditional Hardware Design

- > Optimizes for performance, low power, and density
- > Advanced flows in-system updates and remote debug

HLx
Editions

15X Productivity for Hardware Developers

- > Graphical plug-and-play IP Integration
- > High Level Synthesis in C, C++, and System C

SDx
Environments

Enables Full Programmability for Software Developers

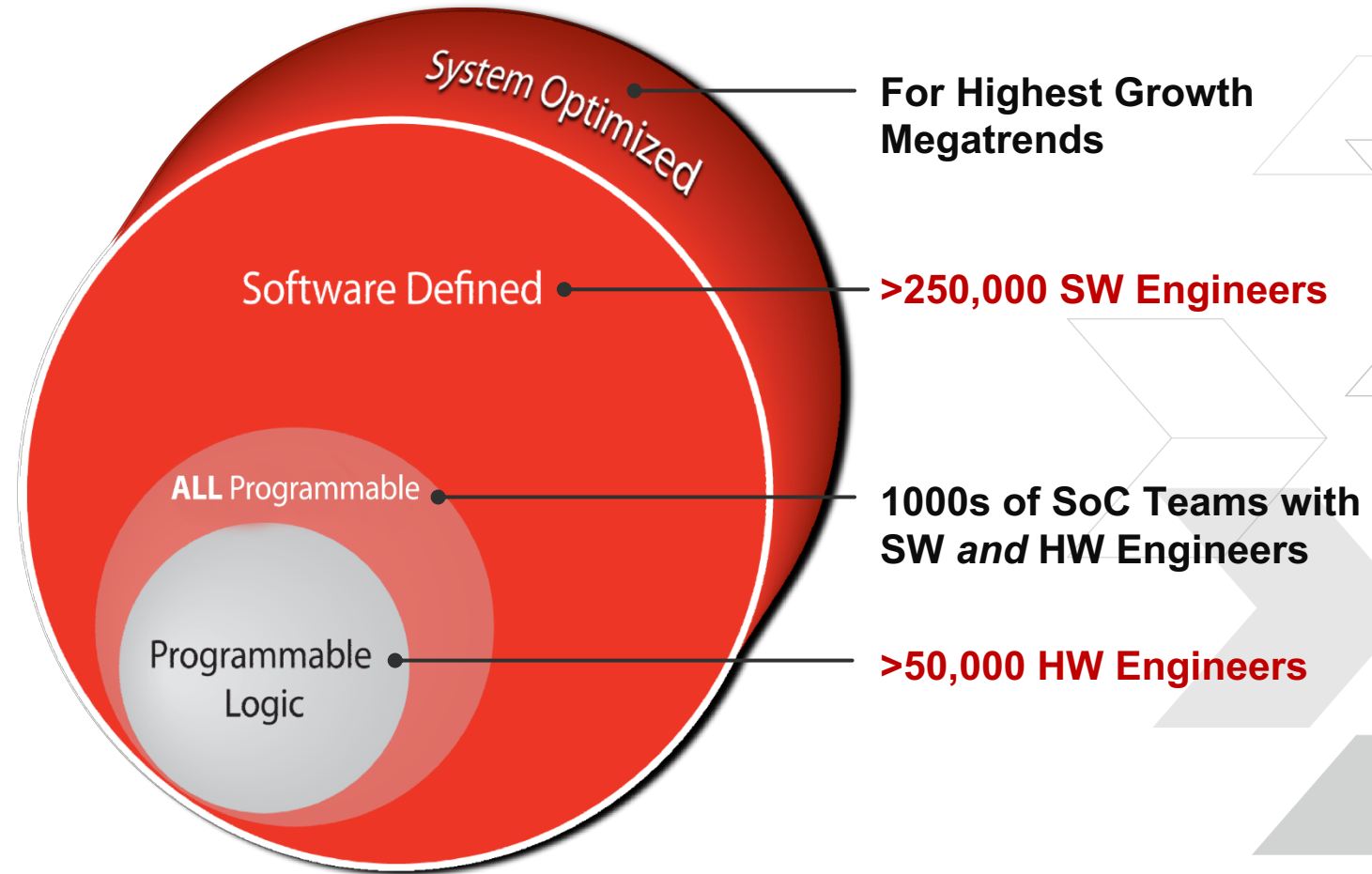
- > CPU and GPU-like development environments
- > Automated acceleration in programmable logic

Enabling Innovation Open Source Pynq Framework

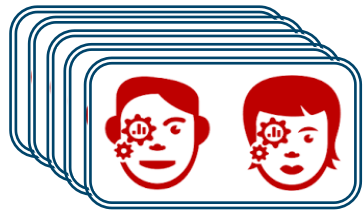


Setting the Course for the Next 5 Years

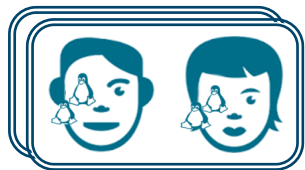
Goal: 5X Potential Users in 5 Years



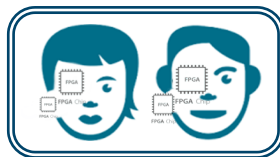
The Goal is Proliferation



Domain Experts,
Software engineers



Emb. software
Engineers



Hardware
Engineers

Modern projects have many, more software engineers

How do we get them to use Xilinx devices?

How can we let them experience Xilinx advantages?

We must speak to them in their 'language'

They do not use CAD tools



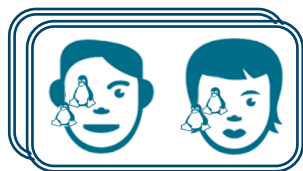
Is a framework for creating platforms that software engineers can use to experience the benefits of Xilinx silicon

PYNQ enables rapid development for FPGA designers and embedded s/w engineers also

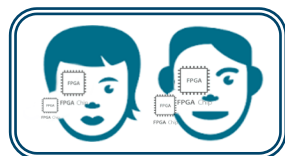
PYNQ™ Python Productivity for Zynq



Domain Experts



Embedded software Engineers



Hardware Engineers

Targeting the data center artificial intelligence, machine learning, data science

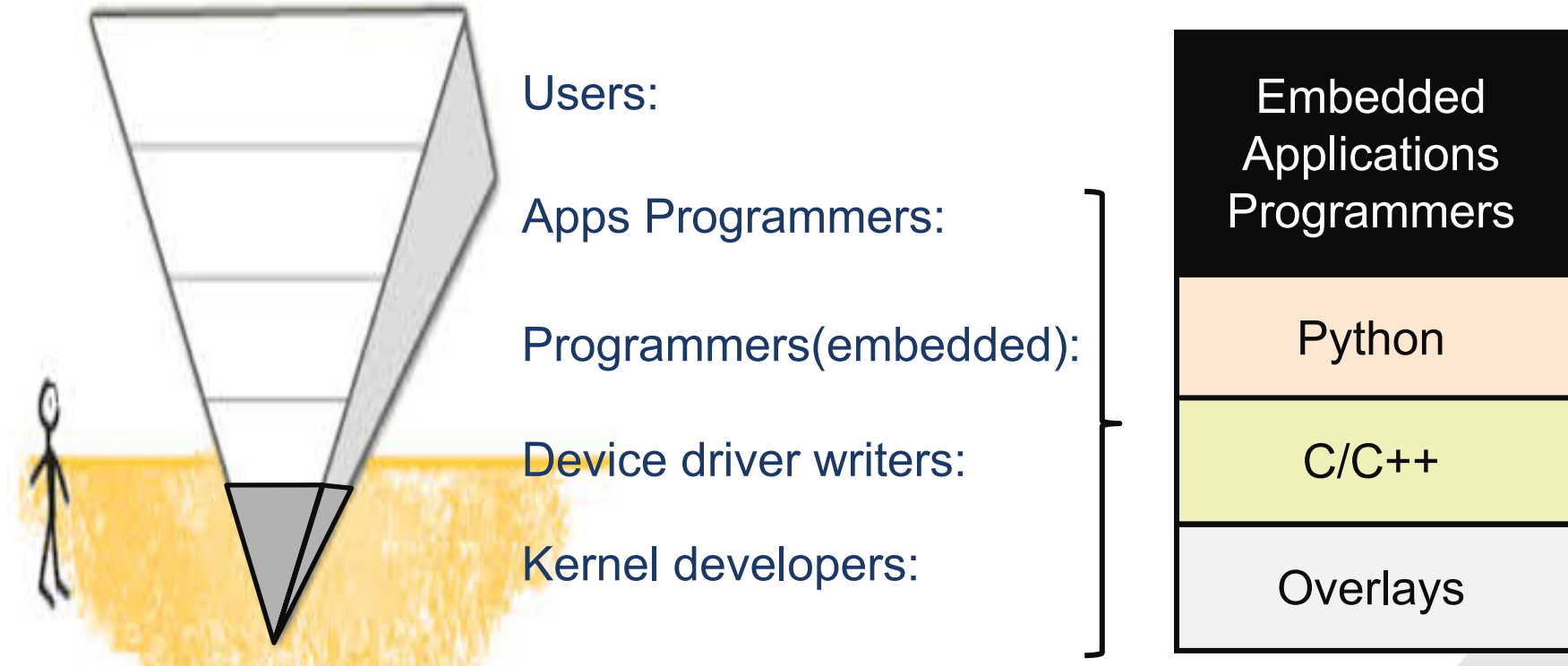
New users are not hardware designers, or embedded systems designers

PYNQ™

Enable more people to program Xilinx processing platforms, more productively

Productivity Languages & Hardware Overlays

Zynq / Zynq UltraScale+



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

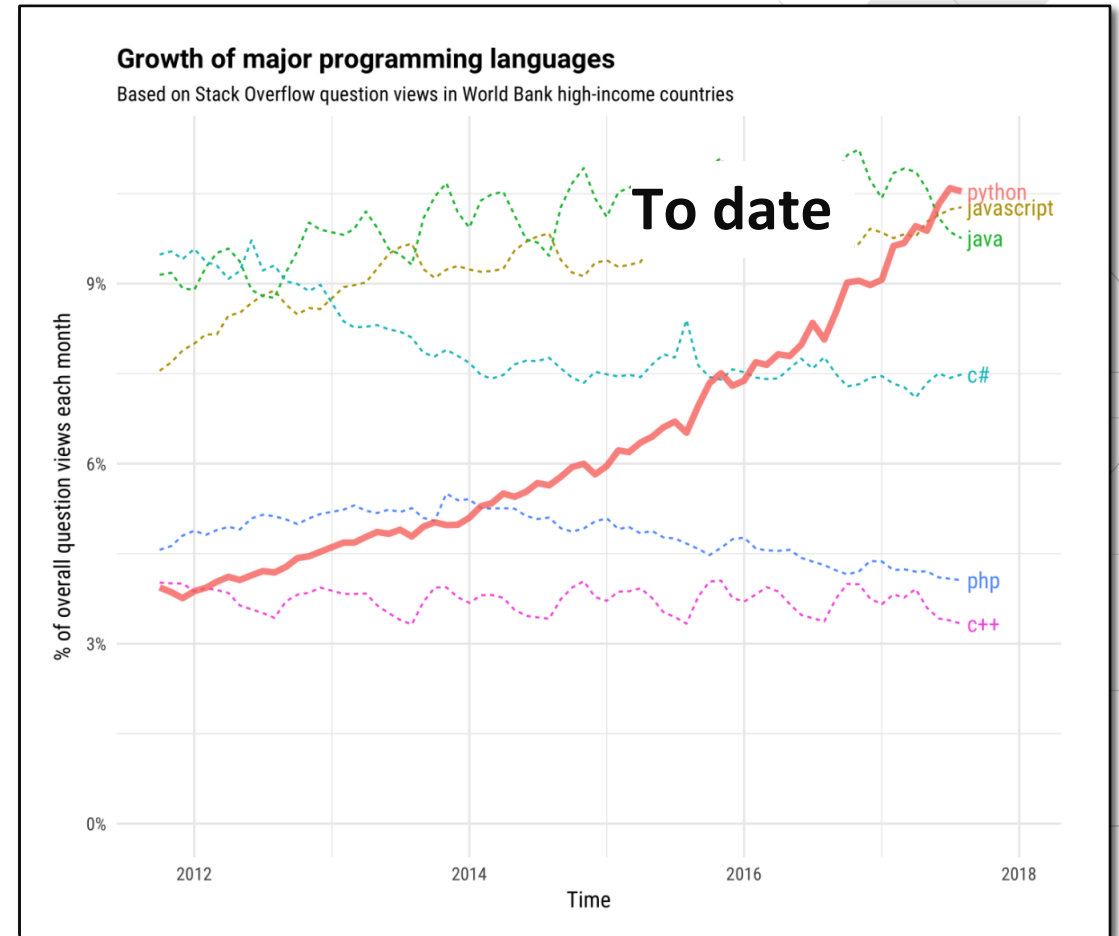
Python is increasingly the Language of Choice

Top Programming Languages,
IEEE Spectrum, July'18

Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥️ 📱	100.0
2. C++	📱 🖥️ 📱	98.4
3. C	📱 🖥️ 📱	98.2
4. Java	🌐 📱 🖥️	97.5
5. C#	🌐 📱 🖥️	89.8
6. PHP	🌐	85.4
7. R	🖥️	83.3
8. JavaScript	🌐 📱	82.8
9. Go	🌐 🖥️	76.7
10. Assembly	📱	74.5

Python is listed as an
embedded language
for the first time

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



<https://stackoverflow.blog/2017/09/06/incredible-growth-python/>

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

Jupyter Notebooks to JupyterLab IDE

2017 ACM
Software System Award

Code editor Terminal

The screenshot displays the JupyterLab IDE interface. On the left, a code editor shows Python code for image processing using Darknet. The middle panel displays the output of the code, including a classification result for a bicycle image with bounding boxes for 'bicycle', 'dog', and 'car'. On the right, a terminal window shows the execution of a script. Below the terminal, a visualization panel displays a waveform and a state transition diagram for an FSM. The waveform shows the output of the FSM against the expected Gray code count sequence. The state transition diagram shows the sequence of states (S0/000, S1/001, S2/011, S3/010, S4/110, S5/1) and transitions between them.

Jupyter notebooks

Visualization

Jupyter ... Julia, Python, R
Default engine of data science

Taught to 1,000+ Berkeley
students every semester

2+ million notebooks
on GitHub

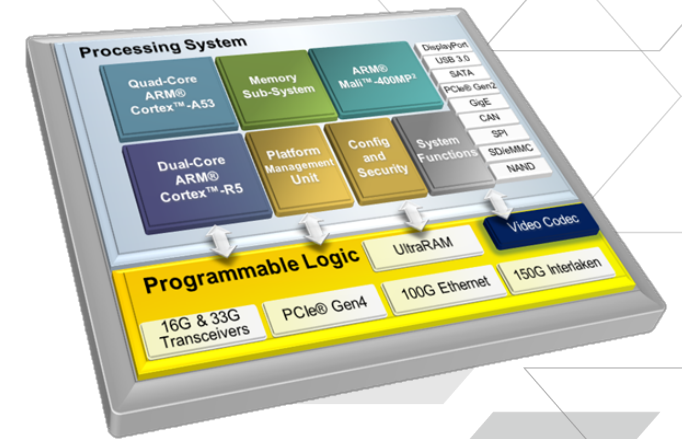
Next-gen browser IDE

Includes Jupyter Notebooks

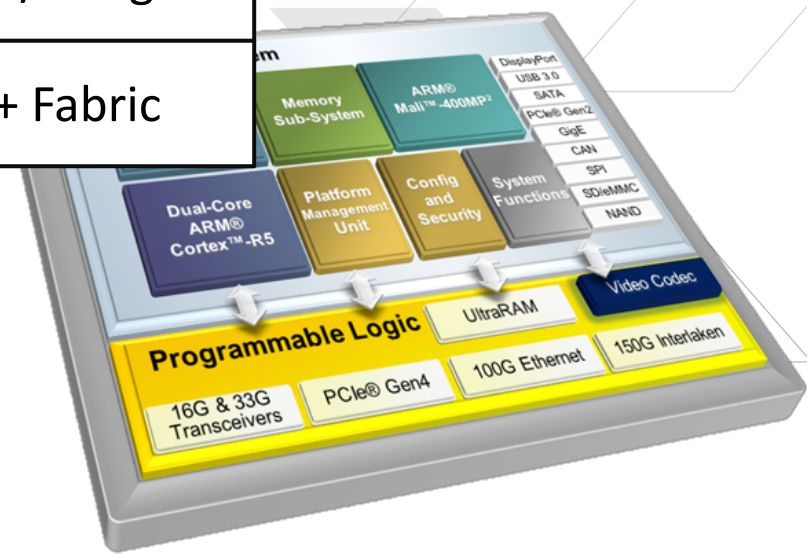
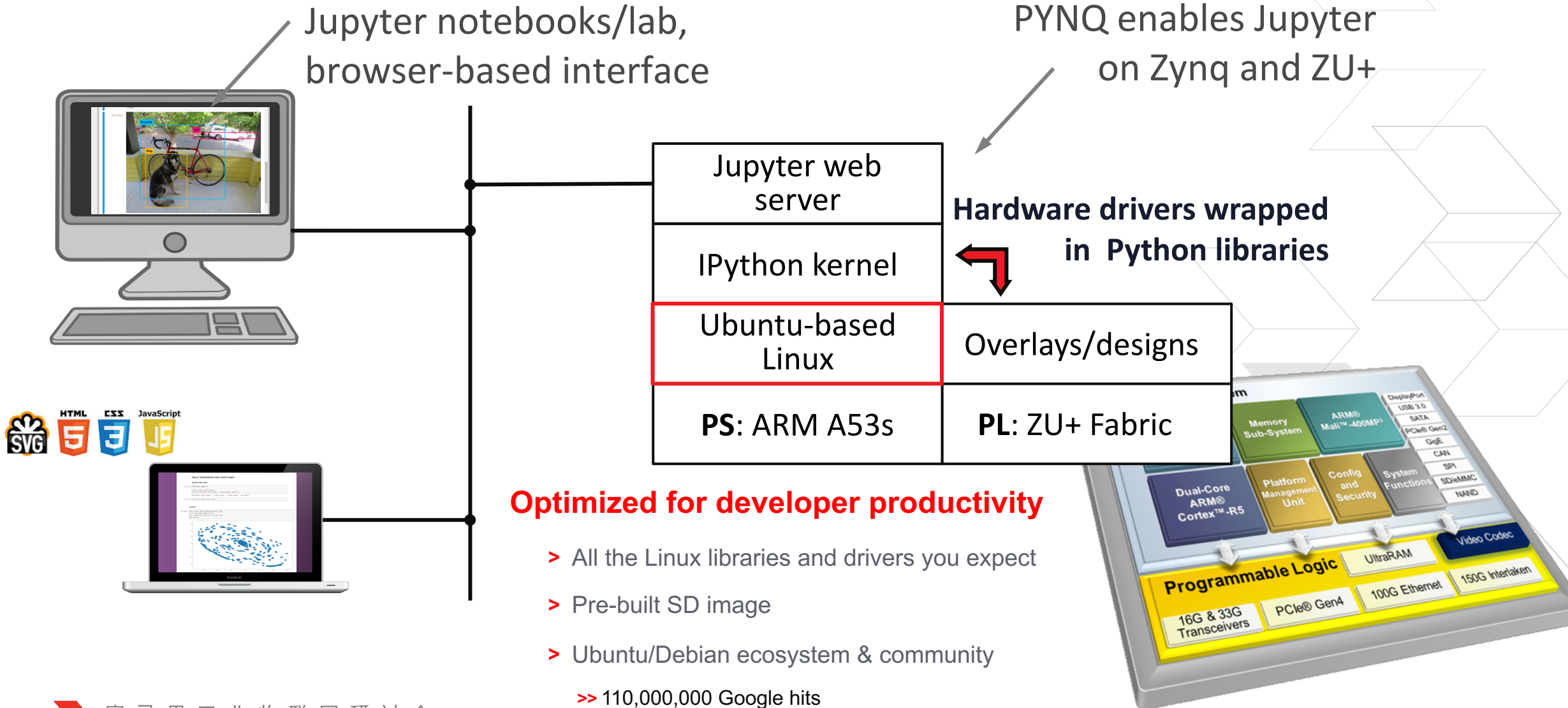
JupyterLab runs on Zynq and Zynq UltraScale+

The screenshot displays a JupyterLab interface with several open notebooks and a terminal. The main notebook shows Python code for object detection using Darknet. A section titled "6. Draw detection boxes using Darknet" explains the image postprocessing. Below the code, the results are shown as a photograph of a street scene with bounding boxes around a bicycle, a dog, and a car. Another notebook, "fsm_generators", shows code for running and displaying a waveform. The terminal window shows the execution of a script on a Zynq device, displaying system information and a directory tree of files.

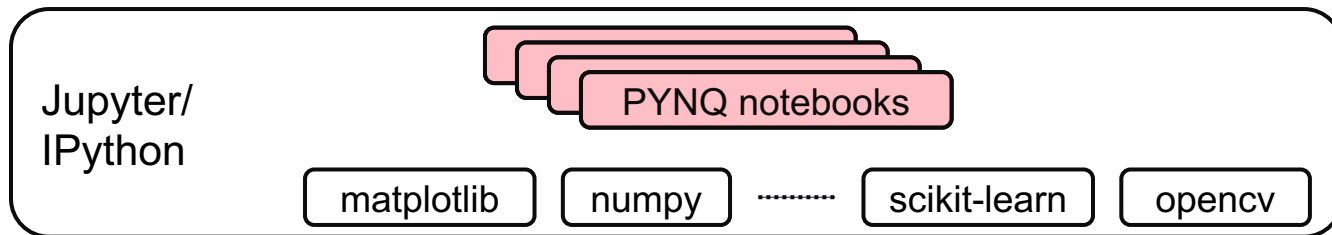
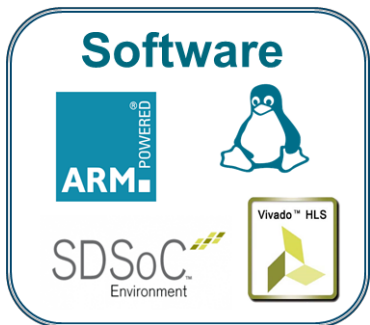
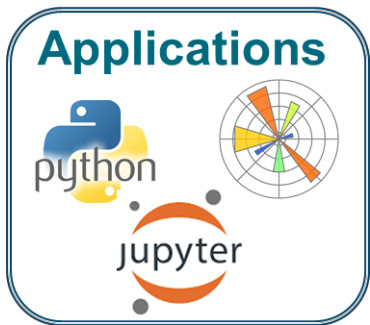
Runs natively on ARM A9 and ARM A53 processors



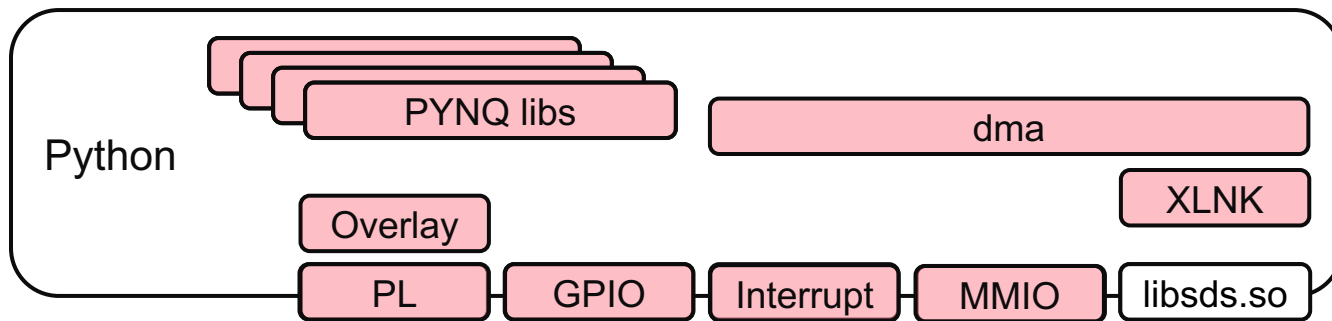
Performant on Zynq 32-bit ARM A9 at 650 MHz, 512MB DRAM, no GPU



PYNQ™ is a Framework



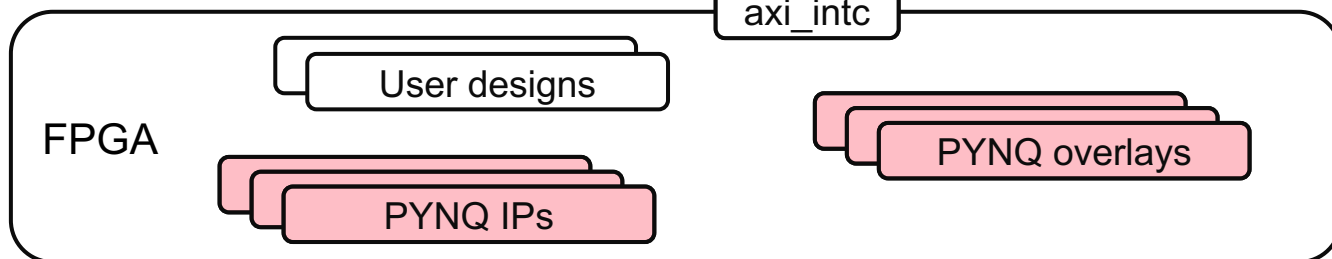
} Apps



} APIs



} Drivers



} Bitstreams

} PYNQ™

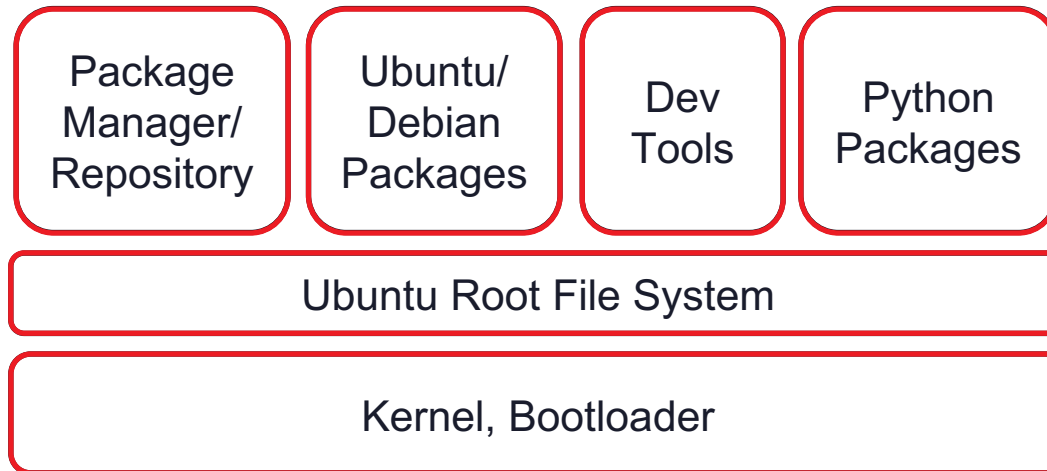
PYNQ's Ubuntu-based Linux

PYNQ uses Ubuntu's:

- Root file system (RFS)
- Package manager (*apt-get*)
- Repositories

PYNQ bundles :

- Development tools
 - Cross-compilers
- Latest Python packages



PYNQ uses the PetaLinux build flow and board support package:

- Access to all Xilinx kernel patches
- Works with any Xilinx supported board
- **Configured with additional drivers for PS-PL interfaces**

Ubuntu-based Linux versus embedded Linux

Ubuntu-based Linux



➤ Optimized for developer productivity

- > All the Linux libraries and drivers you expect
- > Pre-built SD card image
- > Ubuntu/Debian ecosystem & community

>> 145,000,000 Google hits

Embedded Linux



➤ Optimized for deployment efficiency

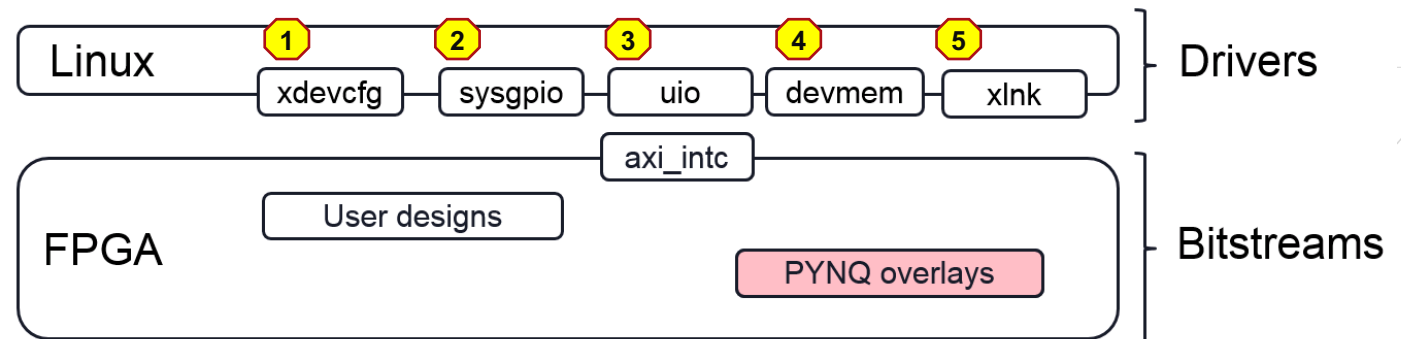
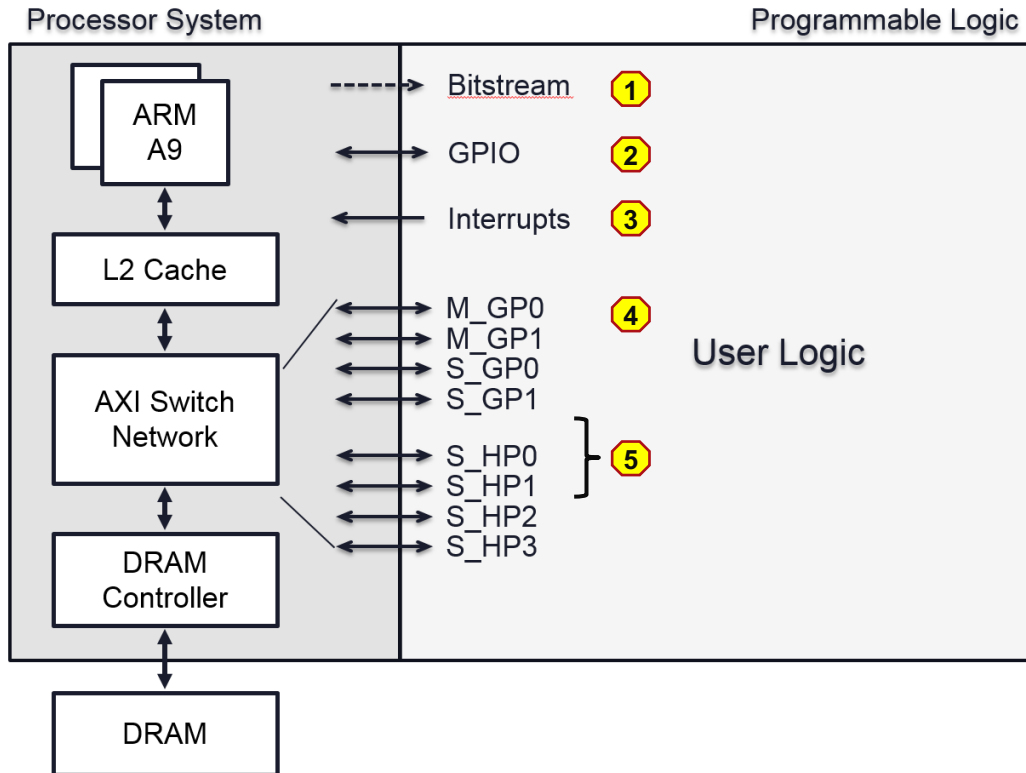
- > Selective Linux libraries and drivers
- > Commonly delivered in flash memory on board
- > PetaLinux ecosystem:

>> 143,000 Google hits

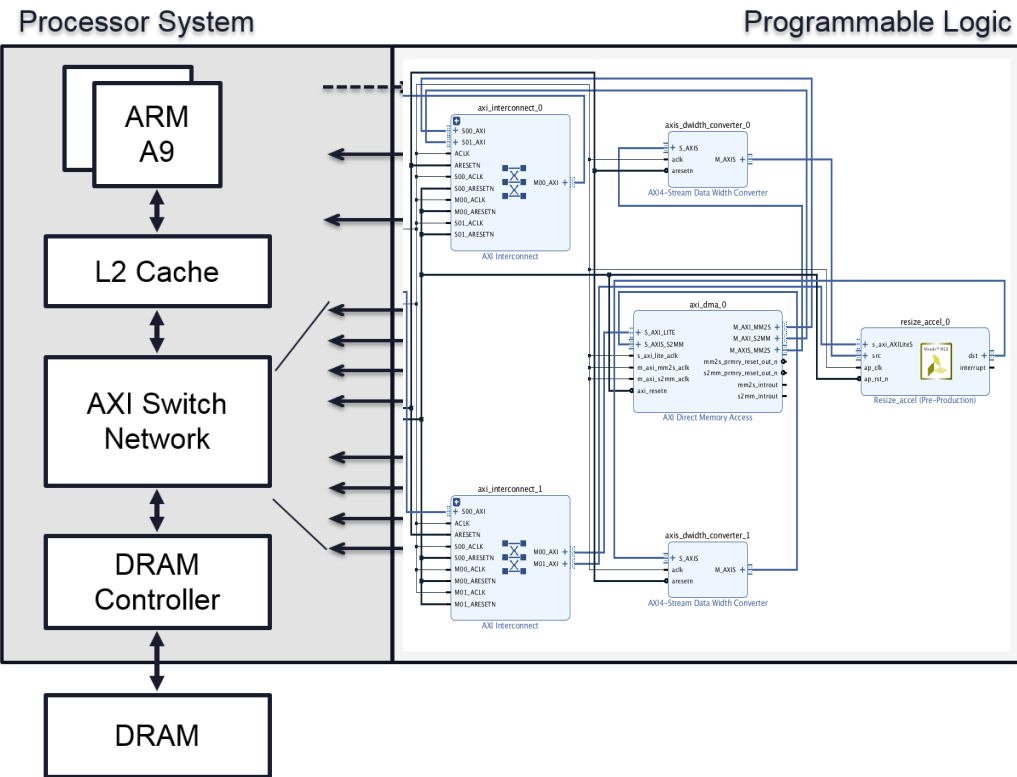
3 orders of magnitude difference

PYNQ provides Linux drivers for PS-PL i/fs ...

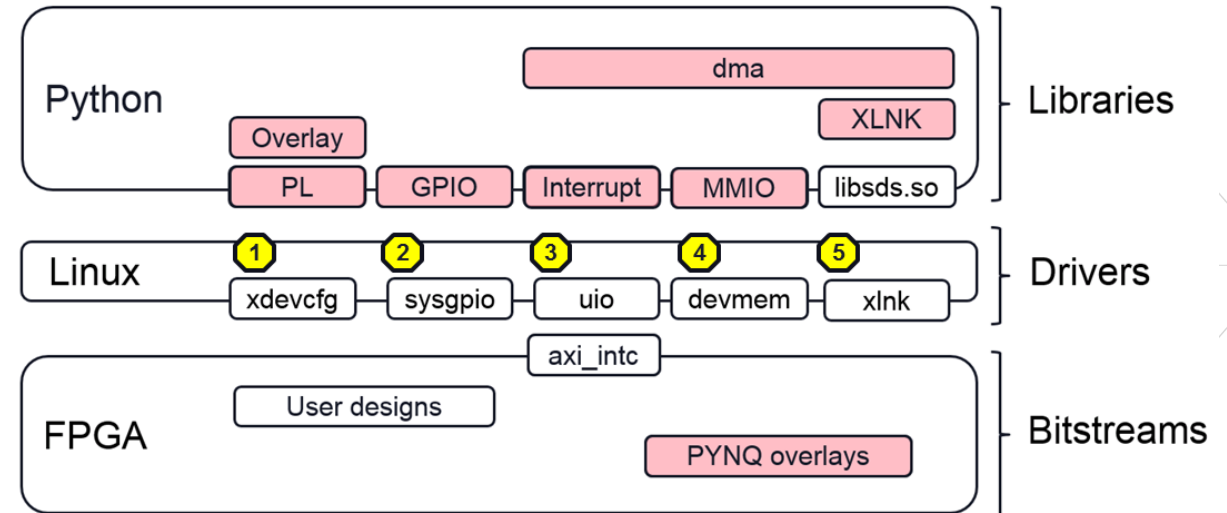
wrapped in Python libraries



Loading a design into Zynq using PYNQ



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



PYNQ automatically configures many design parameters based on data parsed from hybrid library

Software-style packaging & distribution of designs

3. Open image to be classified

Pick the random image from the imagenet 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)
in = Image.open(img_file)
if img_class in synsets.keys():
    print(synsets[img_class])
in
```

Out[4]:

SPYN - III phase AC motor control

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

Objectives

- Access to Motor Control Parameters
- Revised Status Information of the Motor
- Programmable Control of Motor
- Continuous Status Capture from the Motor
- Plot to Visualize Data Collected
- Store Collected Data for Analytics
- Link Interactive Plots to Ingestable Data

Step 1: Download the EDDP bitstream

```
In [1]: from pynq import overlay
from pynq import mtd
import numpy as np

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

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

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:

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:

- Set up HDMI drivers
- Run software only filter2D + dilate pipeline on HDMI input and output results to HDMI output
- Run hardware accelerated dilate function
- Set up widget for controlling different filter kernels
- 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/output. 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_computer_vision import BaseHDMIOverlay
base = BaseHDMIOverlay("/opt/python3.6/lib/python3.6/site-packages/"
    "pynq_computer_vision/overlay/computer_vision/"
    "v2p1iter2001ite.bit")

from pynq import Xilinx
mem_manager = Xilinx()
import pynq_computer_vision.overlays.computer_vision.v2p1iter2001ite as v2

hdm_in = base.video.hdm_in
hdm_out = base.video.hdm_out
```

Out[12]:

Download a design from GitHub with a single Python command:

```
pip install git+https://github.com/Xilinx/pynqDL.git
```

Next steps: scaling across platforms and domains

ISM ML

IIoT vision

SDR

SciPy Jupyter

Software

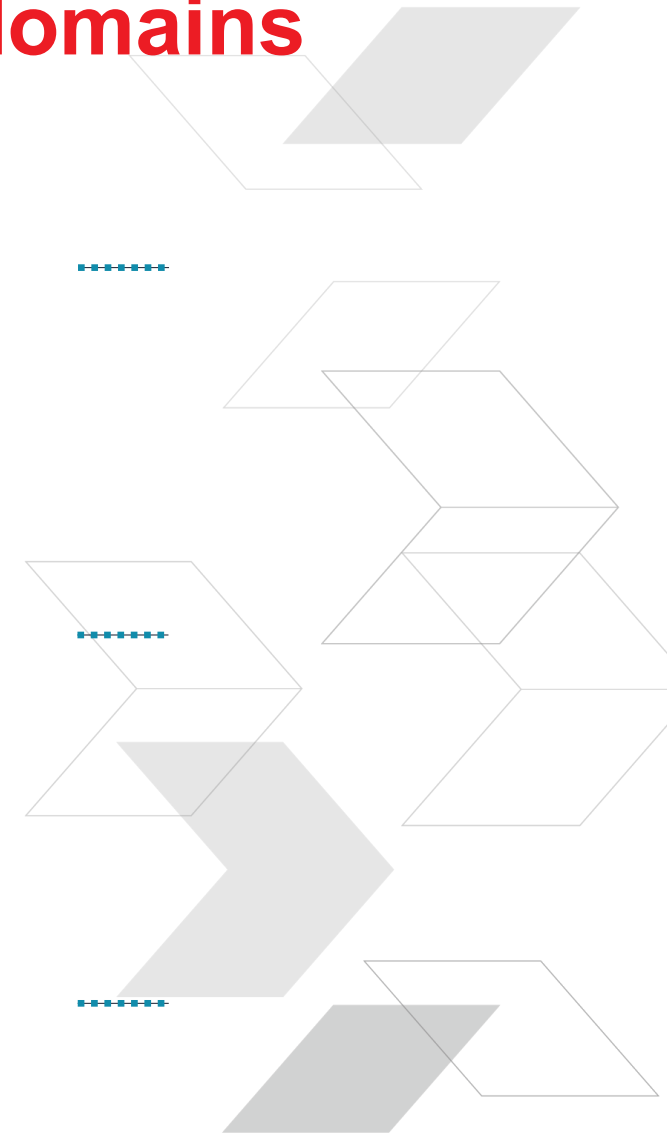
Software

Software

Zynq

ZynqU+

RFSoc



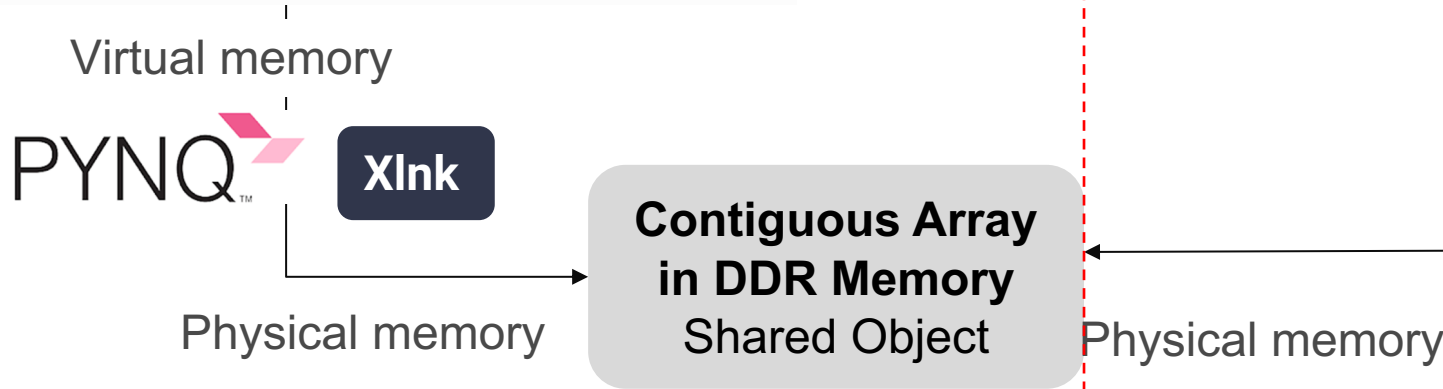
How Numpy interacts with Programmable Logic?



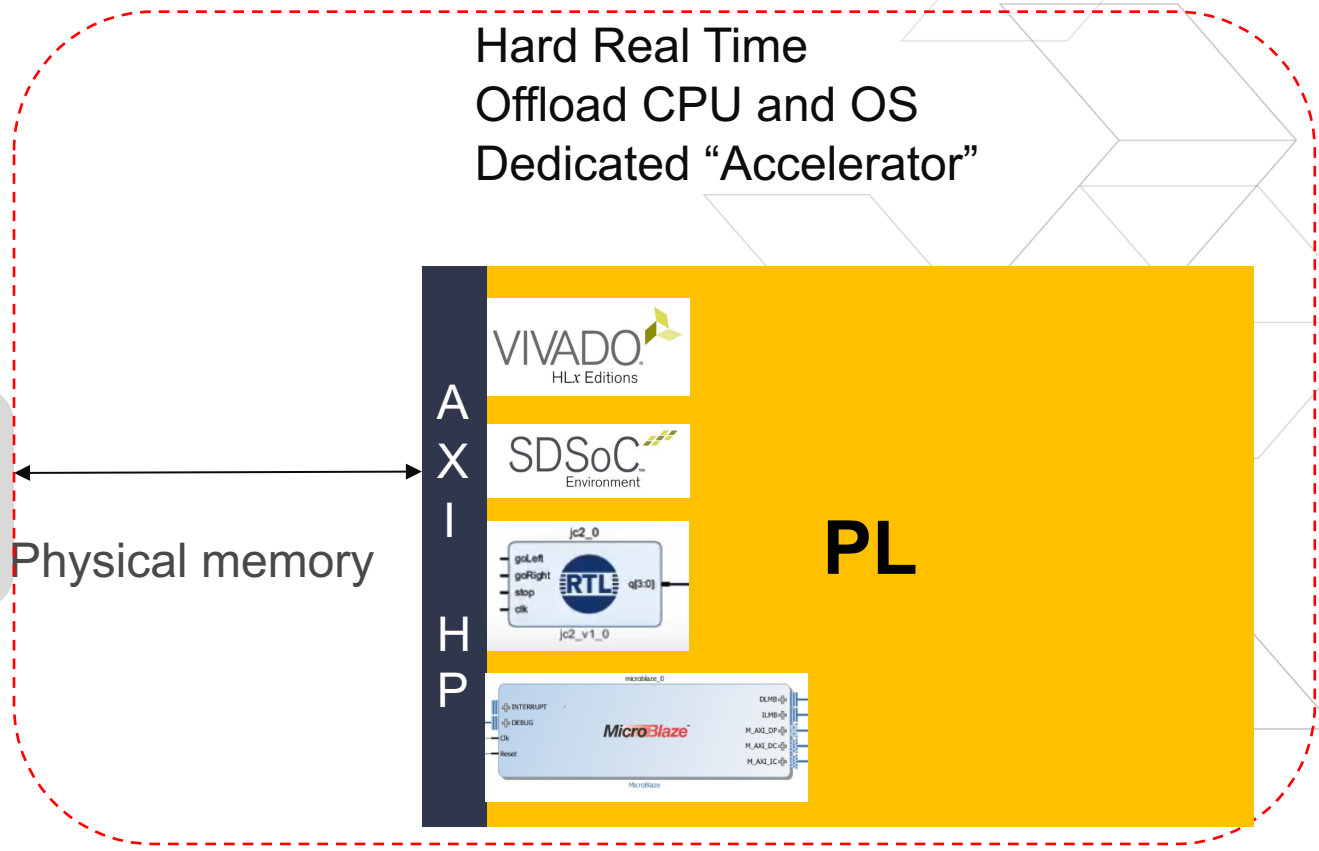
↑ Array, Matrix

```
In [7]: import numpy as np
import pynq

def get_pynq_buffer(shape, dtype):
    """ Simple function to call PYNQ's memory allocator with numpy attributes
    """
    return pynq.Xlnk().cma_array(shape, dtype)
```

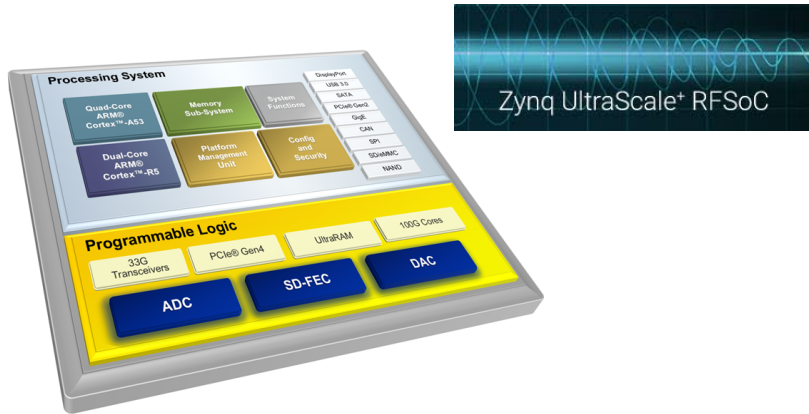


Hard Real Time
Offload CPU and OS
Dedicated "Accelerator"

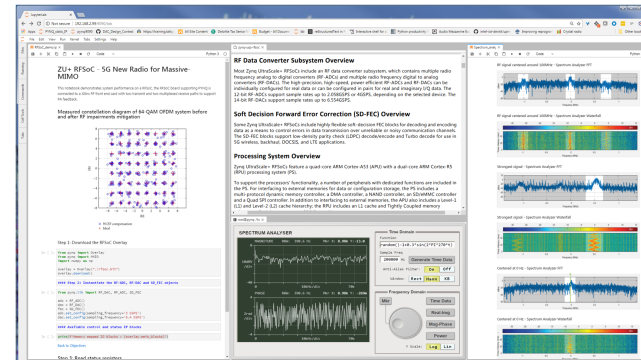


BIG DATA tools for BIG DATA devices

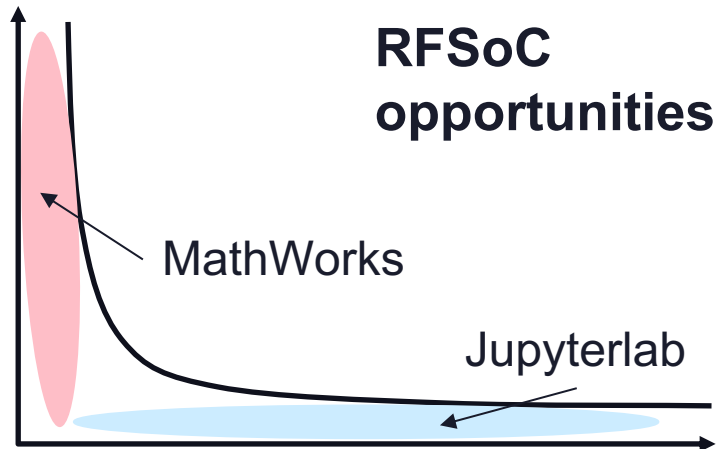
BIG DATA device



BIG DATA IDE



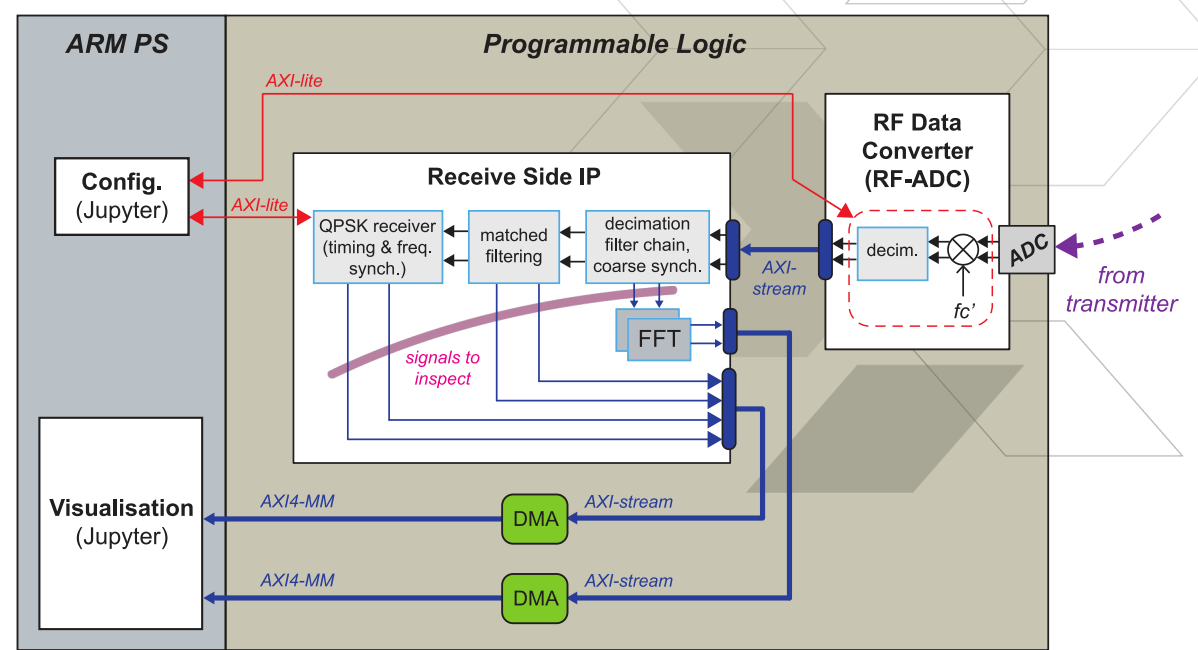
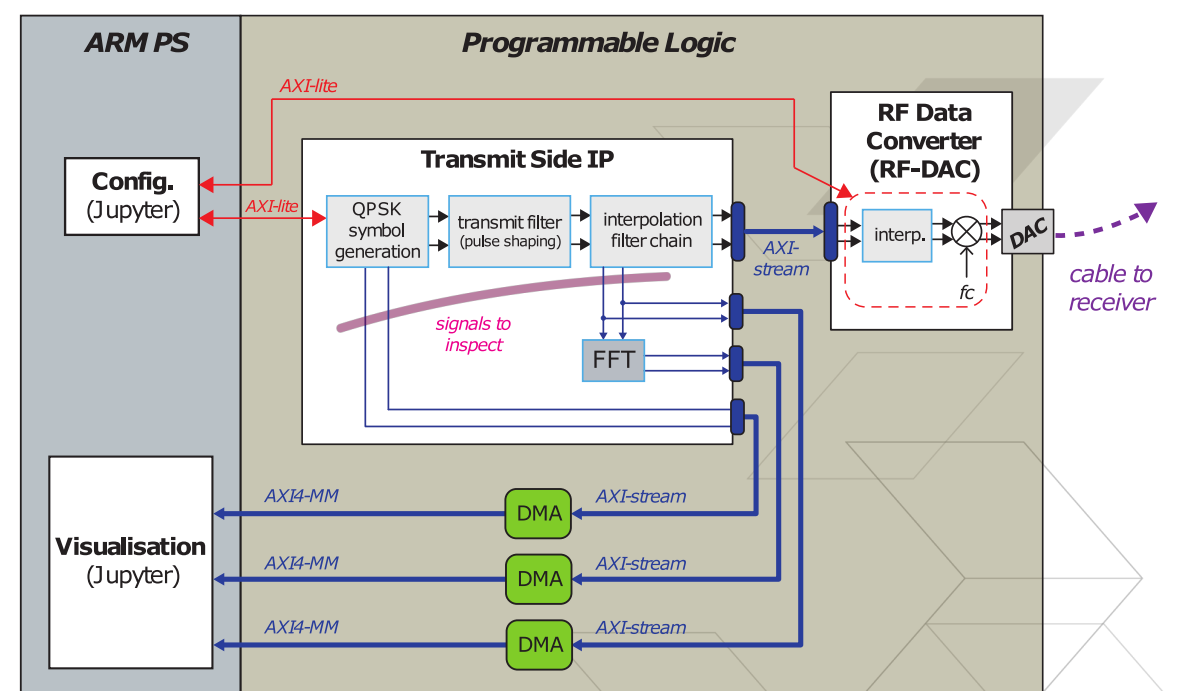
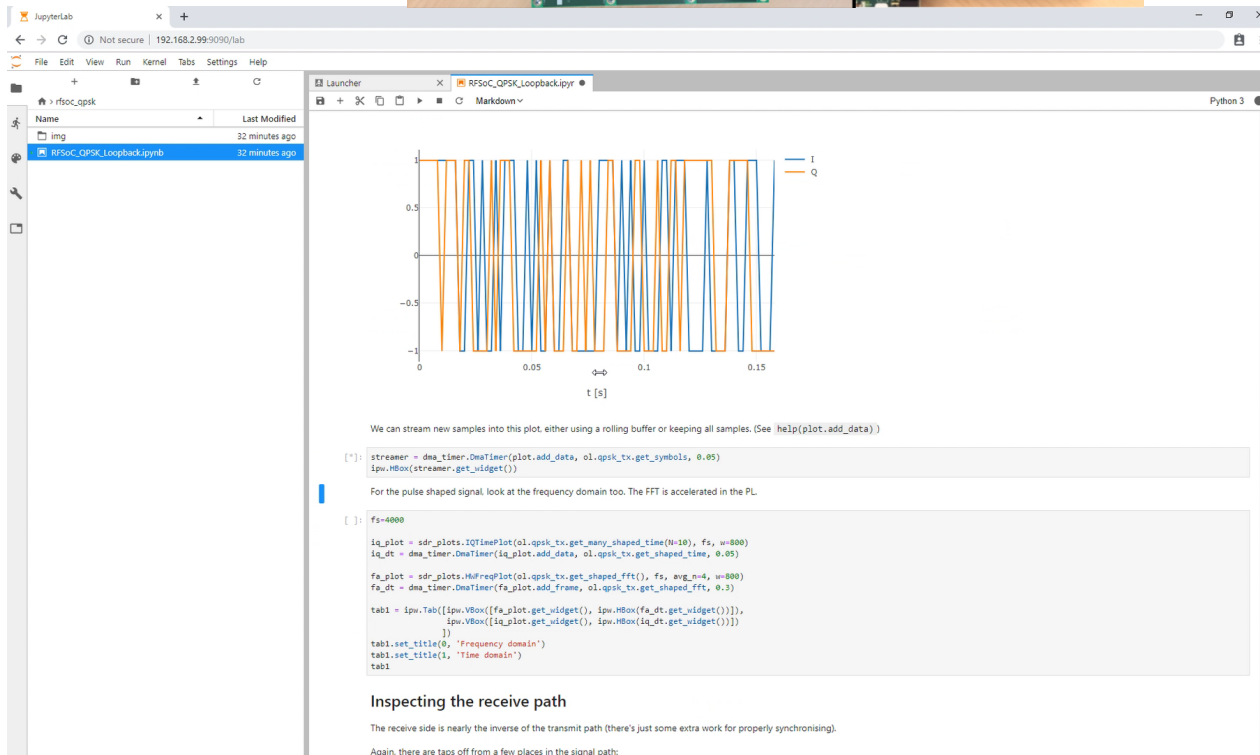
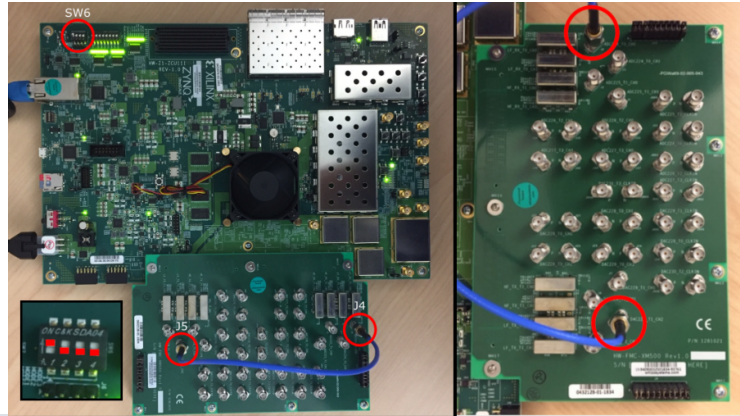
jupyter
JupyterLab



- State-of-the-art BIG DATA analysis
- State-of-the-art BIG DATA interactive visualization
- Opportunities: ML and SDR
 - Cognitive radio, etc

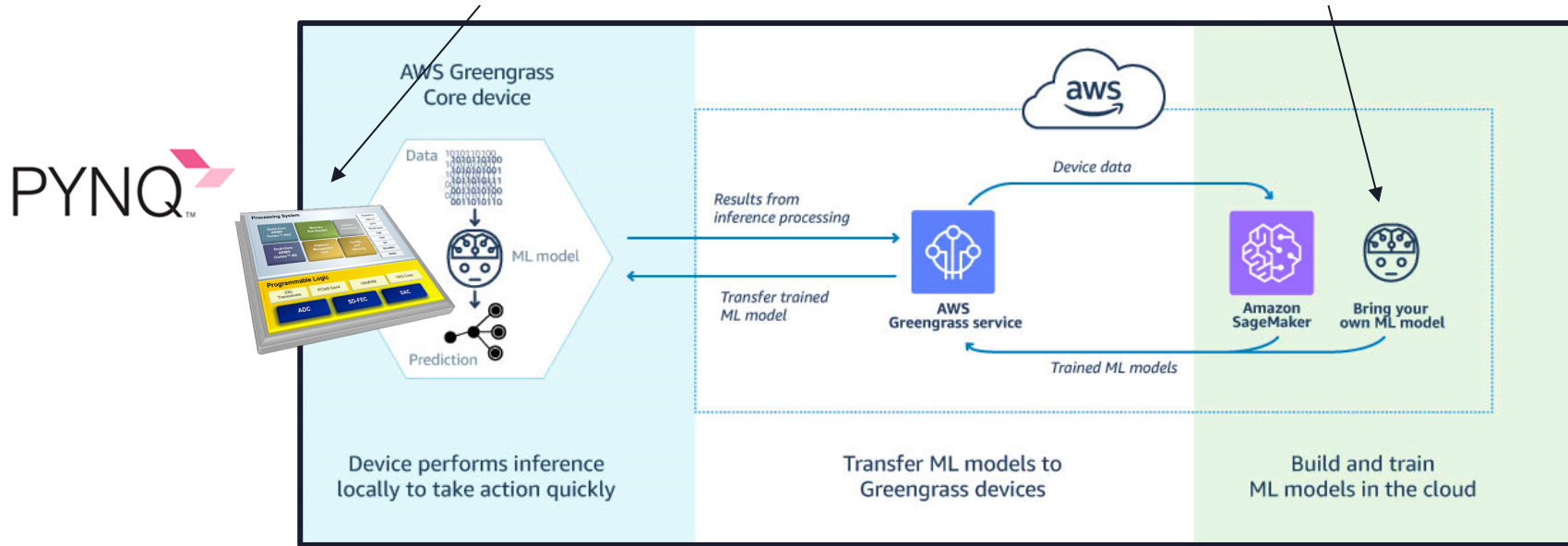
University of
Strathclyde

RF_QPSK Demo



Edge-to-cloud co-design

Common JupyterLab tooling at edge and cloud



PYNQ enables ML experts and radio engineers to focus on their 'value-add'

Edge-to-cloud co-design trade-offs:

- Maximize on-chip processing
- Minimize edge-to-cloud data exchange
- Exploit scalability of cloud processing
- Aggregate intelligence between and across multiple edge nodes
- Co-optimize the above for best system performance

Connect other Python Libraries -

pandas
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

scikit-learn
 Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

NumPy

BULLET PHYSICS LIBRARY

ROS.org

SimPy
 Discrete event simulation for Python

data analysis

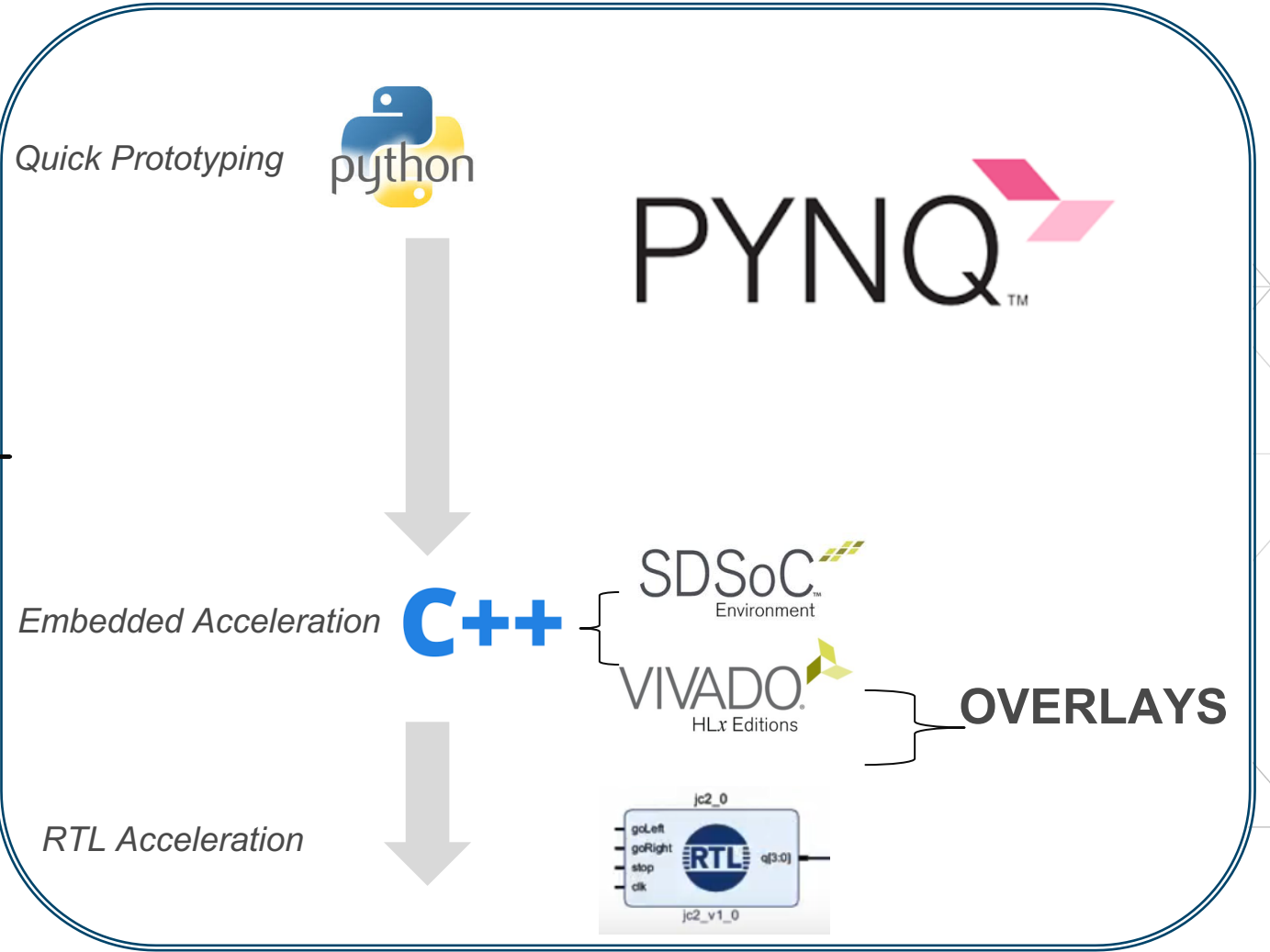
Statistical machine Learning

data analysis

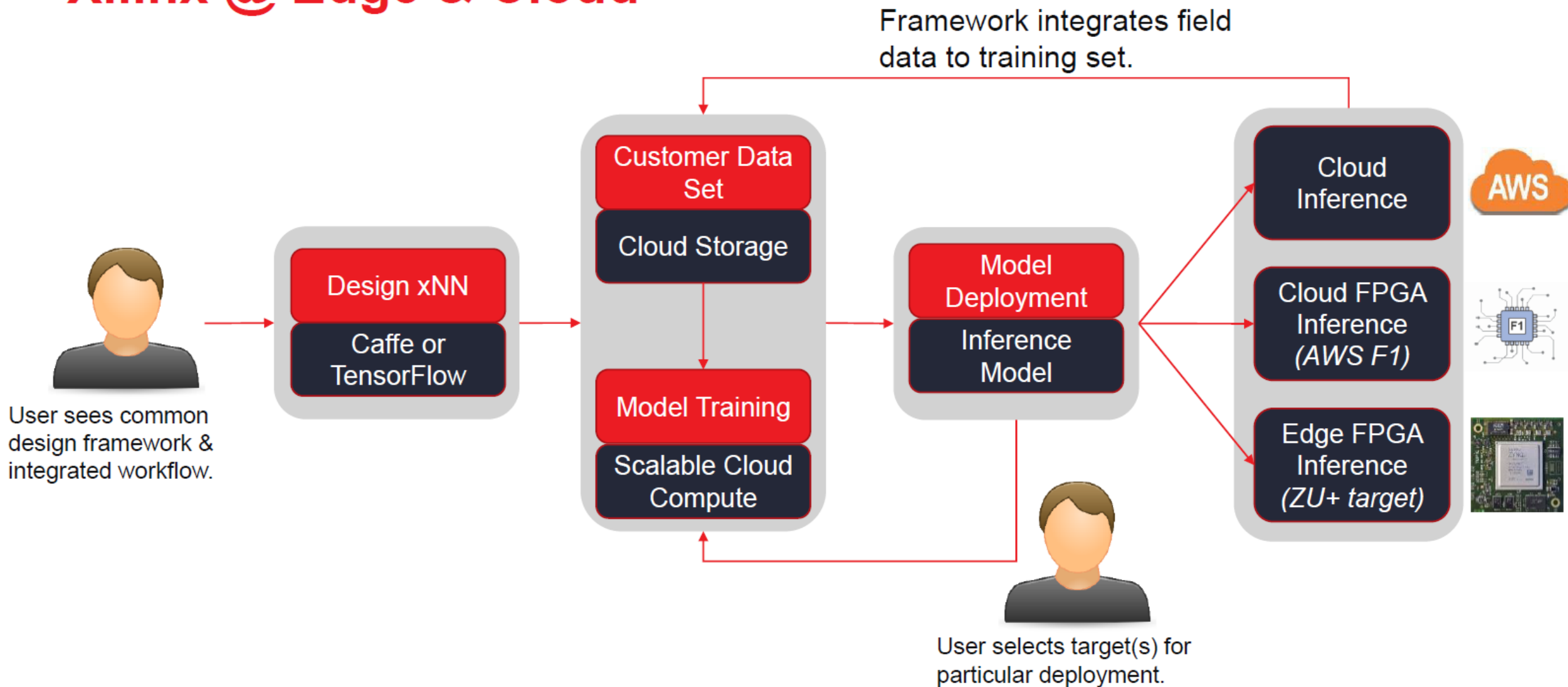
sensors

robots

discrete event simulation



AWS + Xilinx - SageMaker ML Framework target Xilinx @ Edge & Cloud

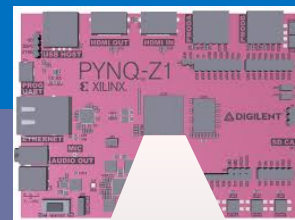


How PYNQ is bridging hardware and software

Software



Open Source



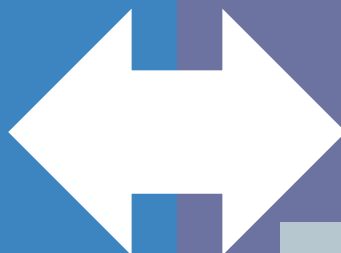
Hardware



Python stack



1001011100111111...



All programmable hardware

Python software libraries



Pre-optimized hw overlays



Efficient porting PYNQ to any Zynq-based platform

3

Target-specific PYNQ components

Board-specific porting
Common interfaces available for re-use

2

PYNQ Software Core
(board independent)

Pre-built images available
Expected to take less than 1 day to port

1

PetaLinux build tools/BSP

Available for any Xilinx-supported board;
Instructions for building for custom boards available

ALINX[®]

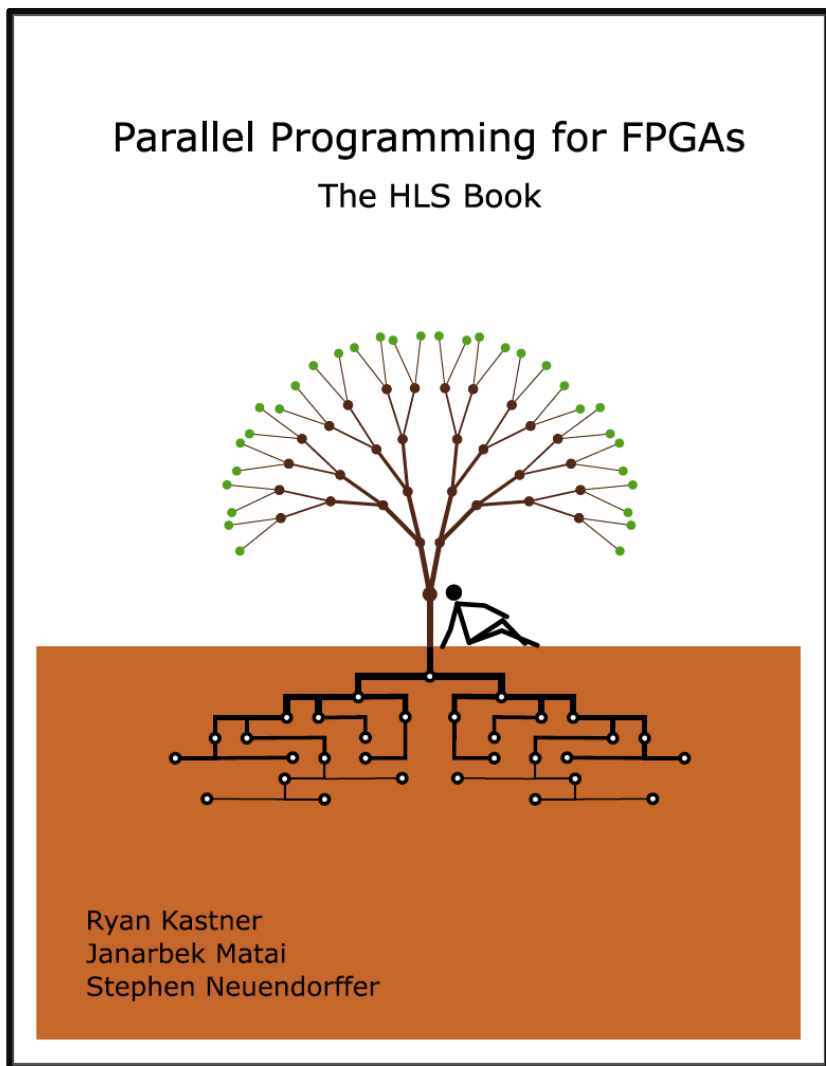
MYiR

威视锐[®]
V3 Technology, Ltd.

Actions



Open source HLS book



Parallel Programming for FPGAs is an open-source book aimed at teaching hardware and software developers how to efficiently program FPGAs using high-level synthesis



公众号回复 pp4fpgas

CECA@PKU FPGA Accelerator

PYNQ & HLS | 跟着北大CECA学FPGA加速器设计

Pooterko Xilinx学术合作 1 week ago

本文的目标是帮助对于深度学习硬件加速器设计感兴趣的朋友快速上手学习加速器设计。

准备

以下是阅读本文的基础，请做好下列基础准备后再上手加速器设计：

1. C语言设计：熟练掌握C语言语法。
2. 计算机体系结构知识：参考书《计算机组成与设计》，不需要熟读全书，但需要了解计算机体系结构的基础概念有比较清晰的理解和认识，如流水线、数据并行等。

高层次综合

利用高层次综合工具，开发者只需要编写高级语言的代码完成程序功能，开发工具会自动将代码综合成相同功能的RTL级实现(基于Verilog或VHDL)。开发者可以通过一些pragma的方式来指示和调整高层次综合工具生成的硬件模块的架构。高层次综合工具进行FPGA硬件开发的过程，应该是利用软件语言的表达编程。目前，高层次综合的代码都是基于C/C++/OpenCL的，所以对于很多朋友来说，利用高层次综合工具可以大幅度地降低学习难度，缩短开发迭代速度。

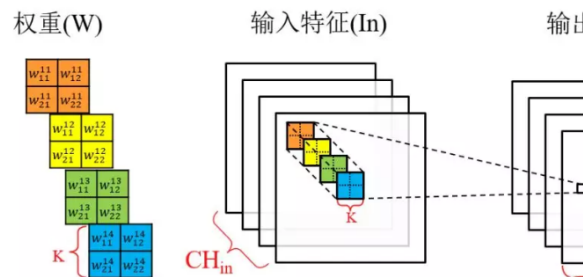
3天入门实例

我们需要使用一个简单的实例来进行入门学习。既然目标是让大家了解硬件设计，那么我们的实战示例就选择使用目前最火爆、最具代表性——卷积神经网络(CNN)。

我们选取卷积神经网络前向计算中耗时最长的卷积操作作为我们本次的实例，需完成以下步骤：

1. 准备工作(1天) 下载Xilinx Vivado HLS或Xilinx SDx工具链，利用软件工具的使用，包括：新建工程、配置工程参数、综合流程等。

2. 软件实现(1天) 实现卷积层的软件版本(C语言版本)，并封装成一个函数，结合高层次综合工具的report和analyze工具分析理解所生成的卷积运算的流程如下图所示：



在HLS工具中重新综合，发现延迟降到了仅仅**1782**个时钟周期(具体数字可能略有差异)，相对于原始无优化版本的加速达到了**141.06**倍！HLS的综合报告里显示加速器调用也已经完全流水化了：

Loop Name	Latency		Iteration Latency	Initiation Interval		Trip Count	Pipelined
	min	max		achieved	target		
- Kernel_Row_Row_Column	1782	1782	20	1	1	1764	yes

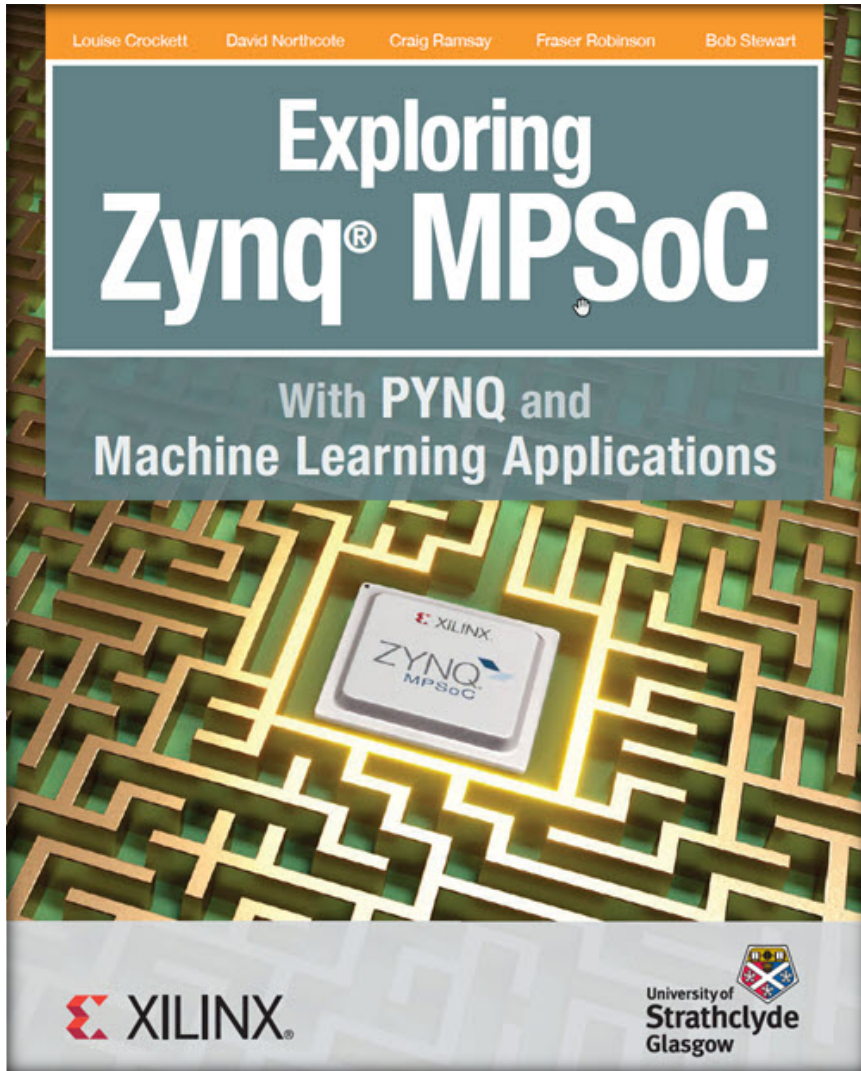
综上所述，我们就完成了一个高效的卷积运算加速器，而基本上利用HLS工具设计FPGA硬件加速器的入门也就完成了。总结来说，使用HLS设计FPGA加速器的一般化设计流程如下：

1. 熟悉并理解目标算法，通过软件运行目标算法，分析性能瓶颈所在；
2. 实现加速目标的软件版本，分析其中可以并行、流水化的部分，并构想可行的硬件架构；
3. 通过代码重构，加pragma等方法在HLS工具中描述目标架构，此过程需注意保证改写的代码功能性与原代码严格保持一致；
4. 调整硬件参数配置，最大化利用硬件资源(计算资源如DSP、存储资源如BRAM)来最大化加速器的性能。

30天精通学习

在完成了上面的3天入门实例后，大家可以进一步学习和实践FPGA加速器的设计，这一部分我们推荐大家利用3到4周的时间对相关知识进行详细、系统的学习。高层次综合的相关知识我们推荐大家可以参考公司推出的《FPGA加速器的设计》一书。

Open source MPSoC book



The book covers the architecture of the device, the design tools and methods, conventional hardware/software co-design approach, and the newer software-defined methodology, as well as hardware and software development, multiprocessing, safety, security and platform management, system booting, PYNQ and machine learning applications



公众号回复 **mpsocbook**

Curriculums All in One (Pynq-Z2)

Digital Logic

Computer Architecture

Image and Video Processing

Machine Learning

FEC Coding

Hardware Software Co-Design

CAD Tools

High Performance Computing

Configurable Computing

Robotics

Speech Recognition

On Chip Networks

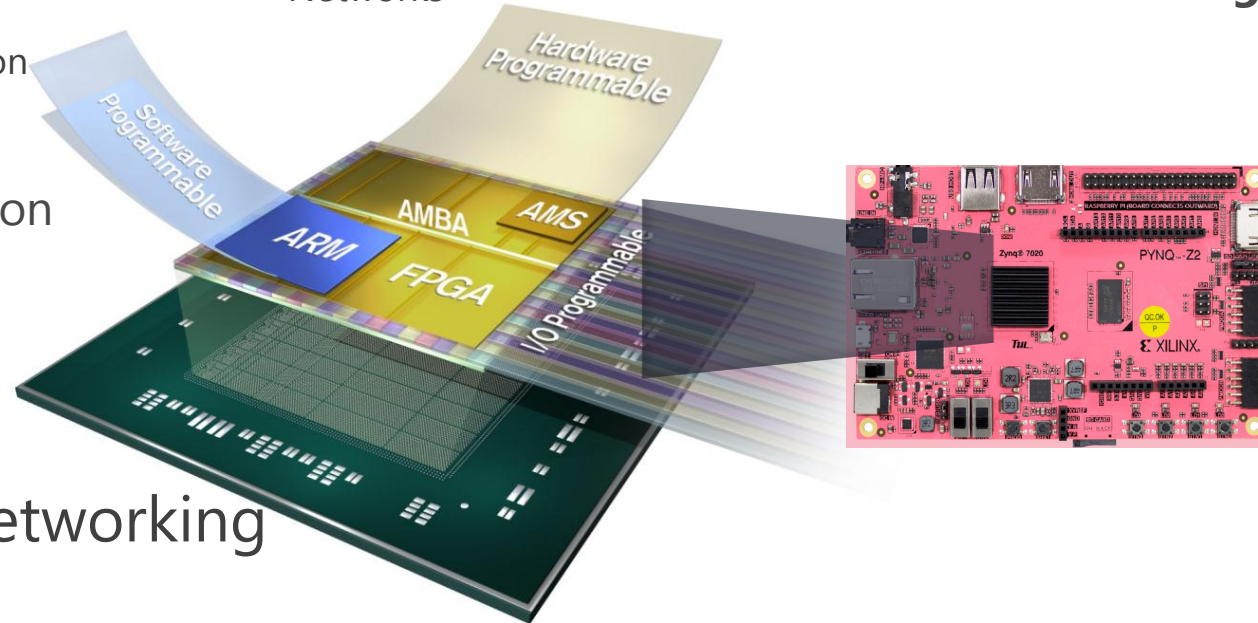
Encryption

Dynamically Reconfigurable Systems

Networking

Digital Signal Processing

Embedded Systems



Ideal for Teaching and Research

Adaptable.
Intelligent.



赛灵思工业物联网研讨会
XILINX IIoT SEMINAR

 XILINX[®]