

Conversation with Xilinx Research Labs

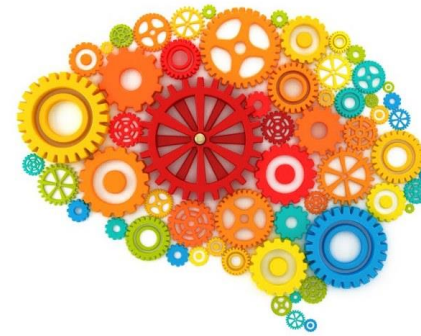
CTO Organization
October 2018



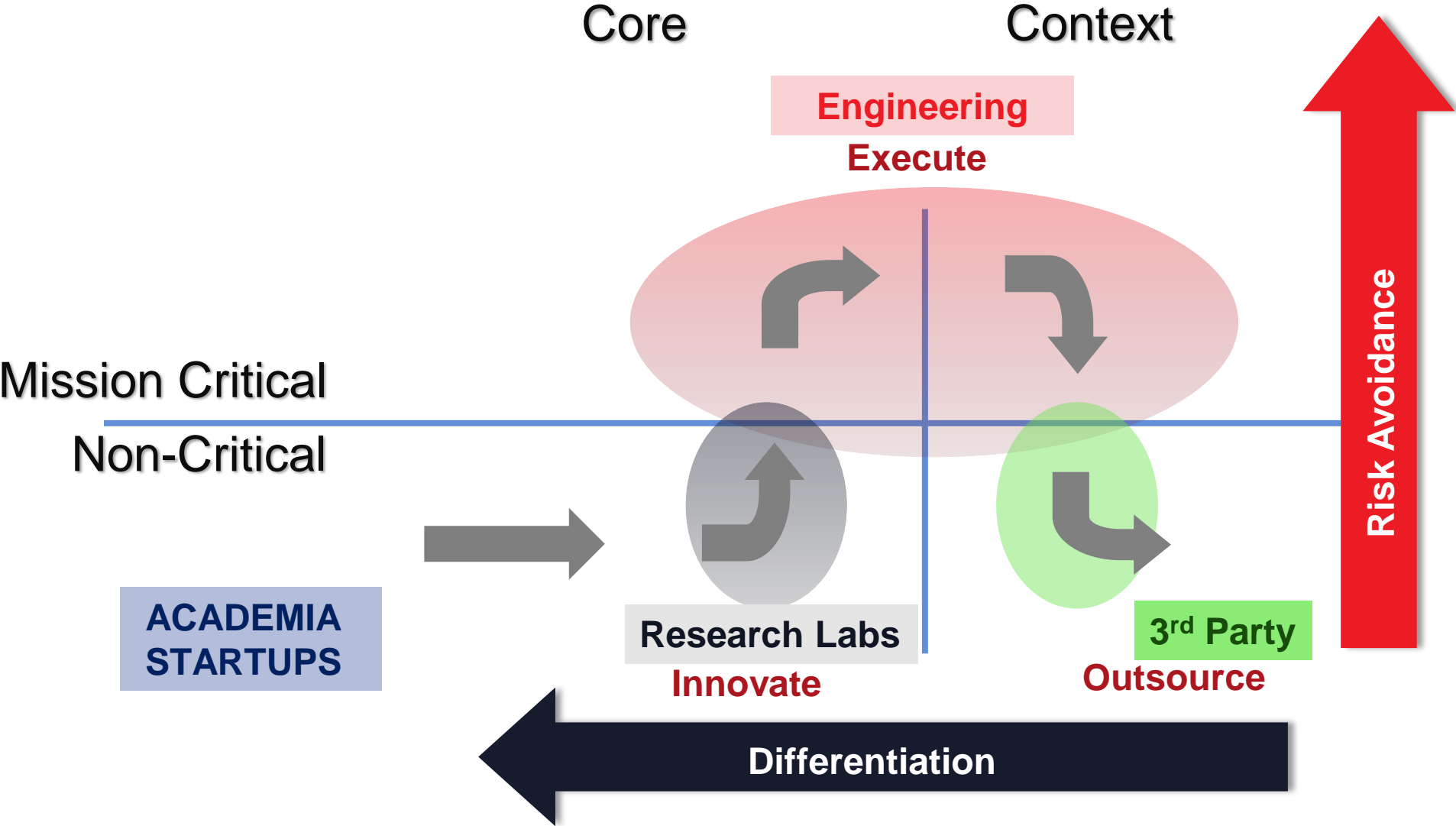
Xilinx Research Mission

Drive technology innovation to increase growth and value of Xilinx

- Enable New Users
- Open New Markets
- Create New Value
- Win Mindshare of Innovators



Research : Pathfinding & Differentiation



Xilinx University Program Mission

Empower academic teaching, research, and entrepreneurship with Xilinx technologies

- > Provide support for teaching, training professors, workshops, hackathons
- > Give students access to our latest technology
- > Enable new business and technical opportunities through research partnerships



Our Corporate Mission

Building the Adaptable
Intelligent World



Our Strategy



Focus on
Data
Center
Opportunity

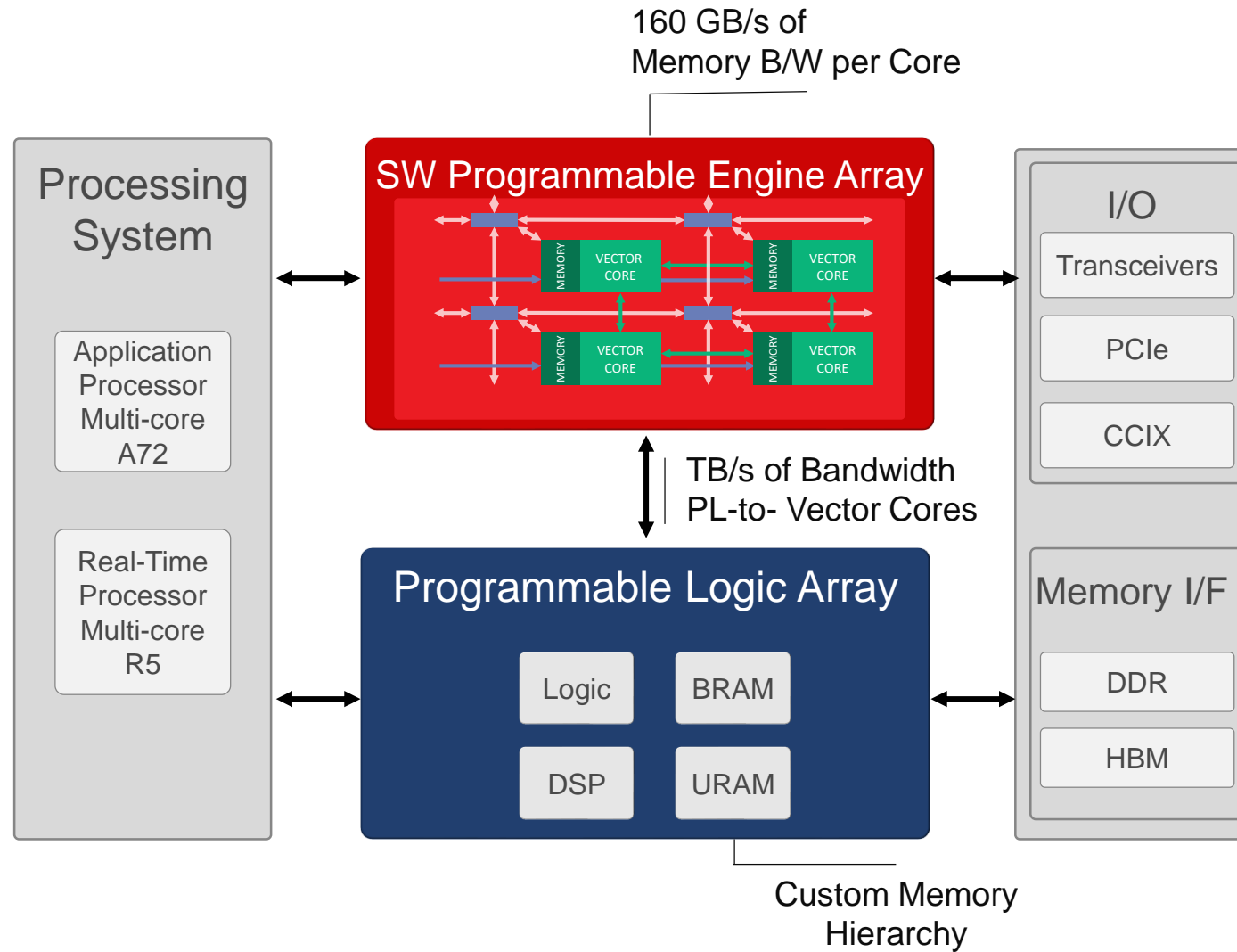


Accelerate
Core
Markets
Growth



Build
Adaptive
Computing
Platform

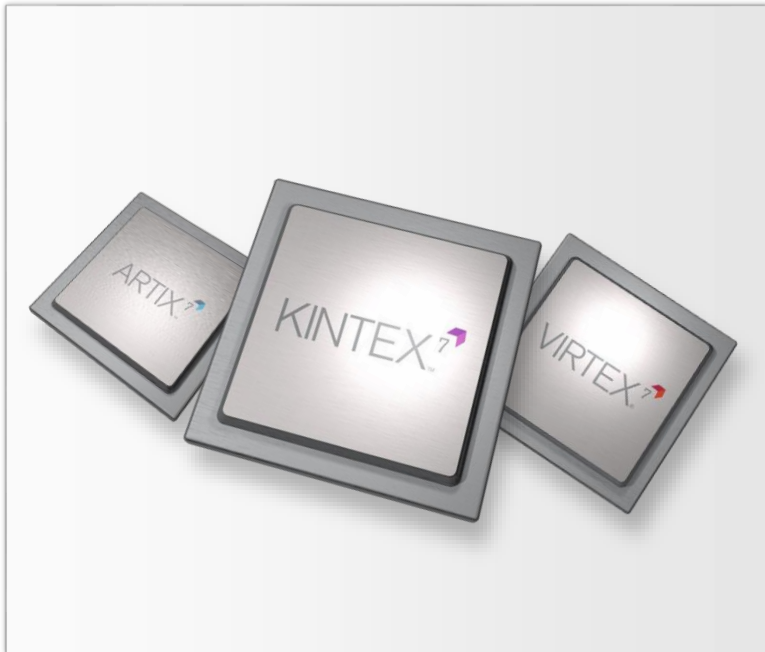
Adaptive Computing



Adaptive Compute Acceleration Platform (ACAP)

Data Center

Devices



Production Boards Compute, Networking, Storage



FPGA as a Service (FaaS)



Enabling Software Developers

Open Frameworks



Software
Application
Developers

Accelerated
Libraries

Machine
learning



Database
analytics



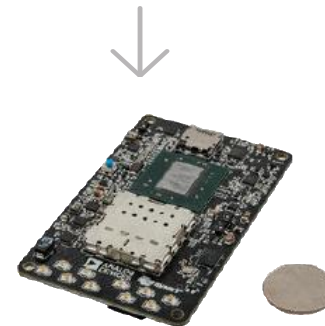
Development
Stack

Development
Environment



System
Developers

Development
Boards



Discussion Topics

- > PYNQ : Python Productivity on Zynq
- > FINN : Software framework for reduced precision Neural Networks
- > Programming SmartNIC using the P4 language and NetFPGA
- > RapidWright : A framework for fast and efficient implementation of modular design
- > Engaging with Xilinx University Program



Python Productivity for Zynq

Presented By

Patrick Lysaght
Senior Director
1st Oct 2018



Overview

- > More productivity
- > Enabling technologies
- > Open source
- > PYNQ
- > Next steps
- > Research opportunities

PYNQ™  

PYNQ™ Python Productivity on Zynq

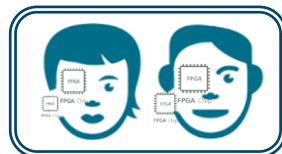


New users are not always hardware designers,
or embedded systems designers



PYNQ™

*Enable more people to program Xilinx
processing platforms, more productively*



AND

*Offers more rapid development for h/w designers
and embedded s/w engineers*

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>

Standard Python comes with comprehensive libraries but also has a huge external ecosystem

152,480 projects 1,079,748 releases 1,477,547 files 263,910 users

python™
Package Index

The Python Package Index (PyPI) is a repository of software for the Python programming language.

PyPI helps you find and install software developed and shared by the Python community. [Learn about installing packages.](#)

Package authors use PyPI to distribute their software. [Learn how to package your Python code for PyPI.](#)

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 Jupyter Notebook is open, showing Python code for image processing with Darknet. The code includes comments and function calls like `lib.forward_region_layer_pointer_nolayer`. Below the code, the notebook output shows a list of detected classes: car (86%), dog (86%), and bicycle (78%). A small image of a bicycle is shown with bounding boxes around it. In the center, a code editor window shows Python code for an inference function, with a red arrow pointing to a comment: `""" Load input image into accelerator memory """`. On the right, a terminal window shows the command `tree` output, listing files and directories like `base`, `arduino`, `data`, `board`, `microblaze`, and `microblaze_python`. Below the terminal, an output view shows a state transition diagram for an FSM with states S0/000, S1/001, S2/011, S3/010, S4/110, and S5/111, and a timing diagram for the FSM output bits.

Jupyter ... Julia, Python, R

Default engine of data science

2+ million GitHub notebooks

Taught to 1,000+ Berkeley students every semester

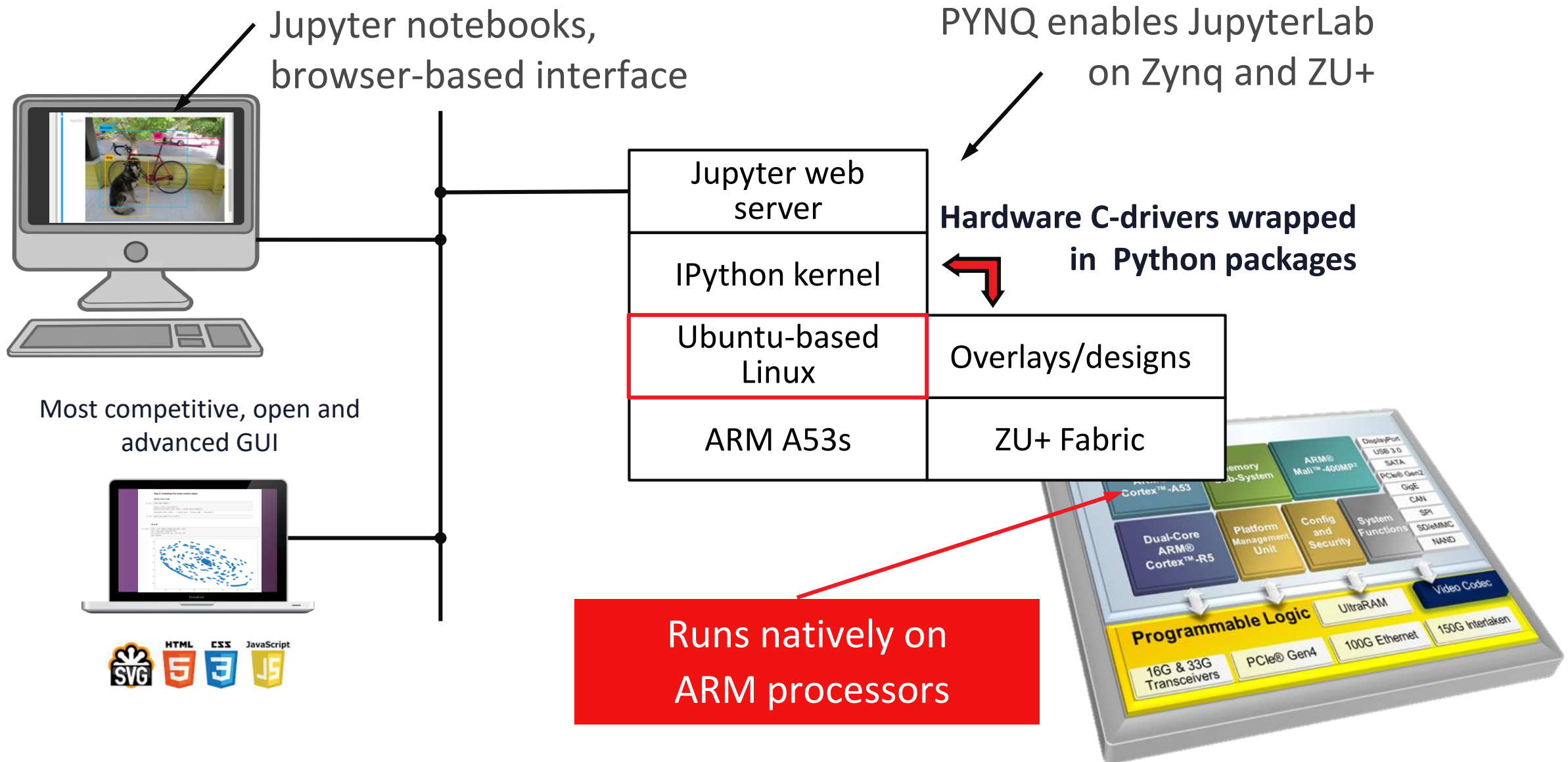
Next-gen browser IDE

Includes Jupyter Notebooks

Jupyter notebooks

Visualization

Python productivity for Zynq



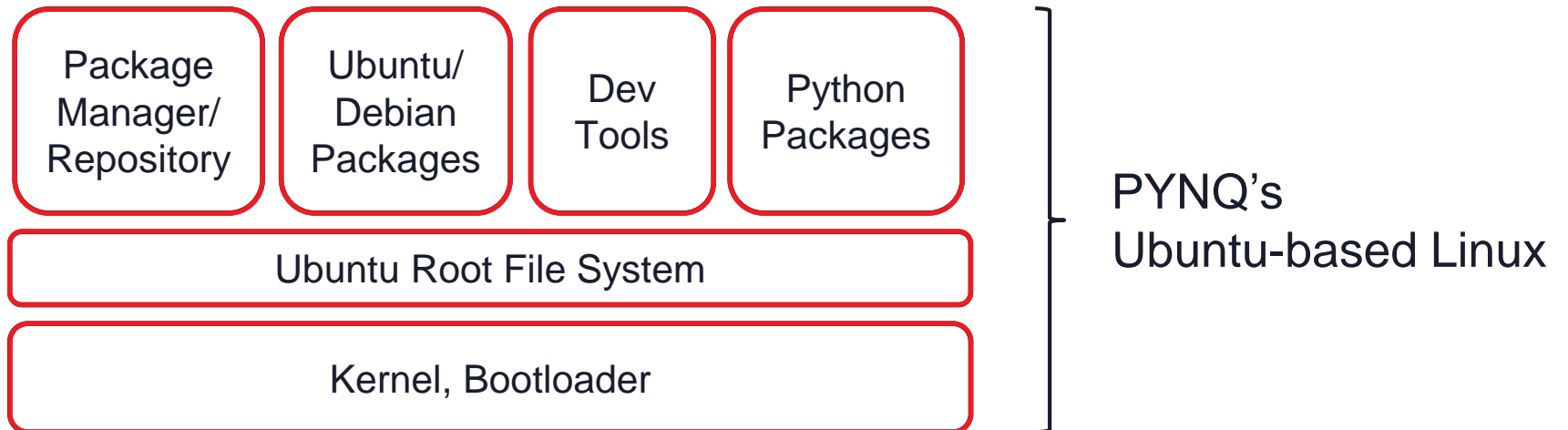
PYNQ's Ubuntu-based Linux

PYNQ uses Ubuntu's:

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

PYNQ bundles :

- Development tools
 - Cross-compilers
- Latest Python packages



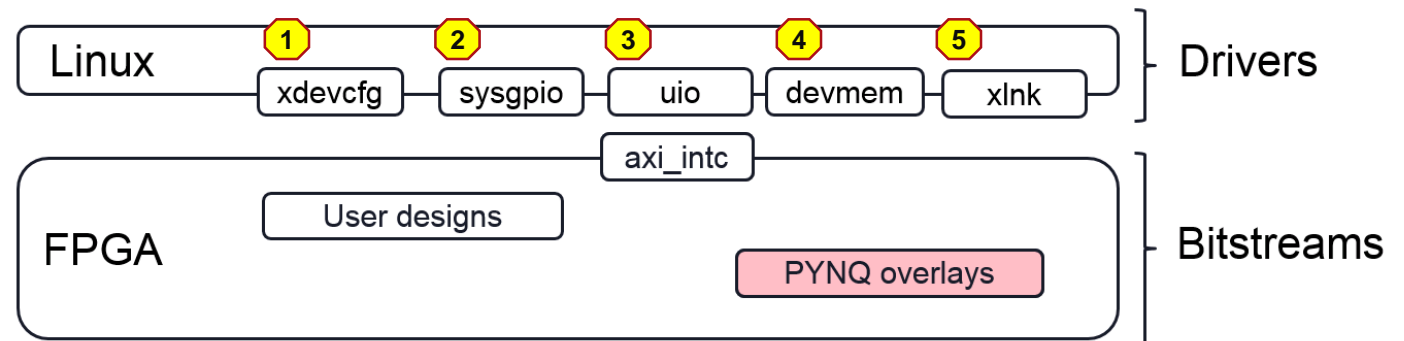
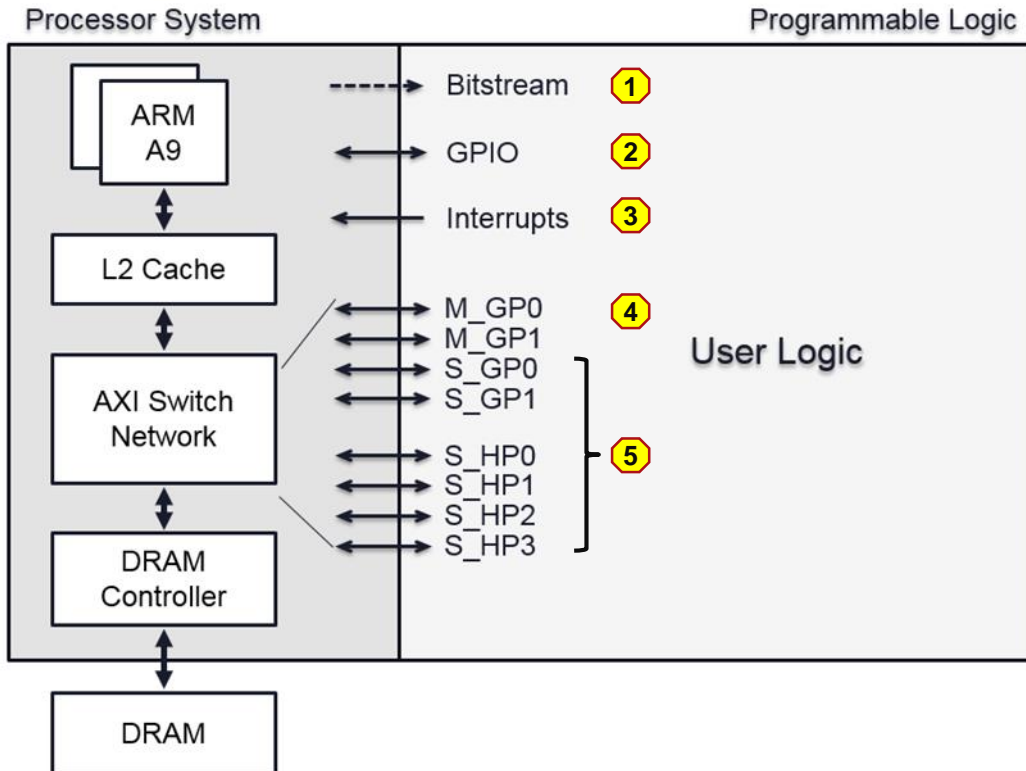
PYNQ uses the PetaLinux build flow and board support package:

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

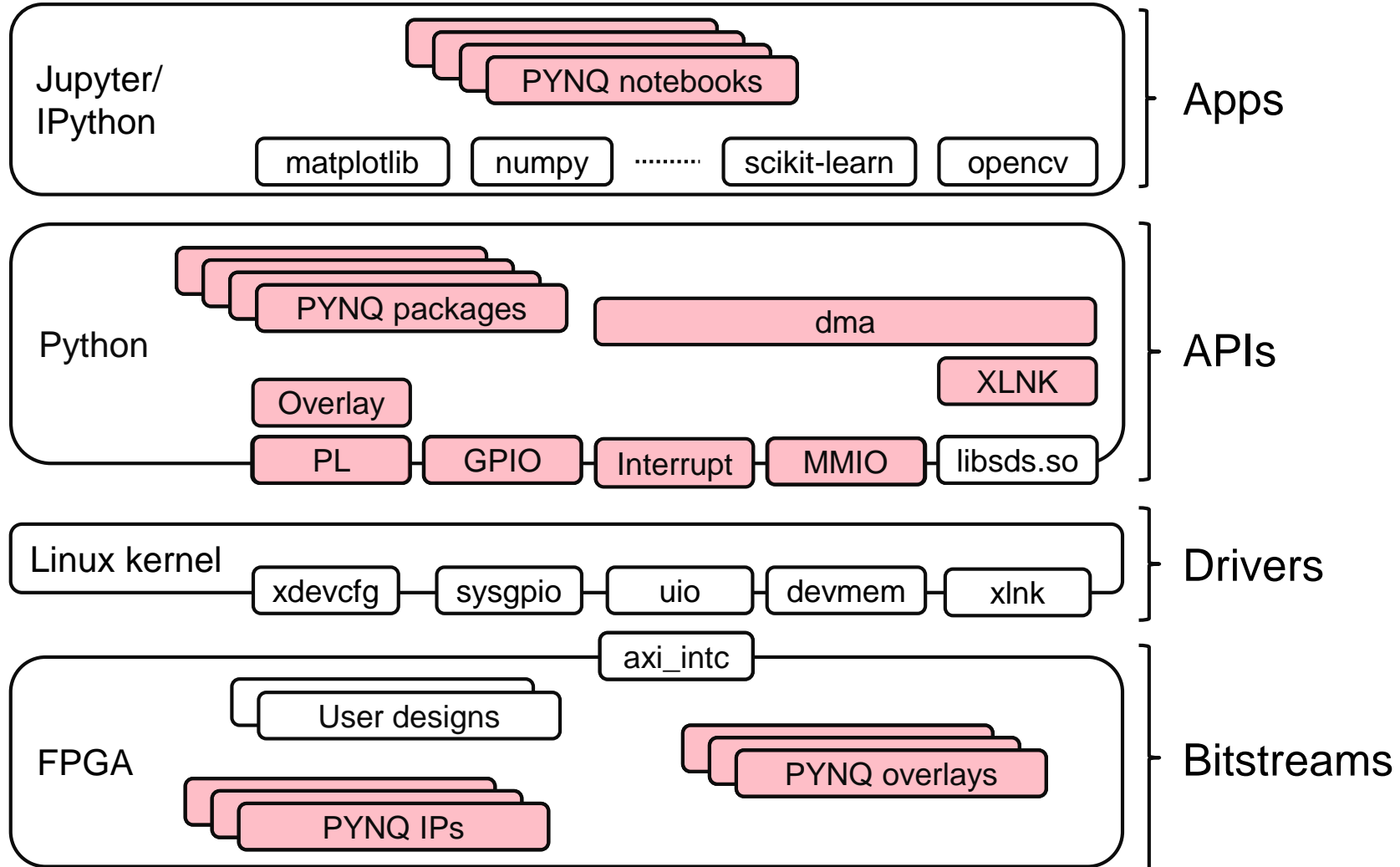
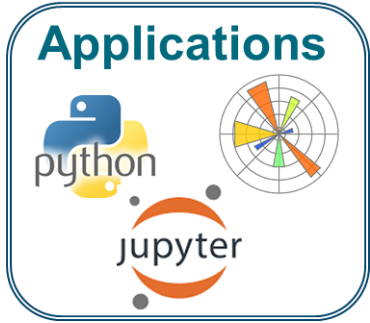
PYNQ provides Linux drivers for PS-PL interfaces ...

wrapped in Python libraries

Zynq



PYNQ™ is a Framework



Software-style packaging & distribution of designs

Enabled by new *hybrid packages*

The image displays four screenshots of GitHub repositories for Xilinx PYNQ designs, illustrating software-style packaging and distribution of designs.

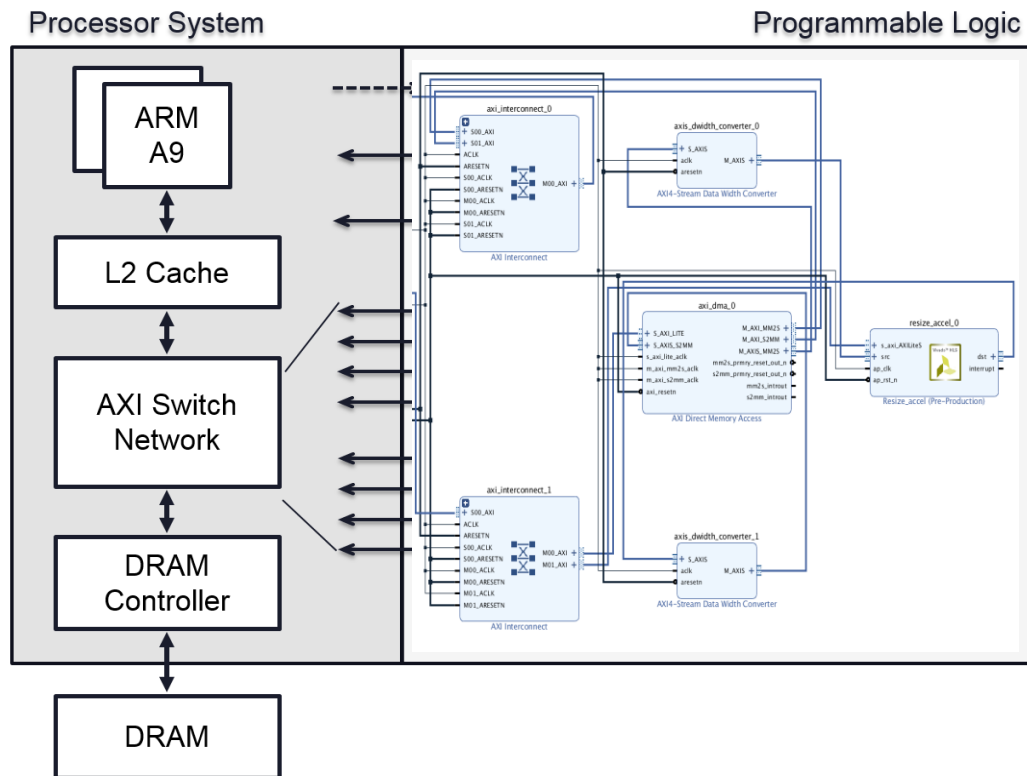
- QNN-MO-PYNQ:** Shows a notebook titled "3. Open image to be classified" with Python code for image classification and an output image of a dog.
- Bot-SPYN:** Shows a notebook titled "SPYN - III phase AC motor control" with objectives and a step-by-step guide for downloading the EDDP bitstream.
- PYNQ-DL:** Shows a notebook titled "Resizing an image" with a hardware block diagram illustrating the flow from PS (ARM) to PL (Resize IP) via AXI Interconnect and DMA.
- PYNQ-ComputerVision:** Shows a notebook titled "OpenCV Overlay: Filter2D and Dilate" with a list of steps for setting up the overlay and running the program.

Download a design from GitHub with a single Python command:

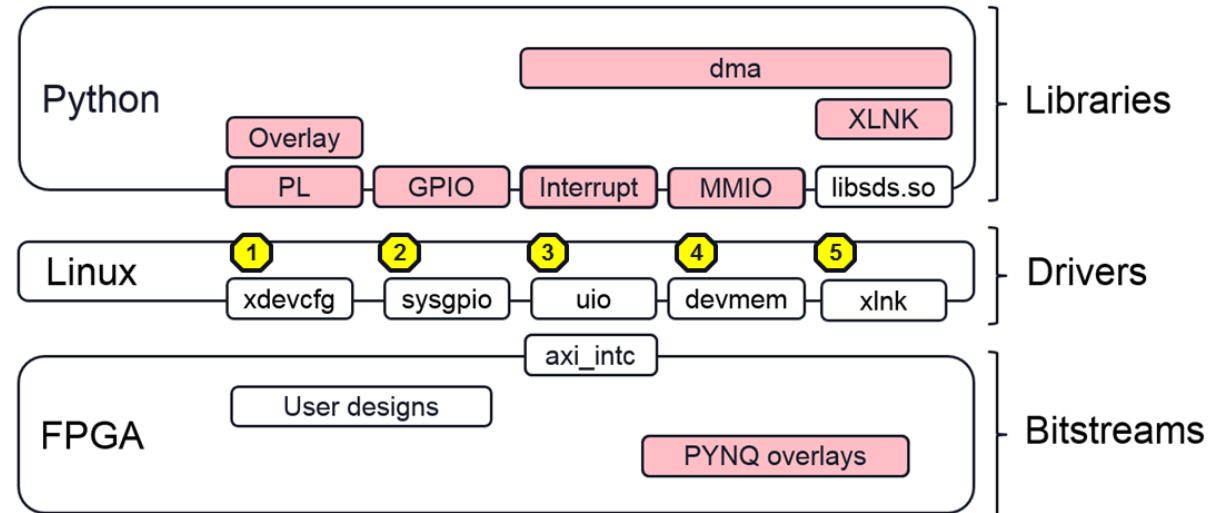
```
pip install git+https://github.com/Xilinx/pynqDL.git
```

Load the downloaded resizer design into Zynq

Zynq



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



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

Notebook Examples

Display the image to be resized

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



Software only re-sizing

Original image size

```
In [5]: old_width, old_height = original_image.size
print("Image size: {}x{} pixels.".format(old_width, old_height))
Image size: 640x360 pixels.
```

Resizing

Setting image resize dimensions

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

Using resize() method from the PIL library

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

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

Display resized image

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



Resized image size

```
In [9]: width, height = resized_image.size
print("Resized image size: {}x{} pixels.".format(width, height))
Resized image size: 320x180 pixels.
```

Display the image in buffer

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

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

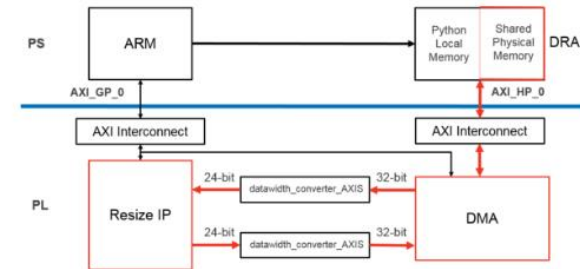


Hardware accelerated re-sizing

Image size: 640x360 pixels.

Run the Resizer IP

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



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

```
def run_kernel():
    dma.sendchannel.transfer(in_buffer)
    dma.recvchannel.transfer(out_buffer)
    resizer.write(0x00, 0x01) # start
    dma.sendchannel.wait()
    dma.recvchannel.wait()
```

```
run_kernel()
```

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



Resized in Hardware(PL): 320x180 pixels.

Activity Snapshots

DAC Contest



2018 DAC System Design Contest on Low Power Object Detection



PYNQ-Z1 Zynq 7020 Performance				
Team Name	IoU	Power (mW)	FPS	TS
TGIIF	0.623798	4200	11.9553	1.267469397
SystemsETHZ	0.491926	2450	25.9678	1.179449366
iSmartZ	0.5733	2590	7.3488	1.163665782

Nvidia TX2 Pascal Performance				
Team Name	IoU	Power (mW)	FPS	TS
ICT-CAS	0.6975	12577	24.55	1.373729
DeepZ	0.6911	13271	25.30	1.359707
SDU-legend	0.6847	10308	23.64	1.358105



Community projects

spoonNN
ETH Zurich
FPGA-based neural network inference project

Video processing
KU Leuven
Hardware accelerated video processing

SPYN
Xilinx ISM, Trenz electronics
Industrial motor control

BNN
NTNU, University Sydney, Xilinx labs
Binarized neural network

inference for DAC 2018 contest

ZipML-PYNQ
ETH Zurich
Hardware accelerated compression

PYNQ networking
Xilinx labs
Overlay with network analysis capability

Video filters with PR
Xilinx labs
Video filtering with partial reconfiguration

FIR filter example
CU Boulder
Example of integrating a FIR filter

contest for neural network object detection

PYNQ bot
IT Teilight
Control of robotic car from PYNQ

QNN
Xilinx labs
Quantised neural network

PYNQ computer vision
Xilinx labs
Build a vision processing pipeline from xOpenCV

CNN on PYNQ
Imperial College London

Accelerated OpenCV image filtering library

PYNQ LED cube
Fudan University, Xilinx China
Controlling an LED cube from PYNQ

LSTM
TU Kaiserslautern
Quantized LSTM on PYNQ

GZip on PYNQ
University Bucharest
GZip compression with DEFLATE-compatible data, and fixed Huffman coding

VectorBlox
HDMI Video processing

Next steps: scaling across Platforms and Domains

ISM ML

IIoT vision

SDR

.....

Software

Software

Software

.....

Zynq

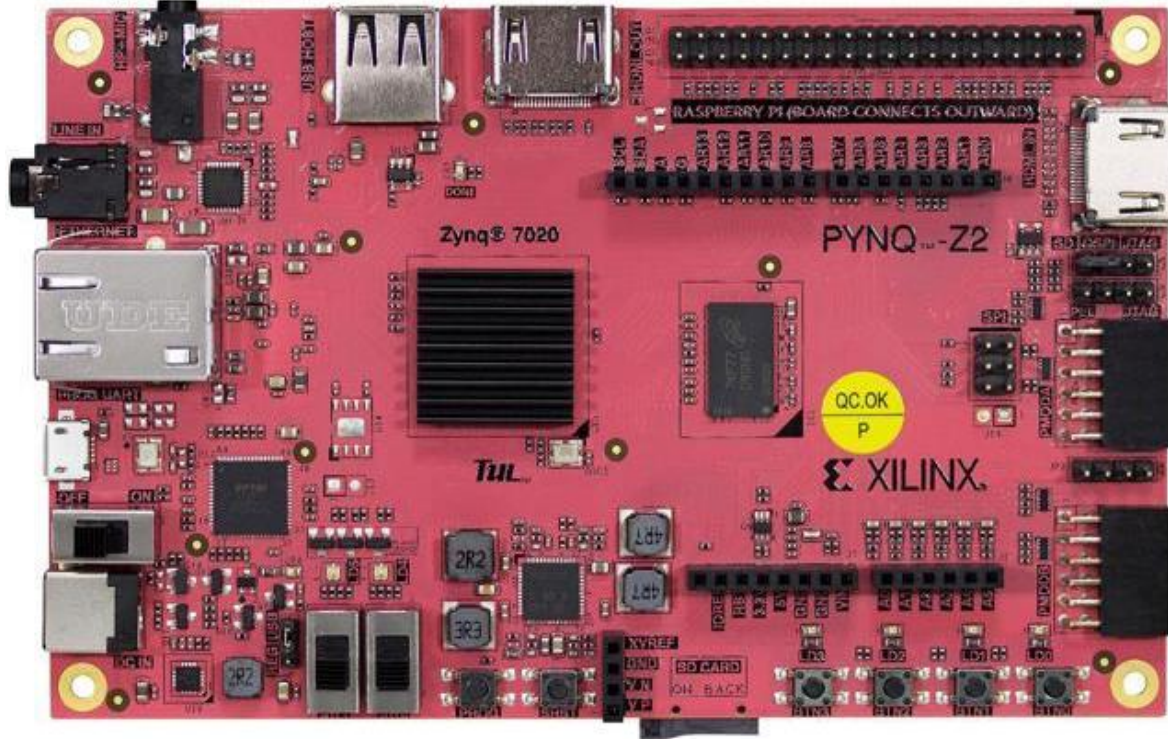
ZynqU+

RFSoc

.....



New PYNQ-Z2 Board available now

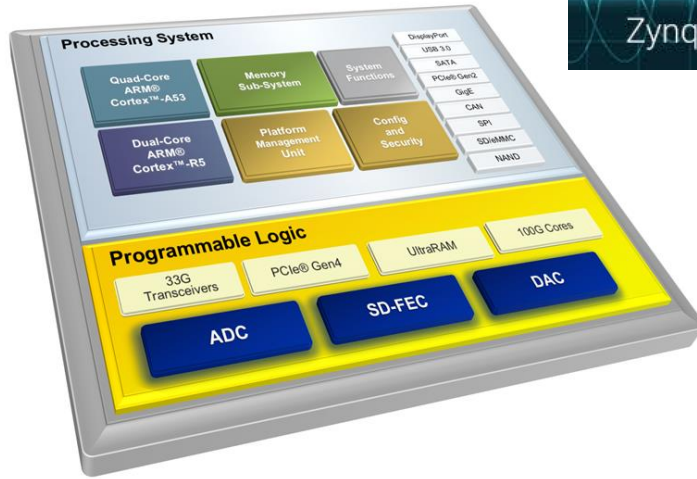


\$119 to everyone in US

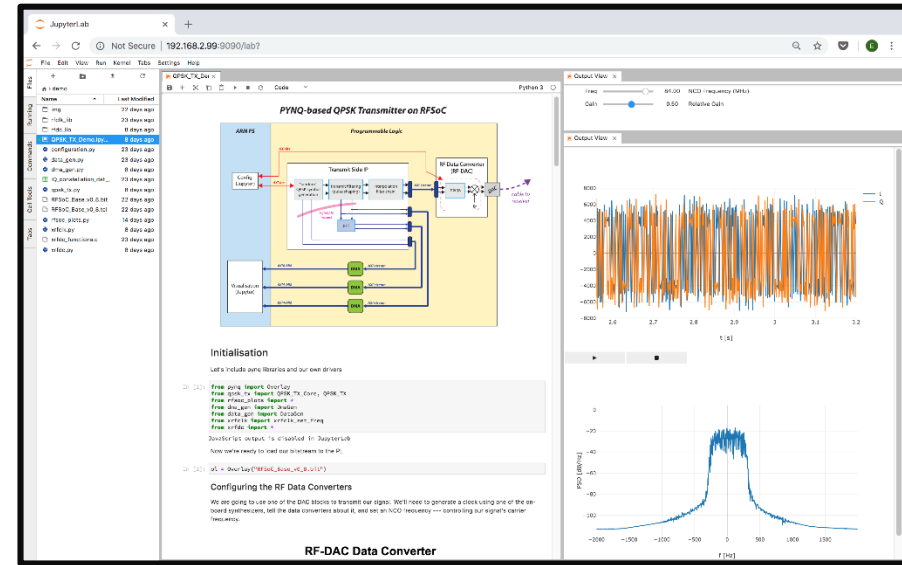
- New PYNQ reference platform
- New stereo audio with on-board codec
- New Raspberry Pi connector
- Open source design
- Manufactured by TUL in Taiwan
- Distributed by Newark & Newegg
- Academic discounts & donations available

New Research Opportunities: RFSoc and JupyterLab

RFSoc



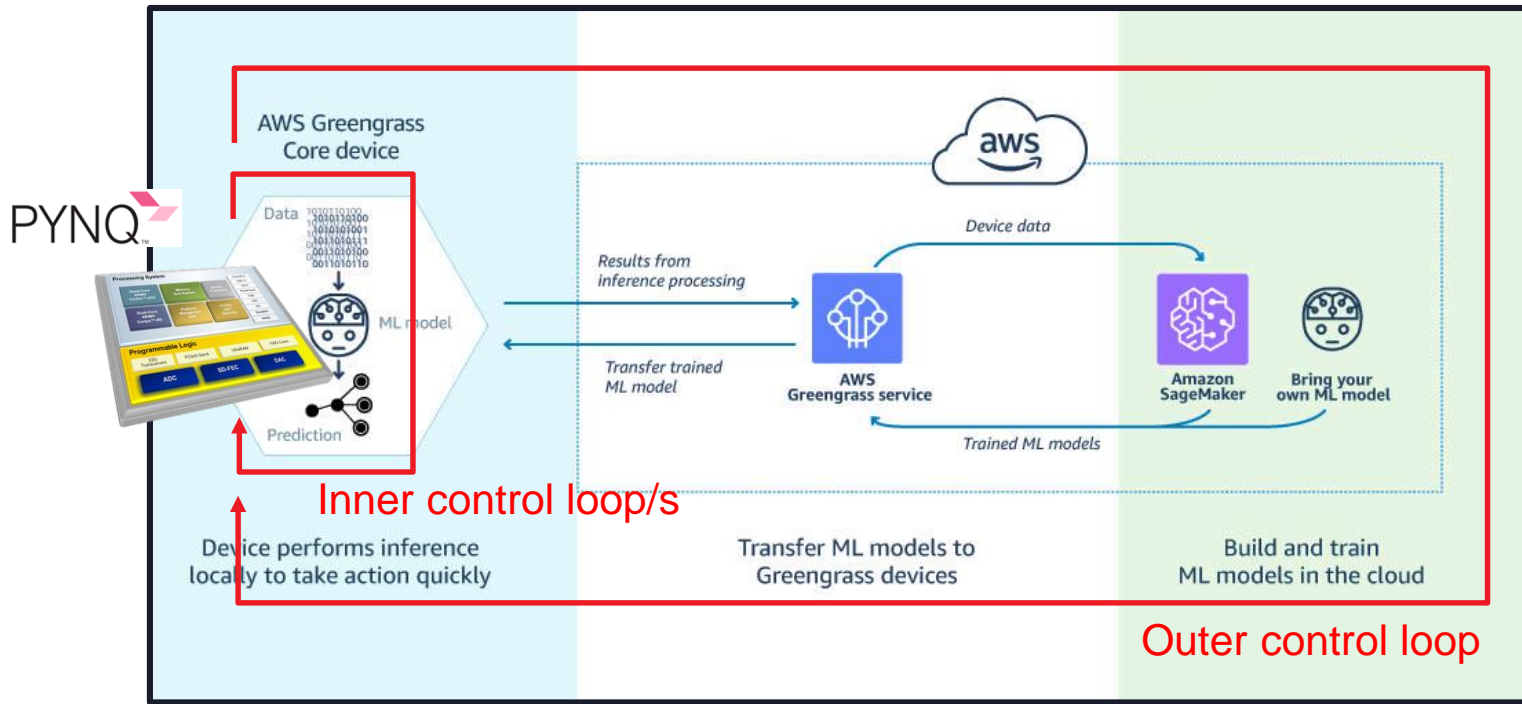
JupyterLab



- State-of-the-art BIG DATA analysis
- State-of-the-art BIG DATA interactive visualization
- Opportunities: ML and SDR
 - Cognitive & agile radios



Edge-to-cloud SDR with Machine Learning



Inner loop/s:

- Real-time control
- Localized ML
- Local communication between nodes

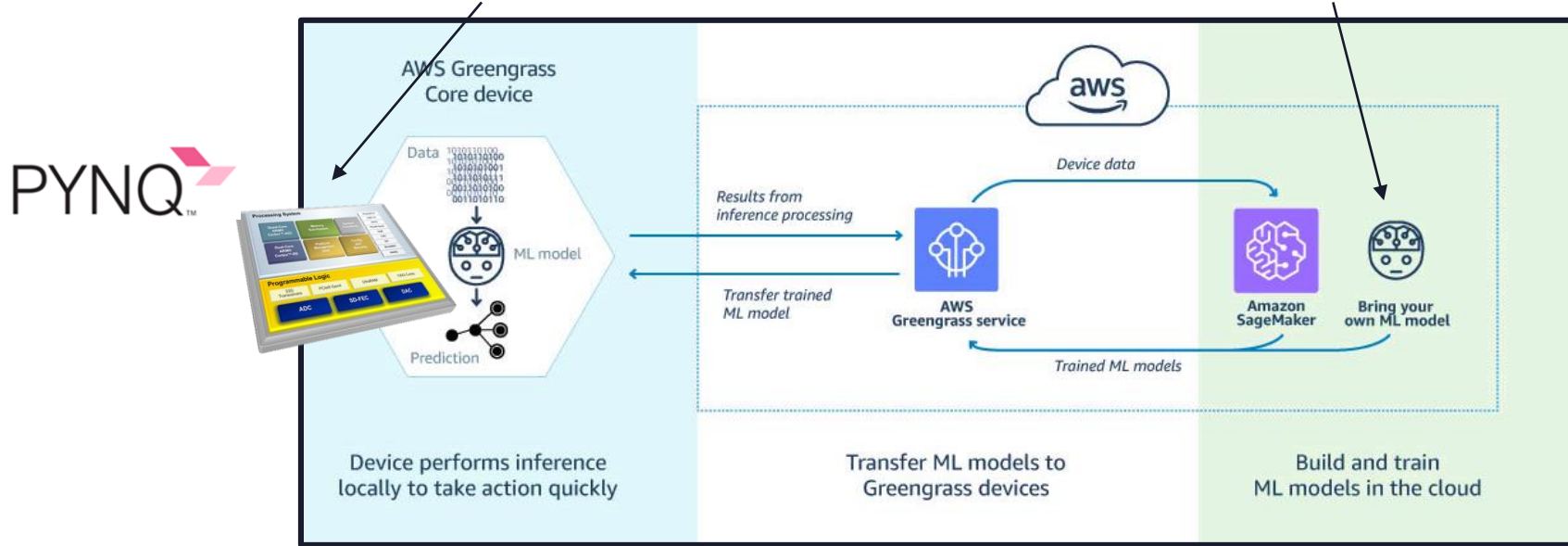
Outer loop:

- Heavy duty ML
- Aggregated across edge nodes
- Longer timescale ML

'No future SDR will be complete without machine learning'

Edge-to-cloud Co-design Opportunities

Common JupyterLab tools at edge and cloud

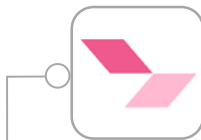


PYNQ enables ML experts and radio engineers to focus on their 'value-add'

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

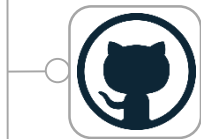
PYNQ™



pynq.io



pynq.readthedocs.org



github.com/Xilinx/PYNQ



tul.com.tw/ProductsPYNQ-Z2.html



pynq.io/support

PYNQ: PYTHON PRODUCTIVITY FOR ZYNQ

[Home](#) [Get Started](#) [Boards](#) [Community](#) [Source Code](#) [Support](#)

What is PYNQ?

PYNQ is an open-source project from Xilinx® that makes it easy to design embedded systems with Xilinx Zynq® Systems on Chips (SoCs).

Using the Python language and libraries, designers can exploit the benefits of programmable logic and microprocessors in Zynq to build more capable and exciting embedded systems. PYNQ users can now create high performance embedded applications with

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



Who is PYNQ for?

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

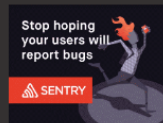
- Software developers who want to take advantage of the capabilities of without having to use ASIC-style design tools to design hardware.
- System architects who want an easy software interface and framework of their Zynq design.
- Hardware designers who want their designs to be used by the widest p

Python productivity for Zynq (Pynq)

latest

Search docs

- Getting Started
- Jupyter Notebooks
- Python Environment
- PYNQ Overlays
- PYNQ Libraries
- Overlay Design Methodology
- PYNQ SD Card
- pynq Package
- Verification
- Frequently Asked Questions (FAQs)
- Glossary
- Useful Links
- Appendix
- Change Log



Cut resolution time for Python errors from 5 hours to 5 minutes with Sentry.

[Docs](#) » [PYNQ Introduction](#)

[Edit on GitHub](#)

PYNQ Introduction

Xilinx® makes Zynq® and Zynq Ultrascale+™ devices, a class of programmable System on Chip (SoC) which integrates a multi-core processor (Dual-core ARM® Cortex®-A9 or Quad-core ARM® Cortex®-A53) and a Field Programmable Gate Array (FPGA) into a single integrated circuit. FPGA, or programmable logic, and microprocessors are complementary technologies for embedded systems. Each meets distinct requirements for embedded systems that the other cannot perform as well.

Project Goals

The main goal of PYNQ, Python Productivity for Zynq, is to make it easier for designers of embedded systems to exploit the unique benefits of Xilinx devices in their applications. Specifically, PYNQ enables architects, engineers and programmers who design embedded systems to use Zynq devices, without having to use ASIC-style design tools to design programmable logic circuits.

PYNQ achieves this goal in three ways:

- Programmable logic circuits are presented as hardware libraries called *overlays*. These overlays are analogous to software libraries. A software engineer can select the overlay that best matches their application. The overlay can be accessed through an application programming interface (API). Creating a new overlay still requires engineers with expertise in designing programmable logic circuits. The key difference however, is the *build once, re-use many times* paradigm. Overlays, like software libraries, are designed to be configurable and re-used as often as possible in many different applications.


New Tab x

← → ↻ 🔍 |

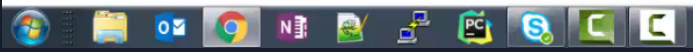
★ 🔗 🔄 📄 ☰

📁 Apps 📄 JupyterLab 📄 Xilinx/PYNQ-DL: Xili... 📄 PYNQ - Python pro... 📄 Master

Gmail Images ☰



Search Google or type URL 🔊





Machine Learning: FINN

Presented By

Kees Vissers

Fellow

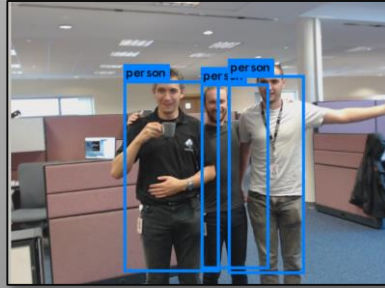
October 1, 2018



Increasing Range of Applications use Machine Learning



Image Classification



Object Detection



Semantic Segmentation

Computer Vision
CNNs



Speaker
Diarization



Speech
Recognition

Speech Recognition
RNNs, LSTMs



Translation



Sentiment Analysis

Natural Language Processing
Sequence to sequence



Recommender



GamePlay

Others

Popular Neural Networks

ResNet50, VGG, AlexNet, InceptionV3



Image Classification

Faster R-CNN, Yolo9000, YoloV2



Object Detection

Mask-R-CNN, SSD

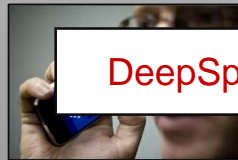


Semantic Segmentation

Computer Vision CNNs



Speaker Diarization



Speech Recognition

DeepSpeech2

Speech Recognition RNNs, LSTMs

Seq2Seq, Transformer

Translation

Seq-CNN

Sentiment Analysis

Natural Language Processing Sequence to sequence

NCF



Recommender

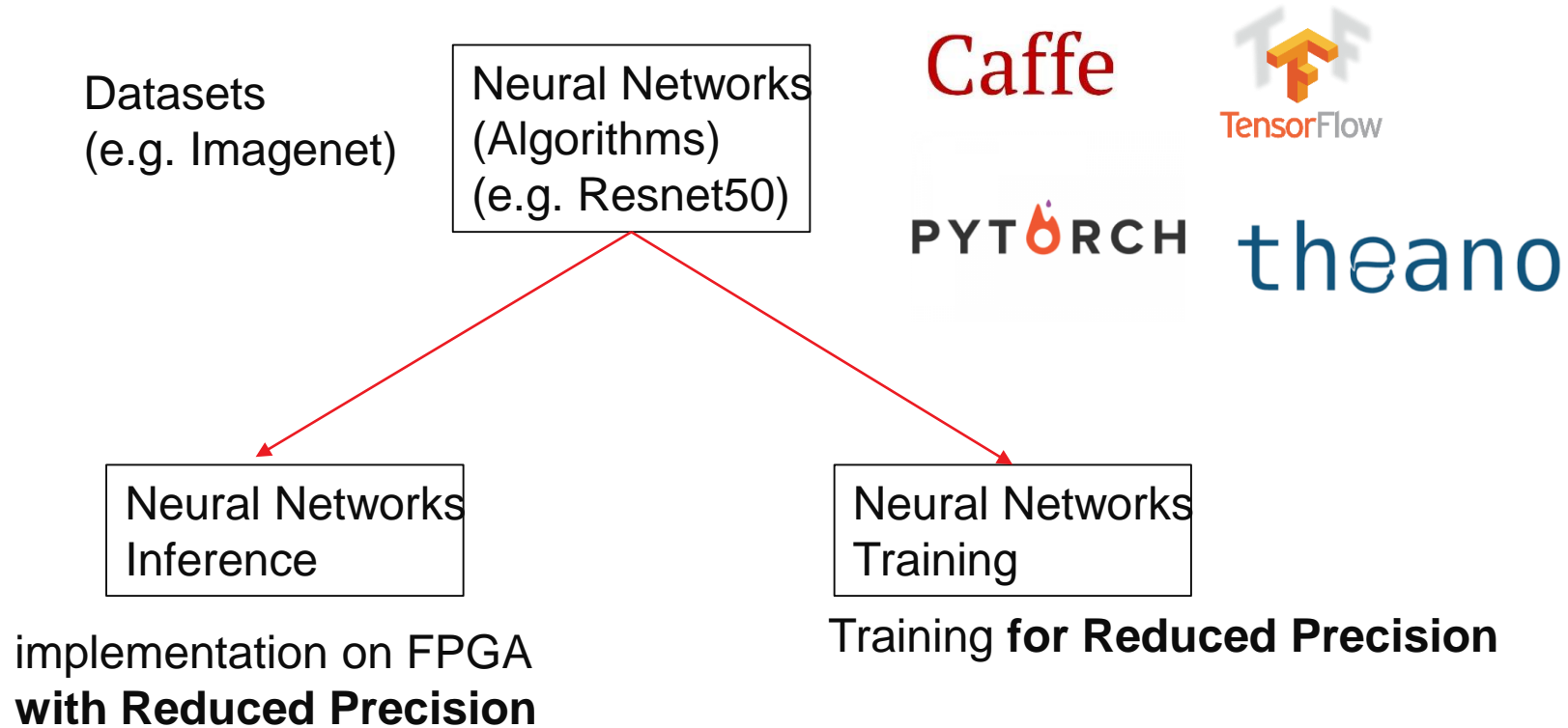
MiniGo, DeepQ, A3C



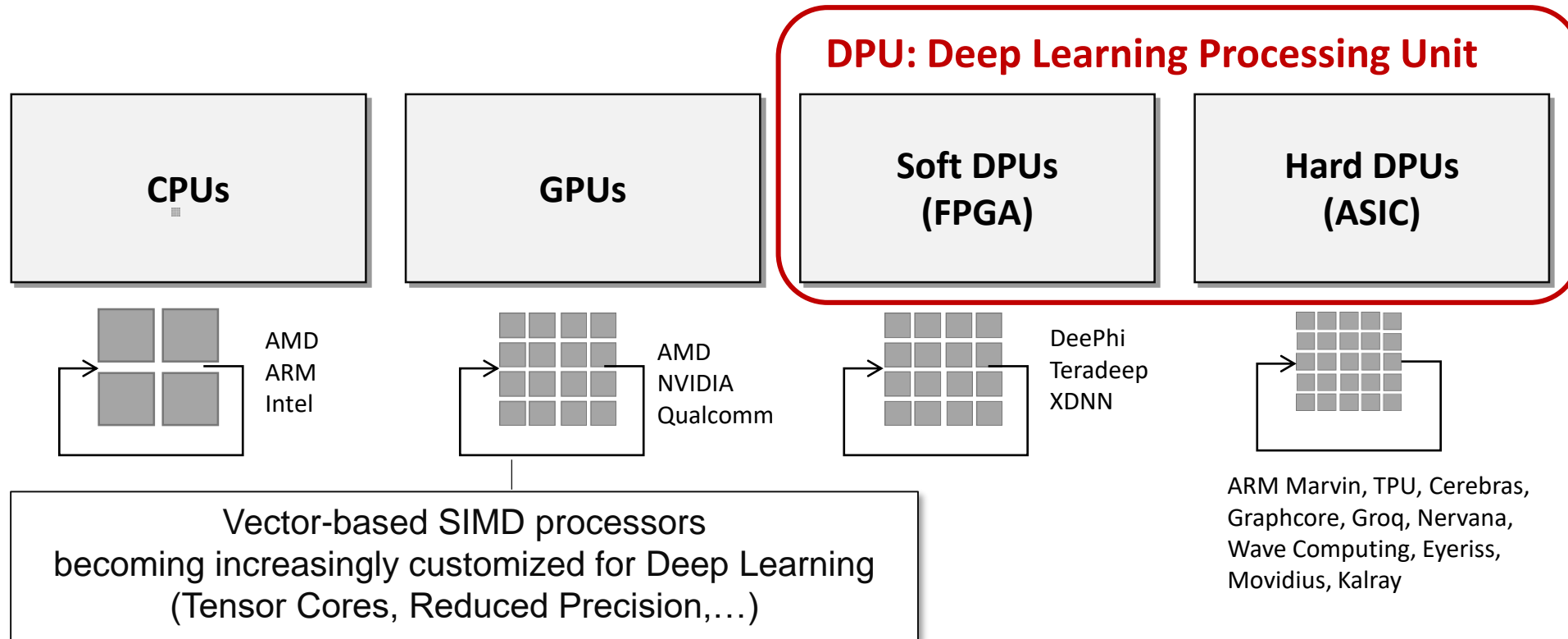
GamePlay

Others

The use of Frameworks for inference on FPGAs



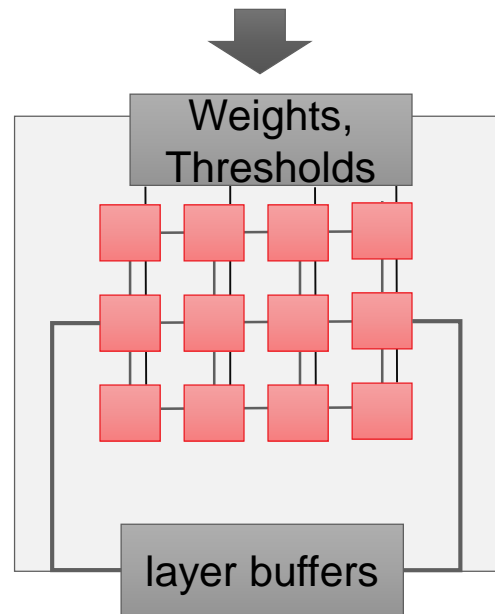
Architectures for Deep Learning



Range of architectural solutions for Inference on FPGA

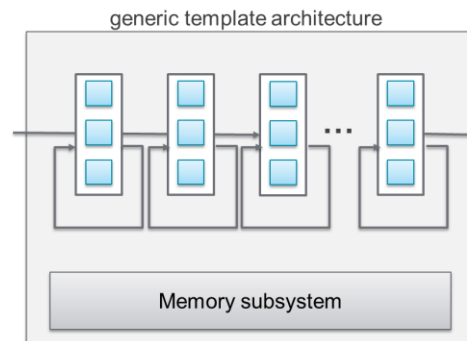
Systolic array processor architecture
XDNN
DeepHi DPU

Dataflow
Direct synthesis
of the Neural Network
FINN, HLS based research solution

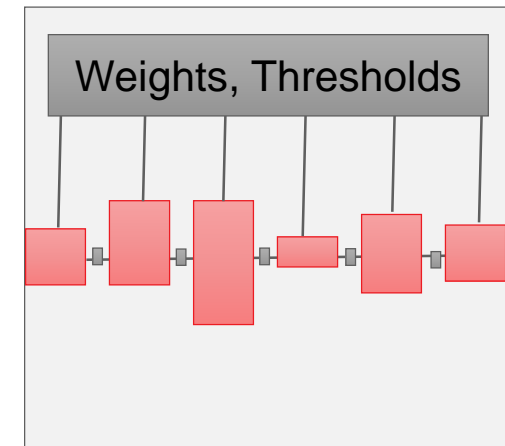


Optimal resource use
Compiler + processor
IP is labor intensive

Intermediate variations



Different Systolic architectures
e.g. DeepHi for CNN
and RNN, LSTM



Scalable for devices
Flexible for bit-precisions
On-chip memory limits

DPU Specialization

Xilinx DPU

XDNN (Cloud/CNN)
Aristotle (Edge/CNN)
Descartes (Edge/Cloud LSTM)

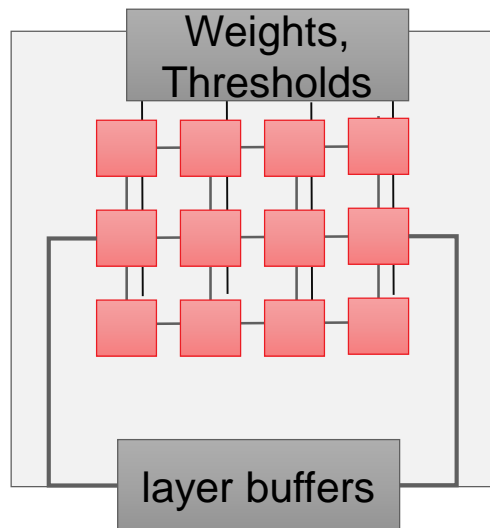
<https://github.com/Xilinx/ml-suite>
<http://deephi.com>

Spectrum of Options

Xilinx FINN

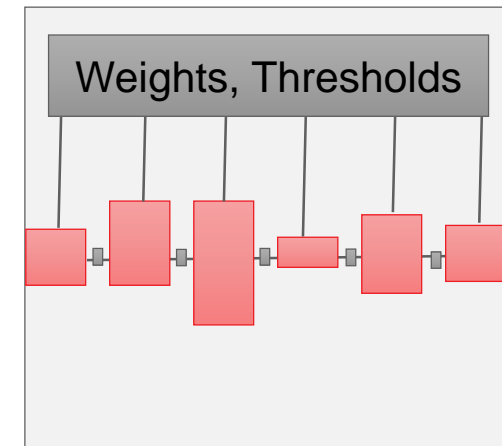
<https://github.com/Xilinx/FINN>
<https://github.com/Xilinx/BNN-PYNQ>
<https://github.com/xilinx/LSTM-PYNQ>

Using Pynq see
<http://pynq.io/ml>

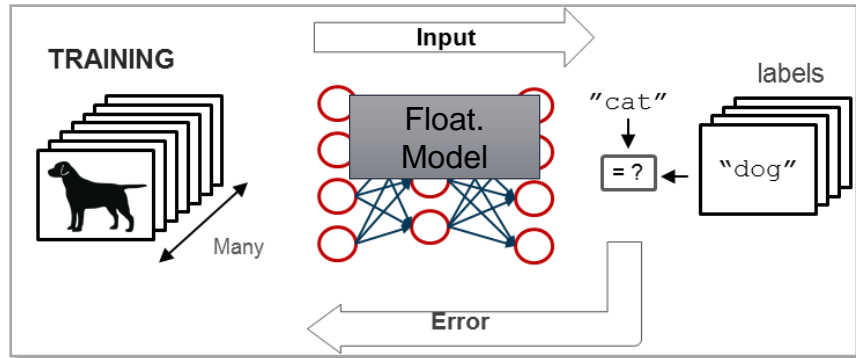


Focus on direct
Implementation
**With reduced
Precision**

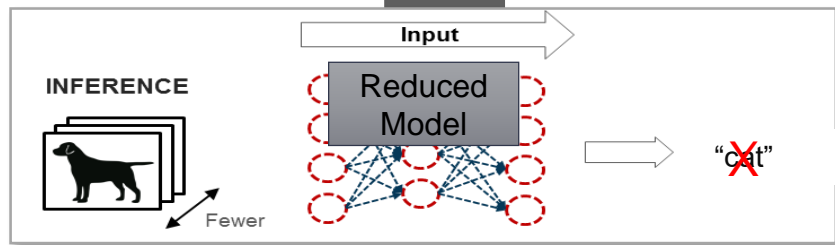
Open Source



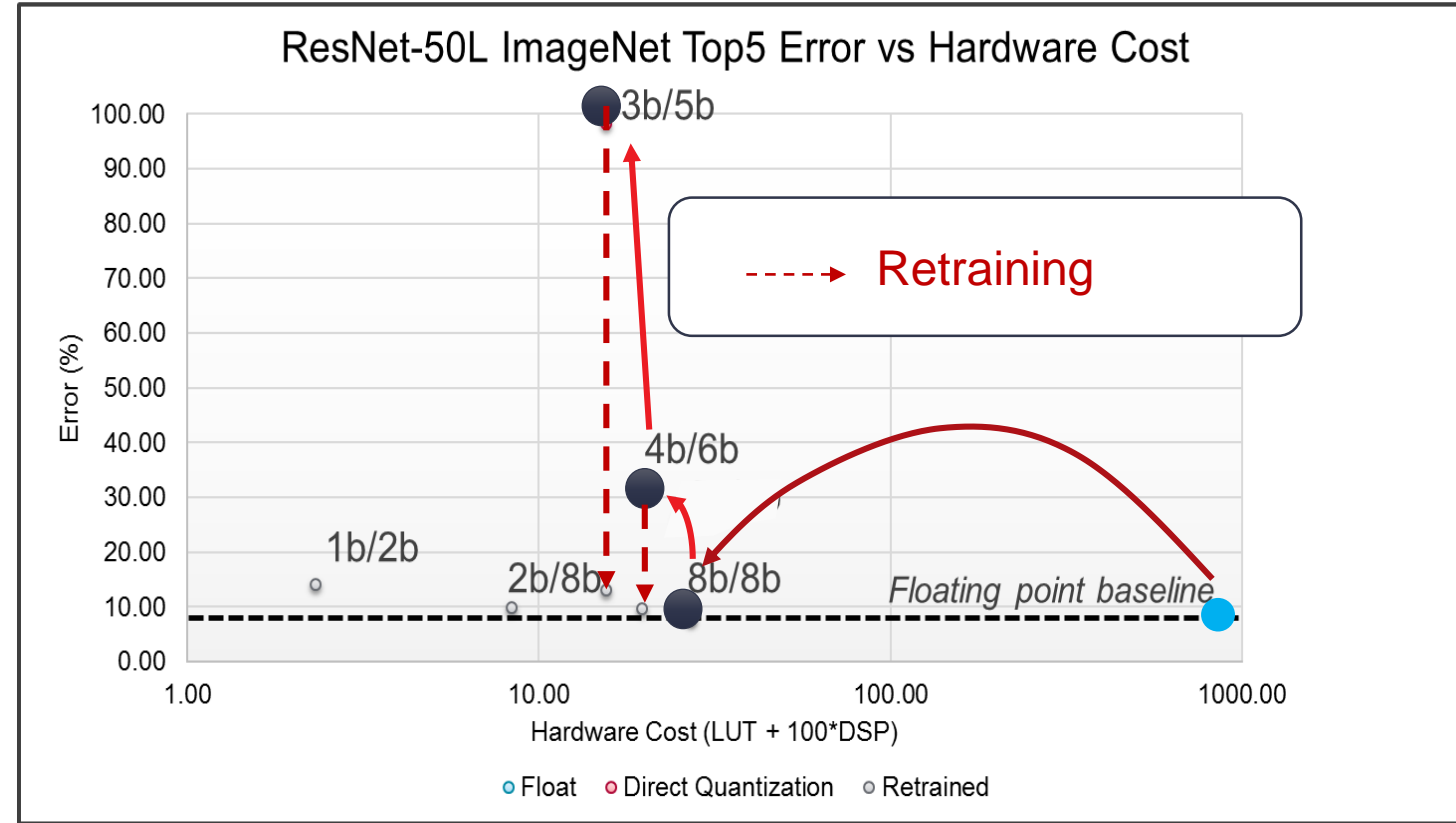
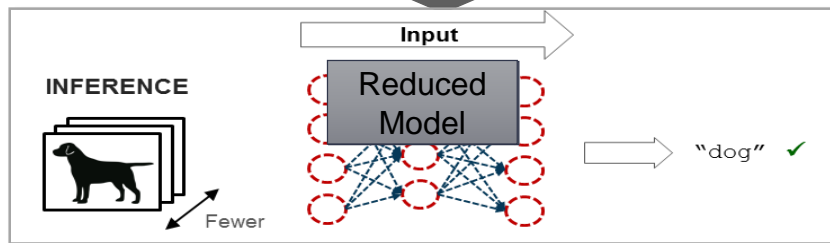
FINN: Reduced Precision and retraining



Reduce Precision

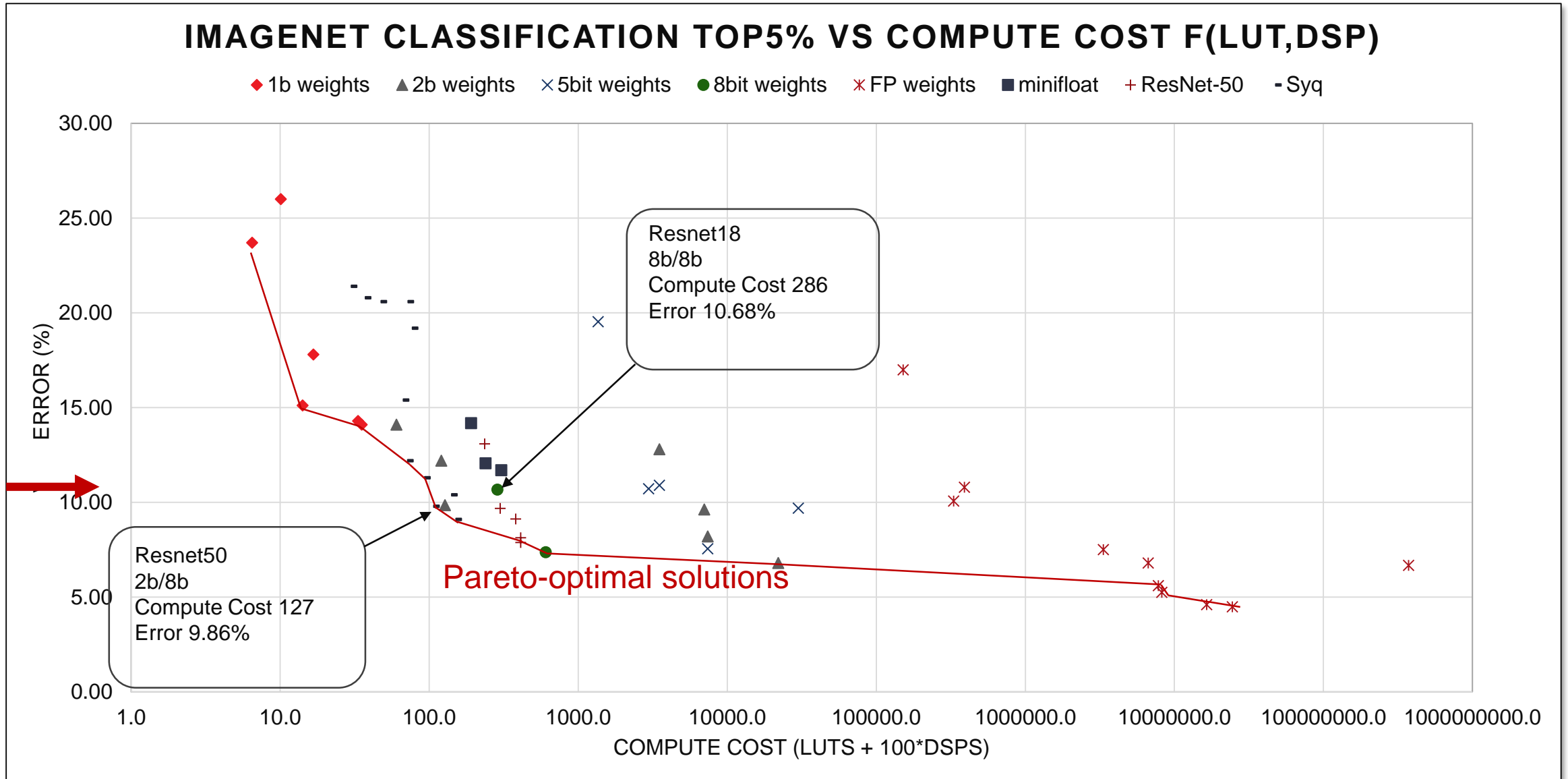


Retraining



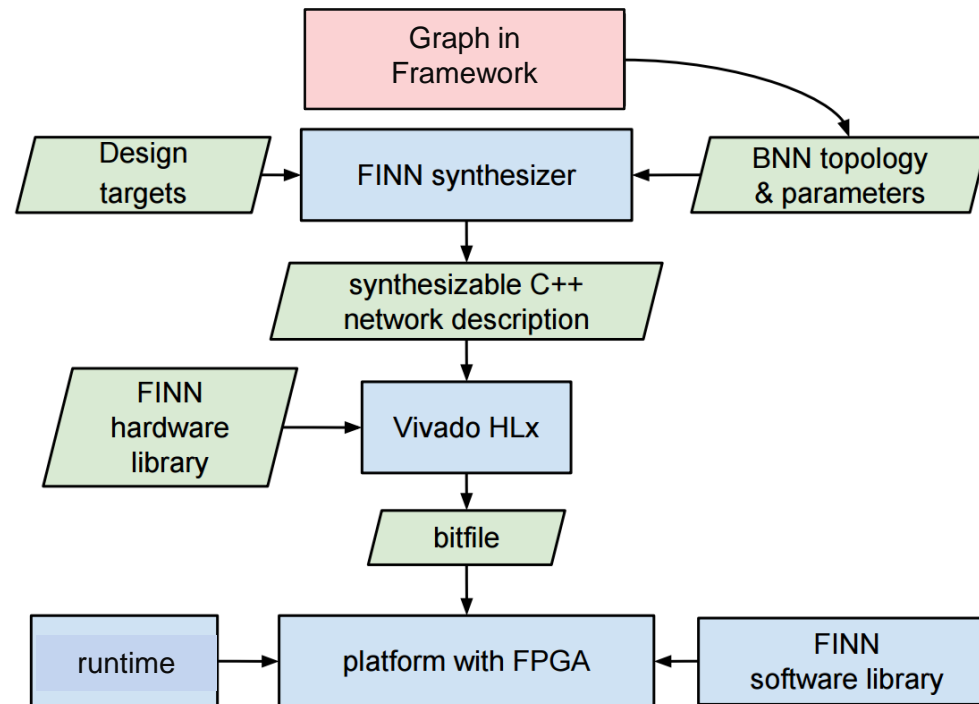
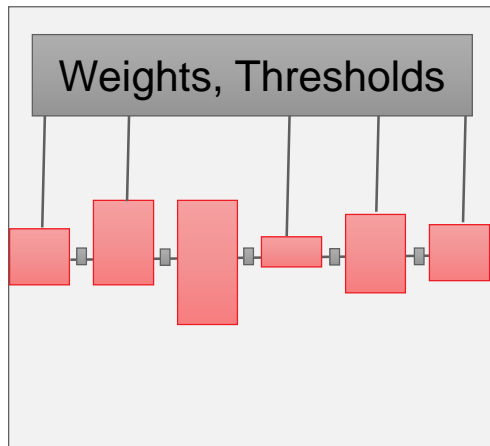
Notation: 3b/5b: 3 bit weights/ 5 bit activation

FINN: Design Space Exploration



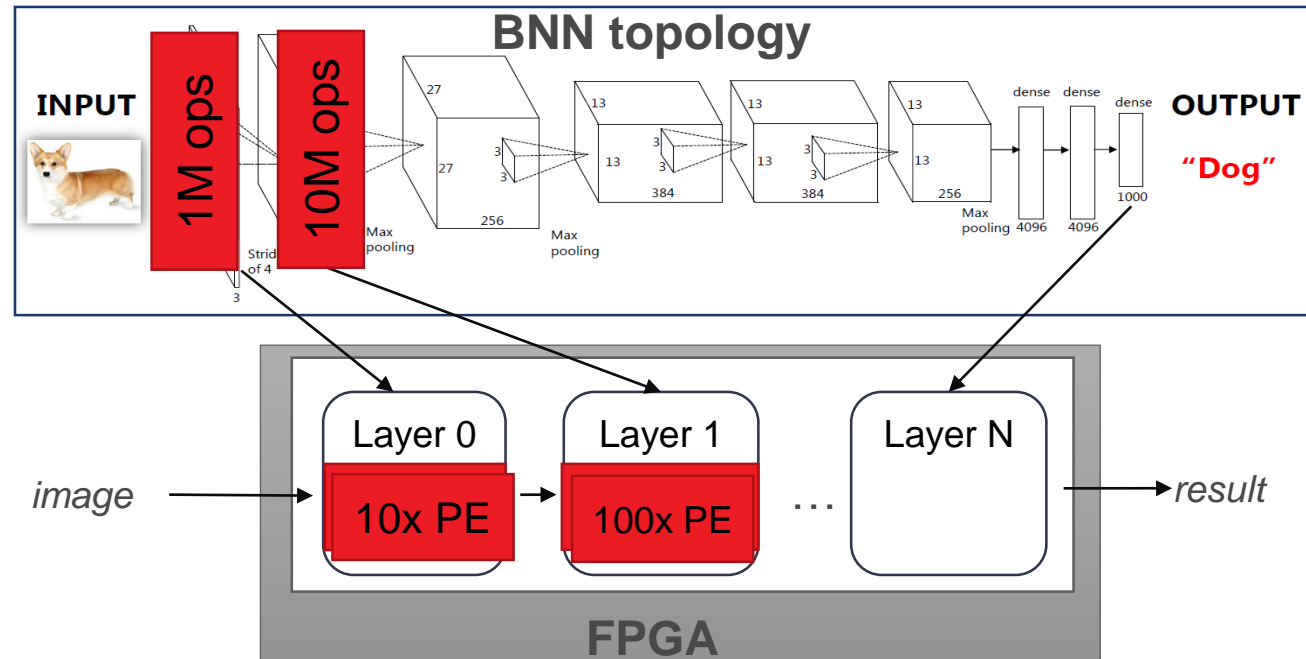
FINN Concepts dataflow implementation of Neural Networks

Caffe theano PyTorch



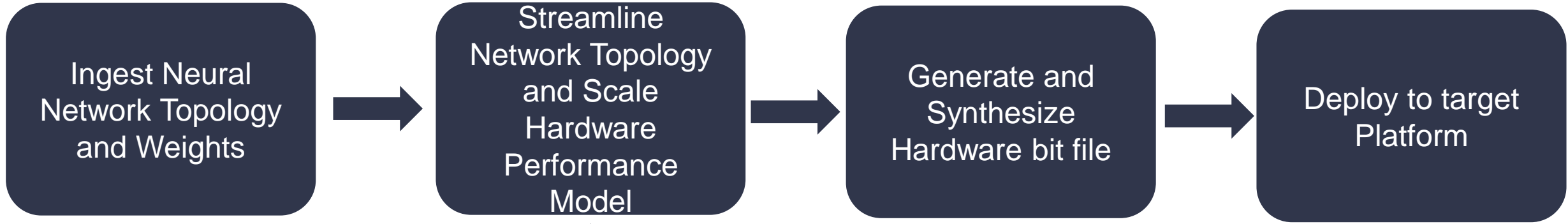
- **Each layer custom parameters**
 - Number of Bits
 - Scaling factor
 - Folding factor

Heterogeneous Streaming Architecture



- > **One hardware layer per BNN layer, parameters built into bitstream,**
- > **Design for balanced throughput**
 - >> Allocate compute resources according to FPS and network requirements
- > **Streaming: Maximize throughput, minimize latency**
 - >> Overlapping computation and communication, batch size = 1, sliding windows between the layers

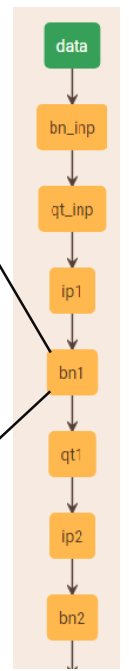
FINN: for binarized neural networks, training in Caffe



Caffe

```

layer {
  name: "conv1"
  type: "BinaryConvolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 20.0499
    decay_mult: 1
  }
  convolution_param {
    num_output: 64
    pad: 0
  }
}
  
```



Target Platform Descriptor File

```

1  {
2  "name": "Xilinx:PYNQ-Z1",
3  "type": "fpga",
4  "frequency" : 200,
5  "part": "xc7z020clg400-1",
6  "resources": {
7    "LUT": 53200,
8    "DSP": 220,
9    "BRAM": 4.9,
10   "URAM": 0
11 }
12 }
  
```



```

template<
// convolution parameters
unsigned int ConvKernelDim,
unsigned int IFMChannels,
unsigned int IFMDim,
unsigned int OFMChannels,
// unsigned int OFMDim,
unsigned int Stride,

// matrix-vector unit parameters
unsigned int SIMDWidth,
unsigned int PECCount,
unsigned int WMemCount,
unsigned int TMemCount,
  
```



```

6. Detailed Classification Information
In addition to highest ranked class, it is possible to get the rank of every class using
example, take another couple of images of a car, an airplane, and a bird and place them
In [6]: from IPython.display import display
im = Image.open('/home/xilinx/jupyter_notebooks/bnn/car.png')
im.thumbnail((64, 64), Image.ANTIALIAS)
display(im)
car_class = classifier.classify_details(im)
print("Car classes: {}".format(car_class))

im = Image.open('/home/xilinx/jupyter_notebooks/bnn/airplane.jpg')
im.thumbnail((64, 64), Image.ANTIALIAS)
display(im)
air_class = classifier.classify_details(im)
print("Airplane classes: {}".format(air_class))

im = Image.open('/home/xilinx/jupyter_notebooks/bnn/bird.jpg')
im.thumbnail((64, 64), Image.ANTIALIAS)
display(im)
bird_class = classifier.classify_details(im)
print("Bird classes: {}".format(bird_class))

Inference took 1602.00 microseconds
Classification rate: 624.22 images per second
Car classes: [256 379 165 160 163 244 249 244 267 268]
  
```

BNN- PYNQ: reduced precision networks + training

theano

- Theano and BinaryNet
- Training Scripts available for network examples
- How-to set-up the training environment
- Support for arbitrary precision on Theano



- Network parameters exporting
- Python scripts to export weights and generate thresholds @ target precision
- Loaded at run-time



- Network description in VHLS (C++)
- Relying on a validated HLS library
- Including all most common layers
- Examples available for multiple networks and multiple precisions



- Bitstream generation
- Scripts targeting multiple supported platforms (PYNQ-Z1, PYNQ-Z2, Ultra96)



PYNQ

- Runtime for PYNQ platforms
- Stack of SW, wrapped in a python class easy to use in jupyter environment

LSTM – PYNQ: environment for LSTM with training

 PyTorch

Pytorch-ocr

- Small training library for OCR with Pytorch using a LSTM-based network
- Can be extended to different OCR datasets
- Export of a trained quantized network's description and weights



Pytorch-quantization

- Support for training of quantized LSTMs and FC at arbitrary multi-bit precision
- Low-level export API of quantized layers



Vivado™ HLS

- Network description in VHLS (C++)
- HLS library for BiLSTM acceleration
- Takes in network's file generated by Pytorch



Bitstream generation

- Scripts targeting PYNQ-Z2
- Baked in per-network weights and config



PYNQ™

Runtime for PYNQ platforms

- Stack of SW, wrapped in a python class easy to use in jupyter environment

Repositories

- > <https://github.com/Xilinx/FINN>
- > <https://github.com/Xilinx/BNN-PYNQ>
- > <https://github.com/xilinx/LSTM-PYNQ>
- > <https://github.com/Xilinx/pytorch-quantization>
- > <https://github.com/Xilinx/pytorch-ocr>
- > <https://github.com/Xilinx/QNN-MO-PYNQ>
- > <http://www.pynq.io>
- > <http://www.pynq.io/ml>
- > <https://github.com/Xilinx/ml-suite>
- > <http://www.ultra96.org>
- > <http://deephi.com>

Publications

- > FPL'18: [FINN-L: Library Extensions and Design Trade-off Analysis for Variable Precision LSTM Networks on FPGAs](#)
- > FPL'18: [BISMO: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing](#)
- > FPL'18: [Customizing Low- Precision Deep Neural Networks For FPGAs](#)
- > ACM TRETS, Special Issue on Deep Learning: [FINN-R: An End-to-End Deep- Learning Framework for Fast Exploration of Quantized Neural Networks](#)
- > ARC'18: [Accuracy to Throughput Trade-Offs for Reduced Precision Neural Networks on Reconfigurable Logic](#)
- > CVPR'18: [SYQ: Learning Symmetric Quantization For Efficient Deep Neural Networks](#)
- > DATE'18: [Inference of quantized neural networks on heterogeneous all-programmable devices](#)
- > ICONIP'17: [Compressing Low Precision Deep Neural Networks Using Sparsity-Induced Regularization in Ternary Networks](#)
- > ICCD'17: [Scaling Neural Network Performance through Customized Hardware Architectures on Reconfigurable Logic](#)
- > PARMA-DITAM'17: [Scaling Binarized Neural Networks on Reconfigurable Logic](#)
- > FPGA'17: [FINN: A Framework for Fast, Scalable Binarized Neural Network Inference](#)
- > H2RC'16: [A C++ Library for Rapid Exploration of Binary Neural Networks on Reconfigurable Logic](#)



Conversation with Xilinx Research Labs: P4 and NetFPGA

Presented By

Gordon Brebner

Distinguished Engineer

1 October 2018



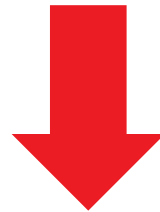
Open source programmable networking on FPGA



P4 programming language for packet processing



NetFPGA platform for line-rate packet processing



Automated workflow for running P4 on NetFPGA

Benefits of Programmable Networking (... or FPGA in fact)

C

- Control and Customization. Make switch or SmartNIC behave exactly as you want

R

- Reliability. Reduce risk by removing unused features

E

- Efficiency. Reduce energy consumption and expand scale by doing only what you need

A

- Add new features on your schedule

T

- Telemetry. Be able to see inside the network

E

- Exclusivity and Differentiation. Add secret sauce to vendor offerings

P4

Programming Protocol-independent Packet Processors



- > **Language first appeared in paper published in July 2014**
- > **Three goals:**
 - >> Reconfigurability in the field – reprogramming of networking equipment
 - >> Protocol independence – not tied to any specific networking protocols
 - >> Target independence – not tied to any specific networking hardware
- > **P4 consortium (P4.org) set up in 2015 – now an open source Linux Foundation project**
 - >> Xilinx was a founding member of P4.org
 - >> Now has >100 members
- > **P4 has emerged as the *de facto* standard language for packet processing**

P4 language elements

Parsers

State Machines,
bit-field extraction

Controls

Match-Action Tables,
control flow statements

Expressions

Basic operations
and operators

Data Types

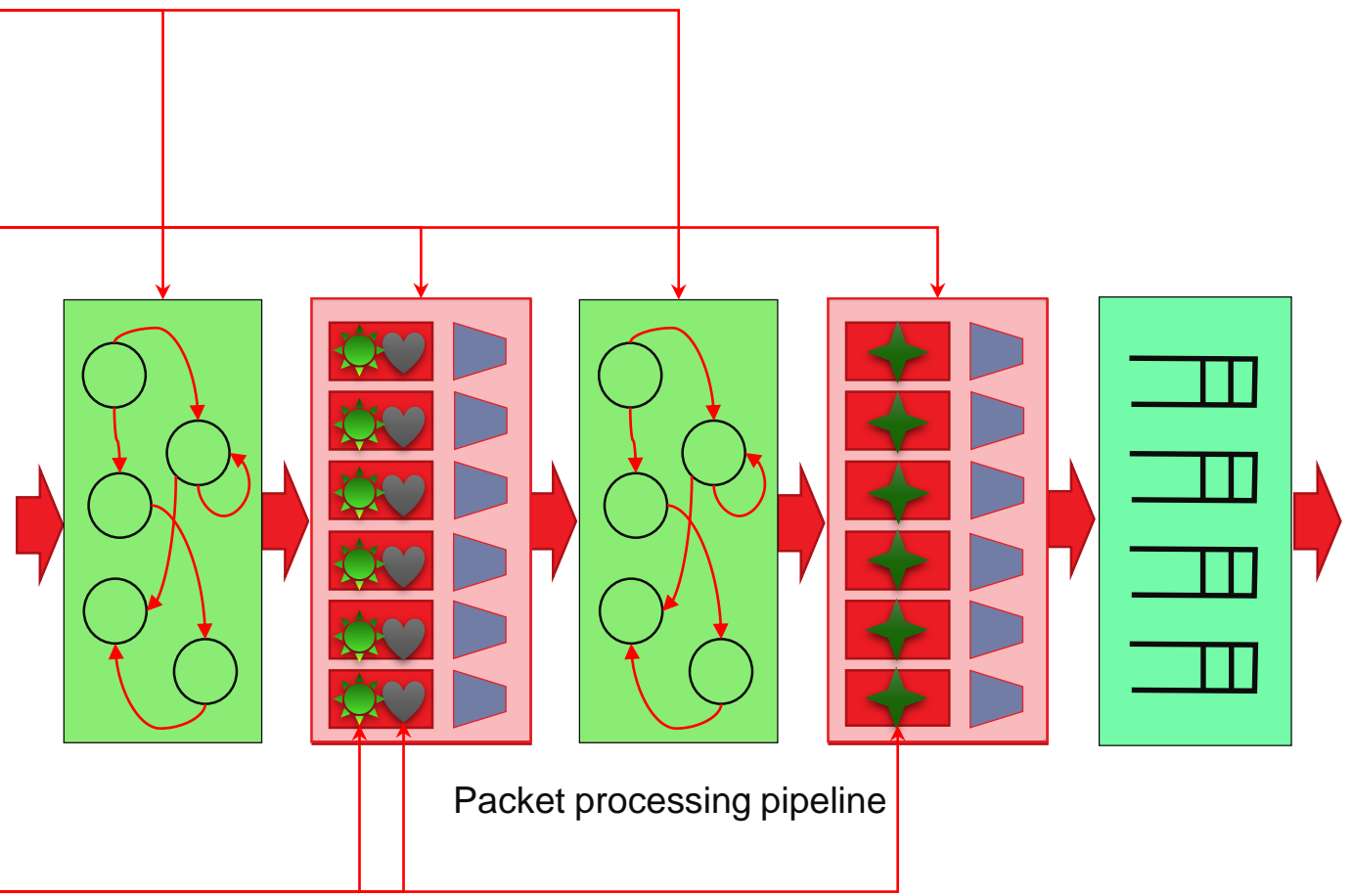
Bit-strings, headers,
structures, arrays

Architecture

Programmable blocks
and their interfaces

Extern Libraries

Support for specialized
components



P4 “Hello World” (networking style) example

```
#include <core.p4>
#include <XilinxSwitch.p4>
struct user_meta_t {}
struct headers {}

parser MyParser(packet_in packet, out headers hdr,
  inout user_meta_t meta,
  inout std_meta_t std_meta) {
  state start { transition accept; }
}

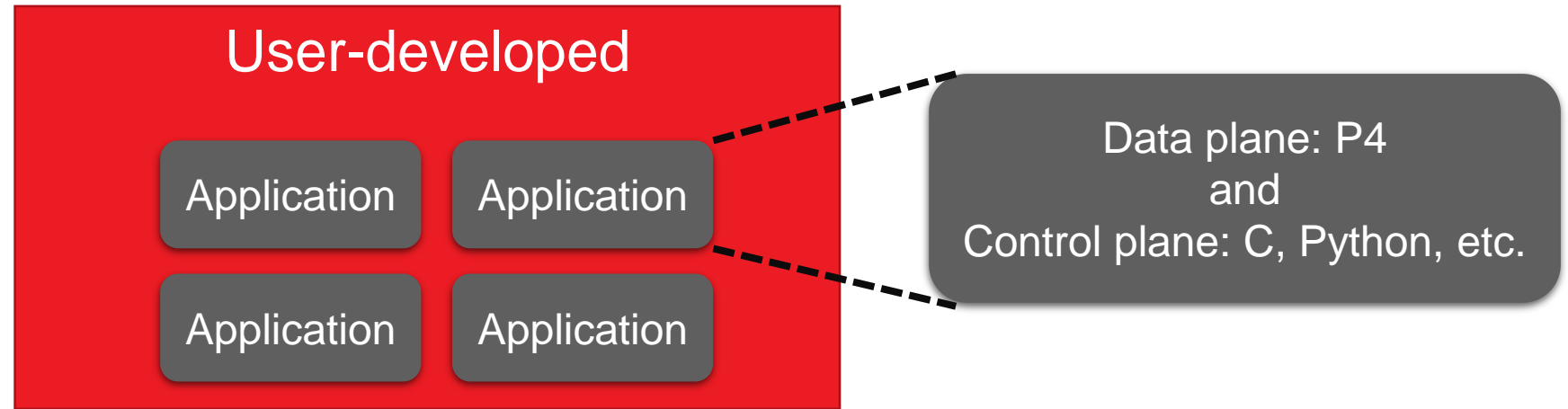
control MyPipe(inout headers hdr, inout user_meta_t meta,
  in std_meta_t std_meta) {
  action set_egress_port(bit<9> port) {
    std_meta.egress_port = port;
  }
  table forward {
    key = { std_meta.ingress_port: exact; }
    actions = {
      set_egress_port;
      NoAction;
    }
    size = 1024;
    default_action = NoAction();
  }
  apply { forward.apply(); }
}
```

```
control MyDeparser(packet_out packet, in headers hdr) {
  apply { }
}
```

```
XilinxSwitch( MyParser(), MyPipe(), MyDeparser() ) main;
```

Key	Action Name	Action Data
1	set_egress_port	2
2	set_egress_port	1

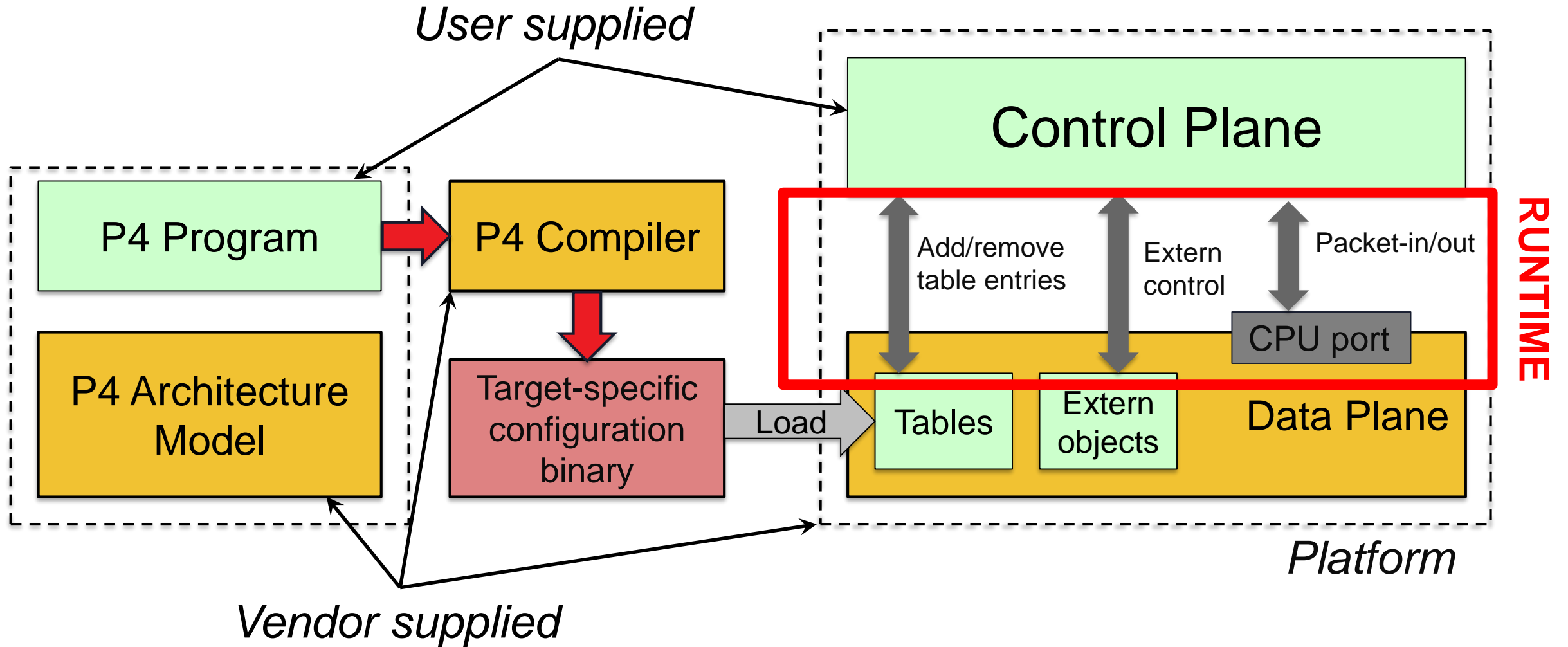
P4 ecosystem



+



Programming and operating a P4 platform



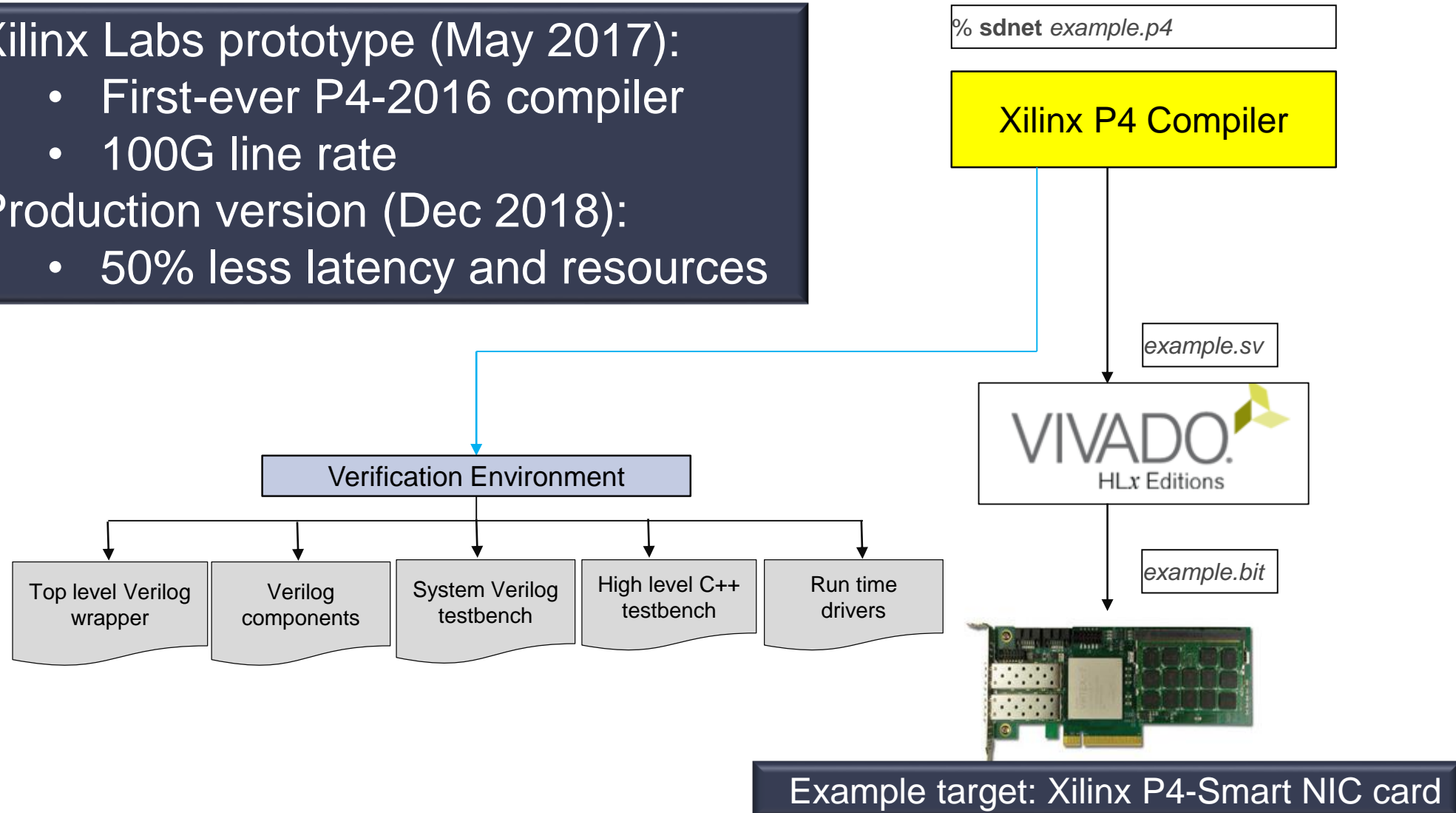
Xilinx P4-SDNet product (www.xilinx.com/sdnet)

Xilinx Labs prototype (May 2017):

- First-ever P4-2016 compiler
- 100G line rate

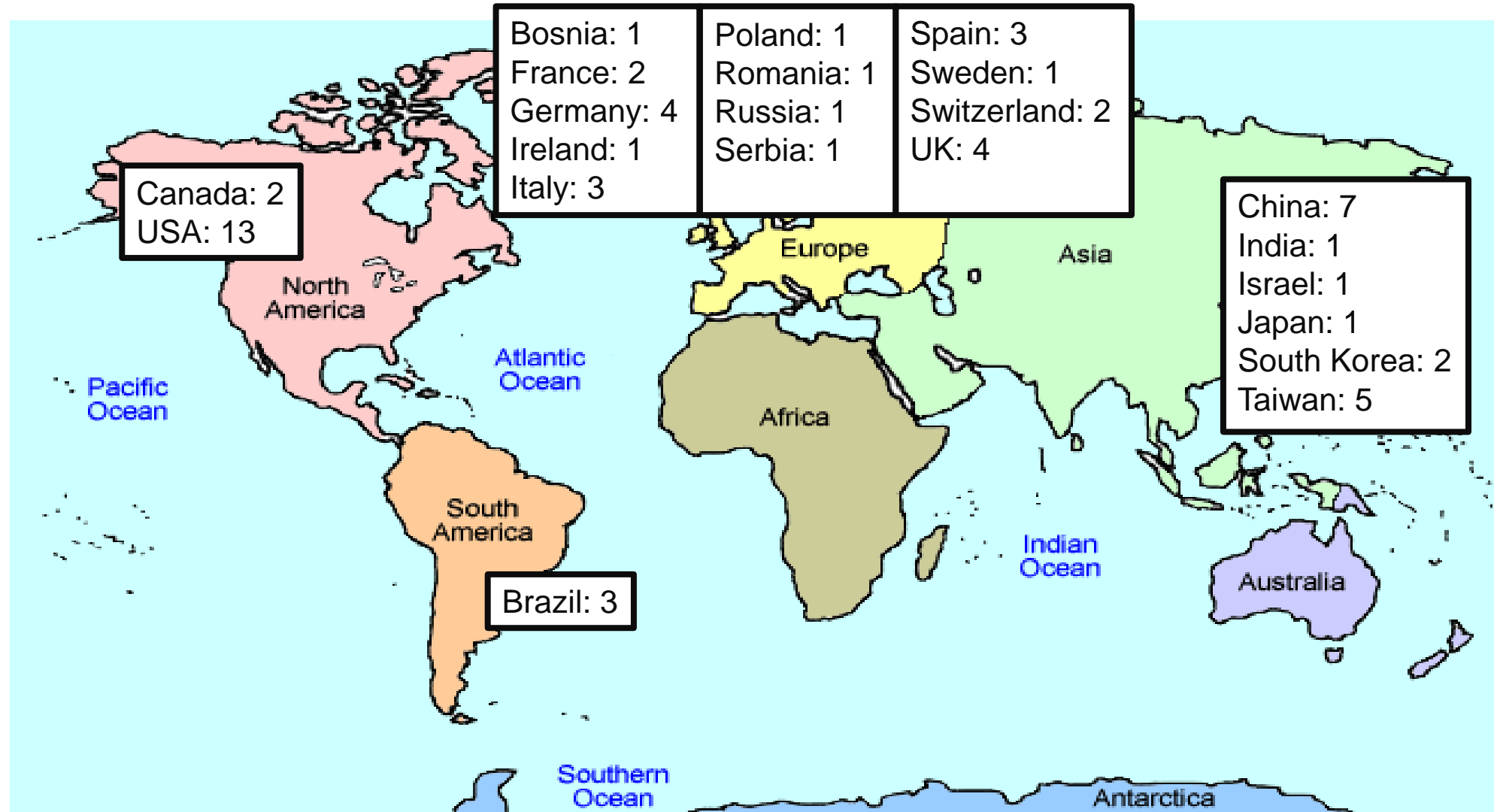
Production version (Dec 2018):

- 50% less latency and resources



Example target: Xilinx P4-Smart NIC card

P4-SDNet research community today: 60 institutions in 22 countries



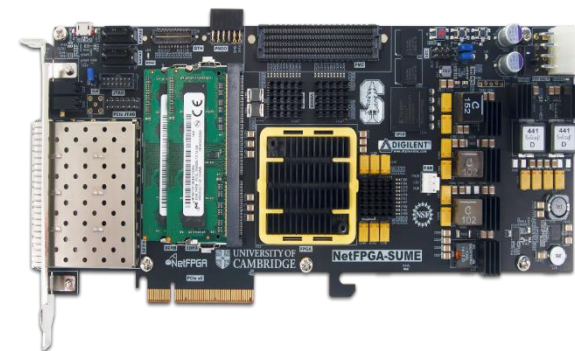
NetFPGA (= Networked FPGA)



- > Line-rate, flexible, open networking platform for teaching and research
- > Community began with Stanford and Xilinx Labs, now anchored at Cambridge
- > NetFPGA systems deployed at over 150 institutions in over 40 countries

Four elements:

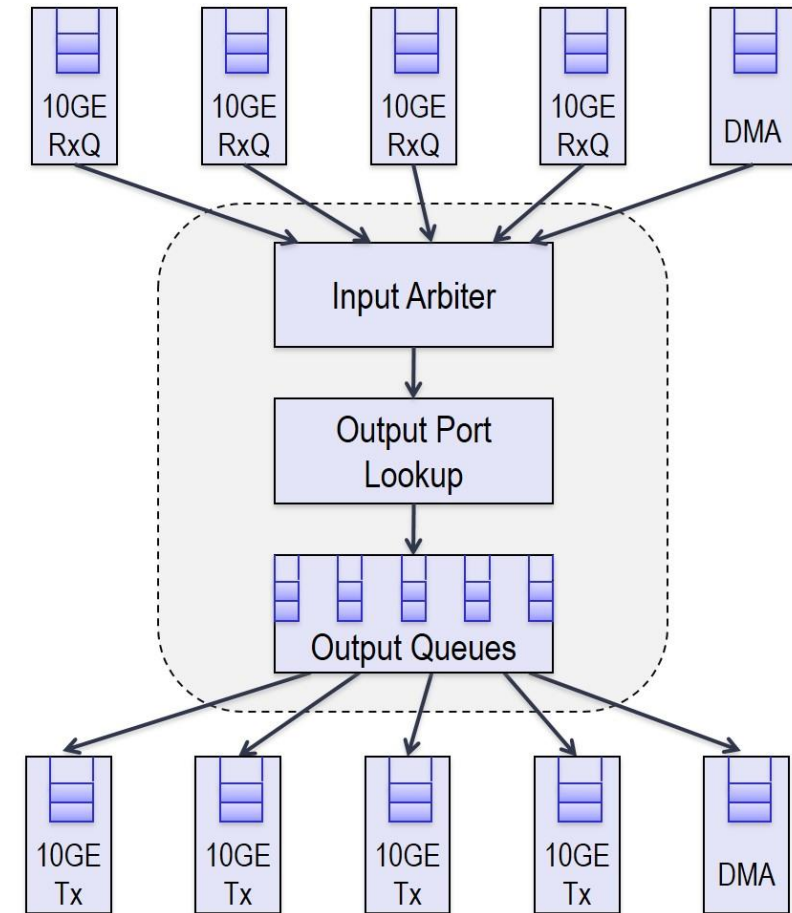
- > Community: NetFPGA.org
- > Low-cost board family
- > Tools and reference designs
- > Contributed projects



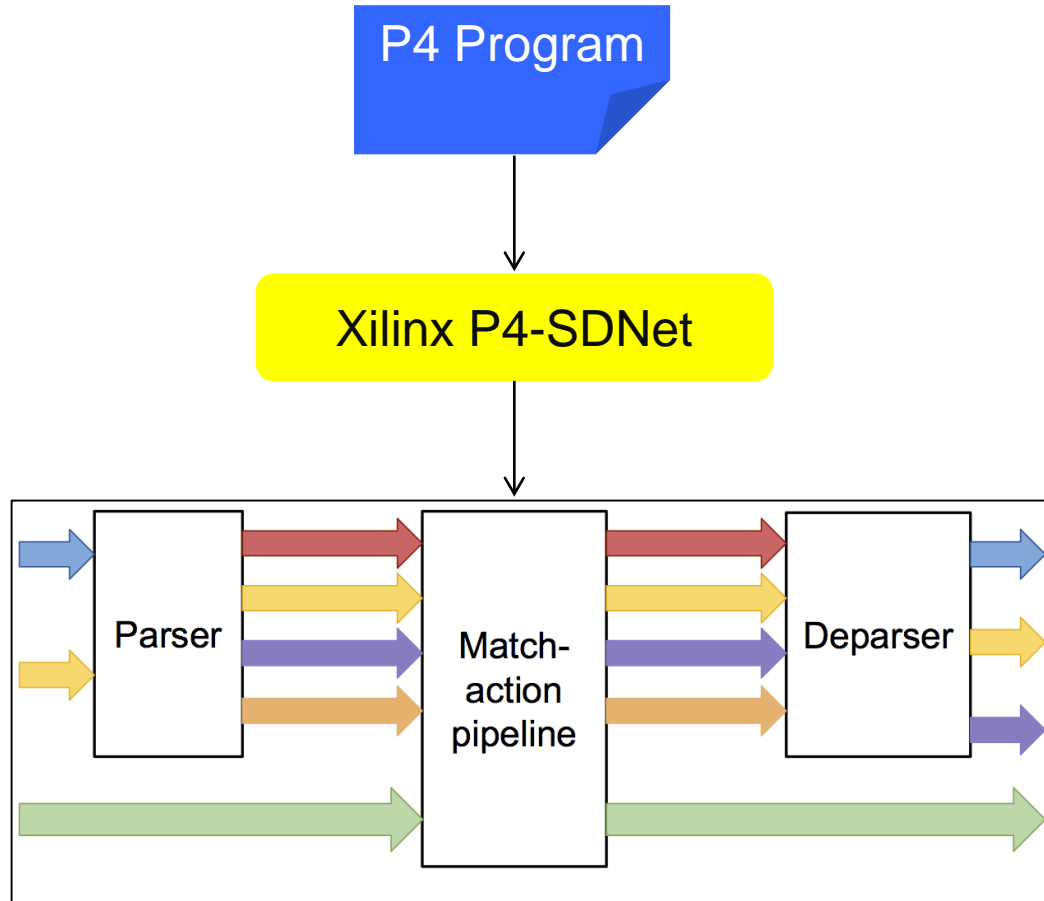
NetFPGA-SUME
4x10G ports

NetFPGA SUME reference switch design

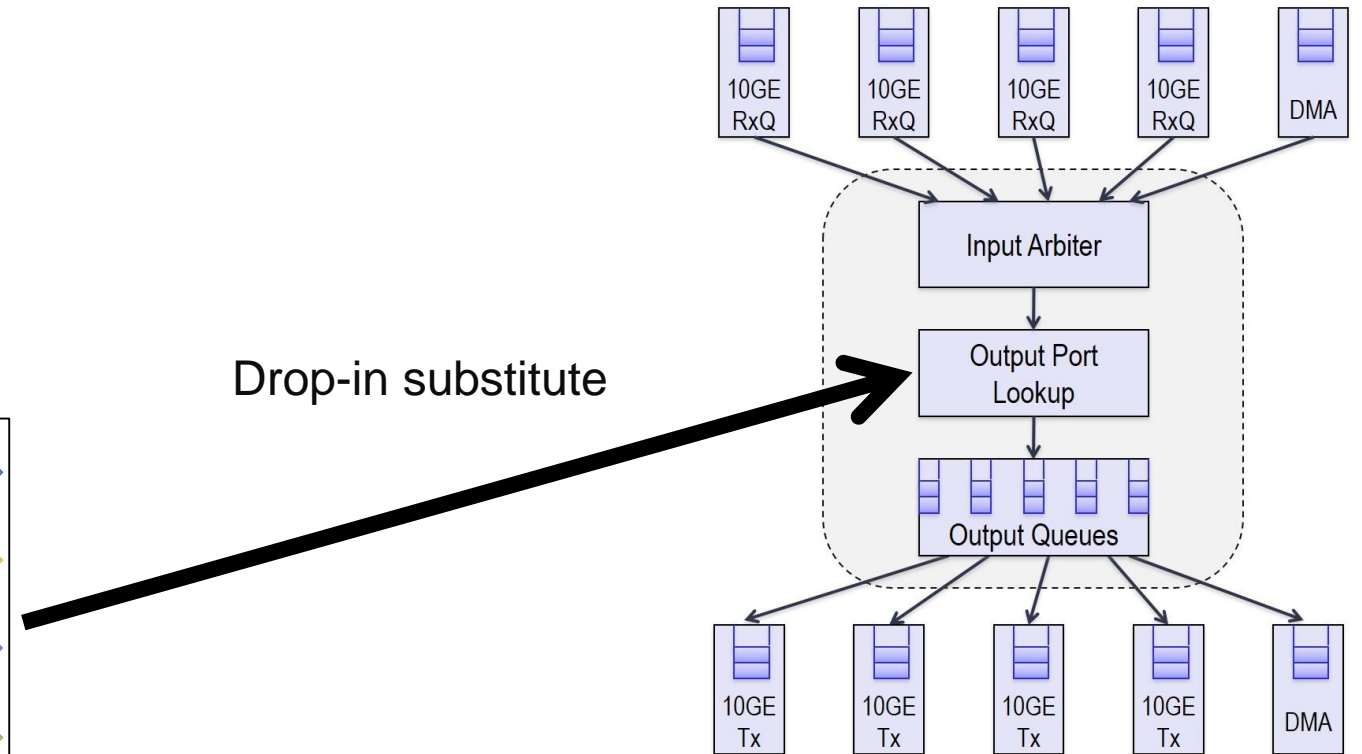
- > 4x10G Ethernet switch, with CPU slow path as 5th port
- > Five-stage pipeline:
 - >> Input ports
 - >> Input arbitration
 - >> Forwarding decision and packet modification
 - >> Output queuing
 - >> Output ports
- > Standard module interfaces
- > Modules can be customized or substituted



P4→NetFPGA workflow overview



NetFPGA SUME reference switch design



P4→NetFPGA workflow steps

- 1. Write P4 program**
- 2. Write additional externs (if required)**
- 3. Write python test packet script**
- 4. Compile to Verilog / Generate API & CLI tools**
- 5. Run simulations to test/debug**
- 6. Build bitstream**
- 7. Check implementation reports**
- 8. Test on the hardware**

iterate

implement

All of the user effort goes here – not on the FPGA detail

P4→NetFPGA community

- > **150 active members from academia and industry around the world, and growing**
- > **Members of the community highly encouraged to contribute in ways such as:**
 - >> New P4 projects
 - >> Extra extern functions
 - >> Performance analysis tools
 - >> Verification tools
- > **Used for research, and for teaching networking concepts on real hardware**
 - >> No hardware design experience needed
- > **Some current projects**
 - >> Distributed congestion control
 - >> In-band Network Telemetry
 - >> In-network compression; In-network key-value cache
 - >> Network-accelerated sorting; Network-accelerated consensus
- > **Getting started:**
 - >> Public documentation: <https://github.com/NetFPGA/P4-NetFPGA-public/wiki>

Research directions

> Language: Extend coverage of P4

- >> Programmable Traffic Management (MIT + NYU + Stanford + Xilinx Labs + P4.org)
- >> Programmable Target Architectures (Cornell + Stanford + Xilinx Labs)

> Infrastructure: Open source hardware reference platform for P4

- >> Complement existing software reference platform
- >> Cover NIC-style architectures as well as switch-style architectures

> Applications

- >> Programmable networking offload and acceleration
- >> Congestion control, in-band network telemetry
- >> In-network computing
- >> ... your ideas here



RapidWright¹: Modular pre-implemented methodology

Presented By

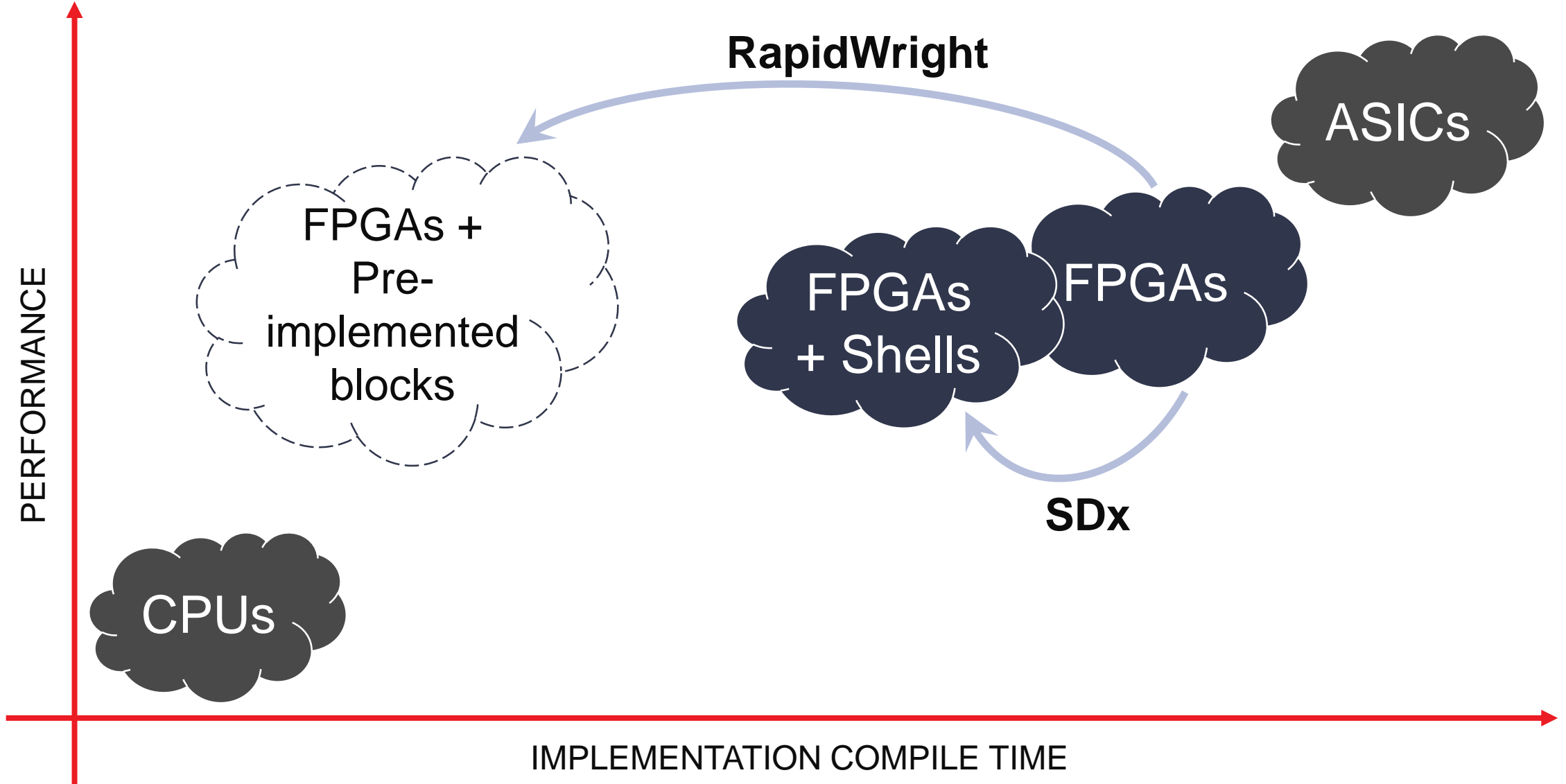
Alireza Kaviani, Ph.D.
Distinguished Engineer,
Xilinx Research Labs

Oct 1, 2018

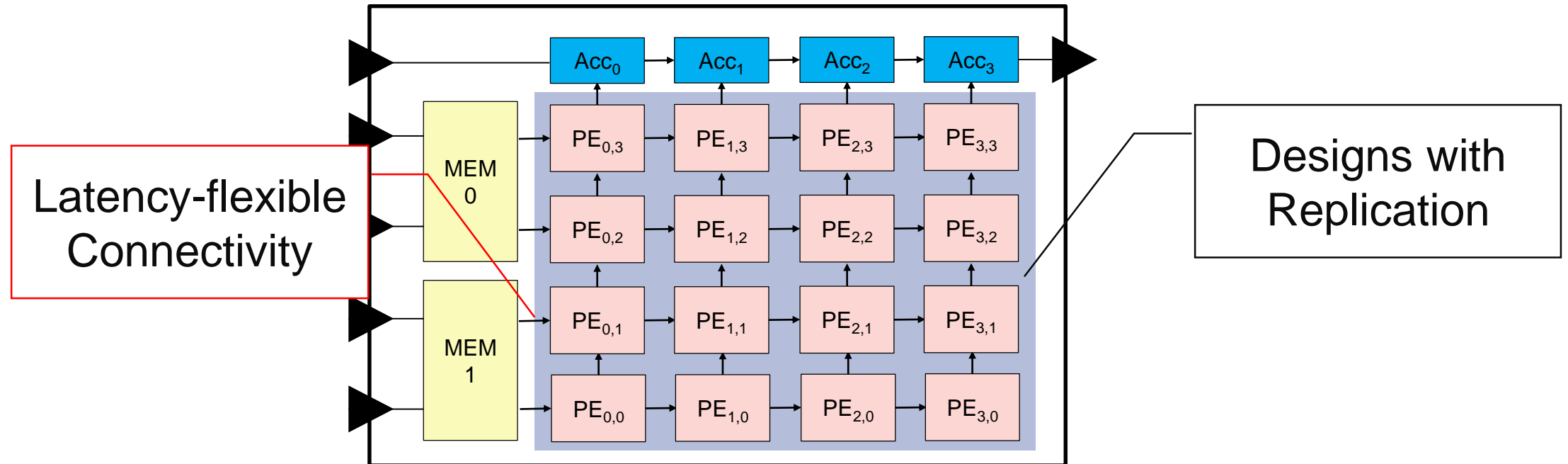
Wright¹ = maker or builder



RapidWright value proposition



Focus on emerging applications



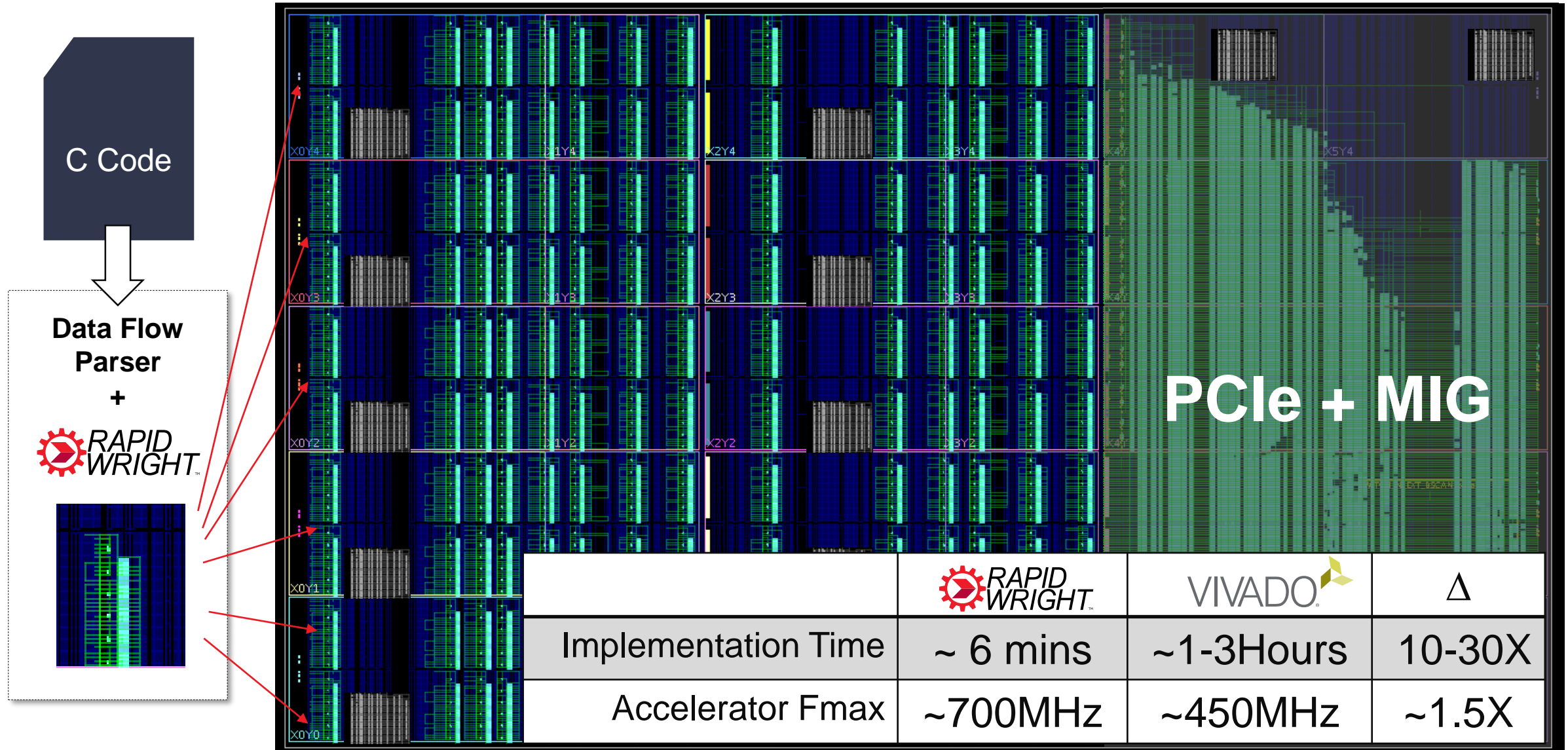
> Module-based approach to implementation

- >> Lock-in performance with reusable modules
- >> Fewer inter-block timing closure issues

> Advantages

- >> >10X reduction in compile time
- >> Near-spec performance
- >> Predictable timing closure

Vision: Rapid accelerator assembly



RapidWright overview

> Enables targeted solutions

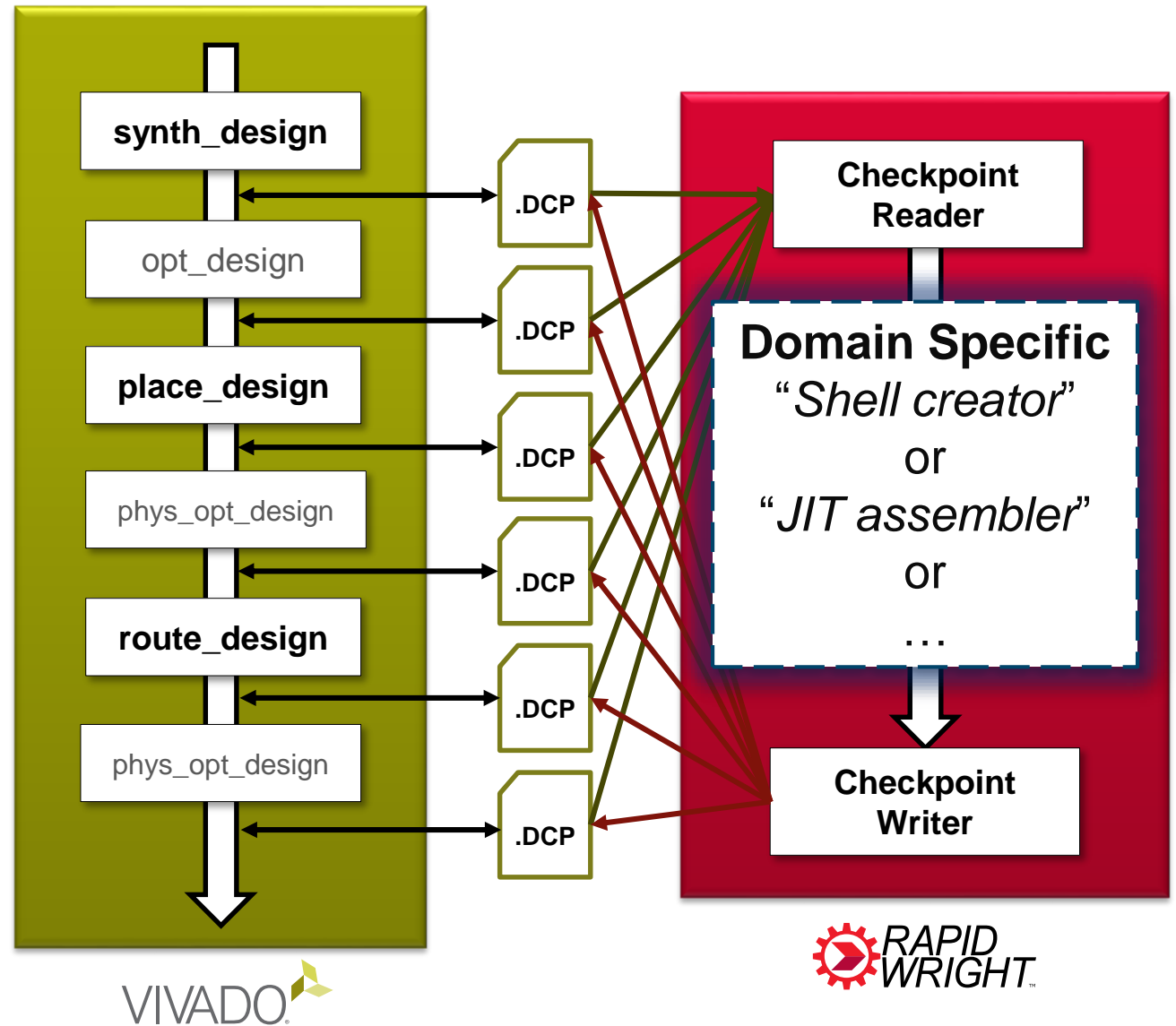
- >> Reuse & relocate pre-implemented modules
- >> Just-in-time implementations
- >> Create shells & overlays

> Companion framework for Vivado

- >> Communicates through Design CheckPoints¹ (DCPs)
- >> Fast, light-weight, open source
- >> Java, Python coding

> Power user ecosystem

- >> Academic algorithm validation
- >> Rapid prototyping of CAD concepts



1: DCP contains netlist + P&R info + constraints

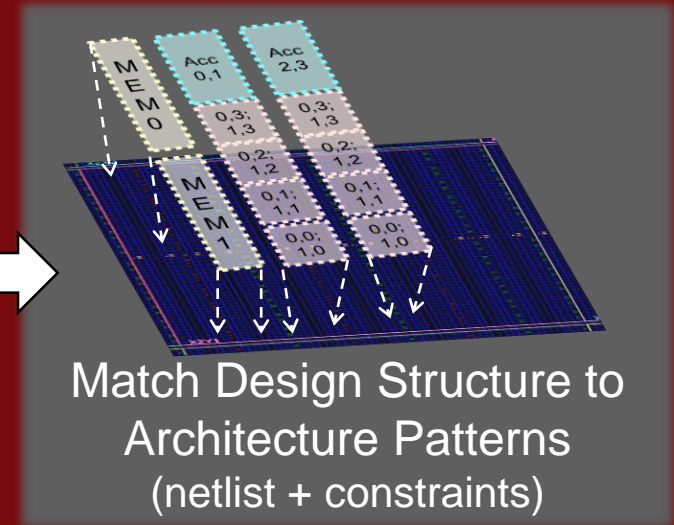
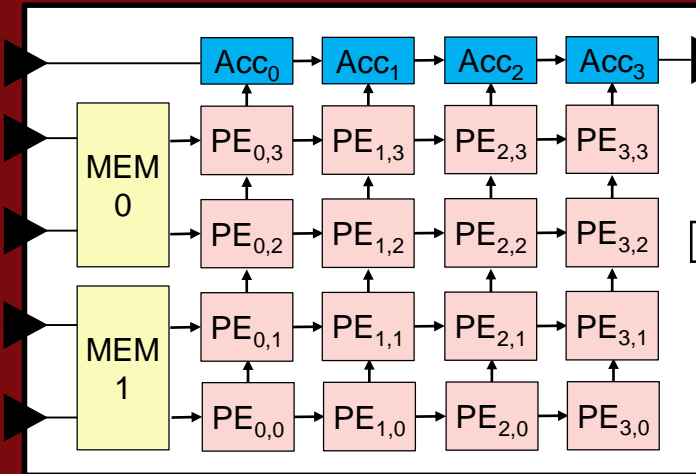
A Modular pre-implemented methodology

DOMAIN DESIGN TASKS

1. Design selection attributes:

- Modular
- Latency tolerant
- Prefers replication

2. Placement planning

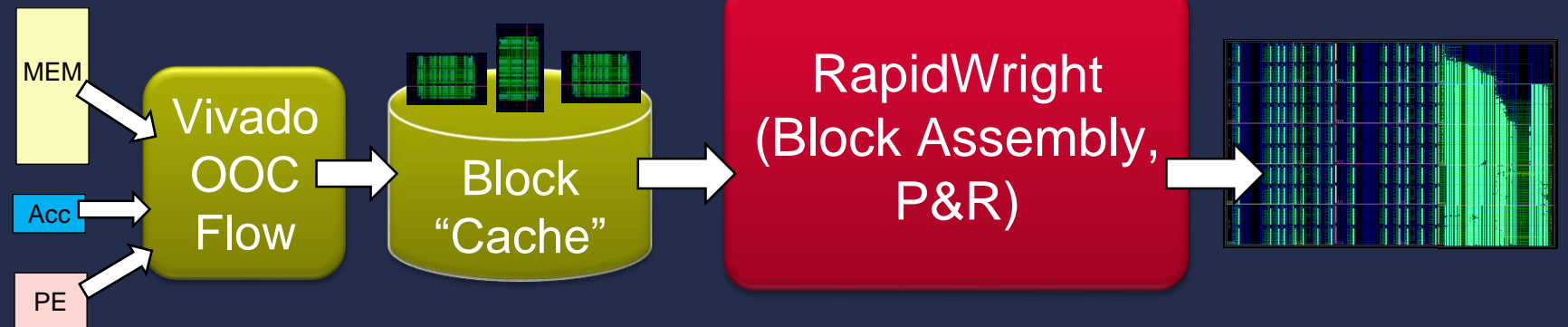


IMPLEMENTATION TOOL TASKS

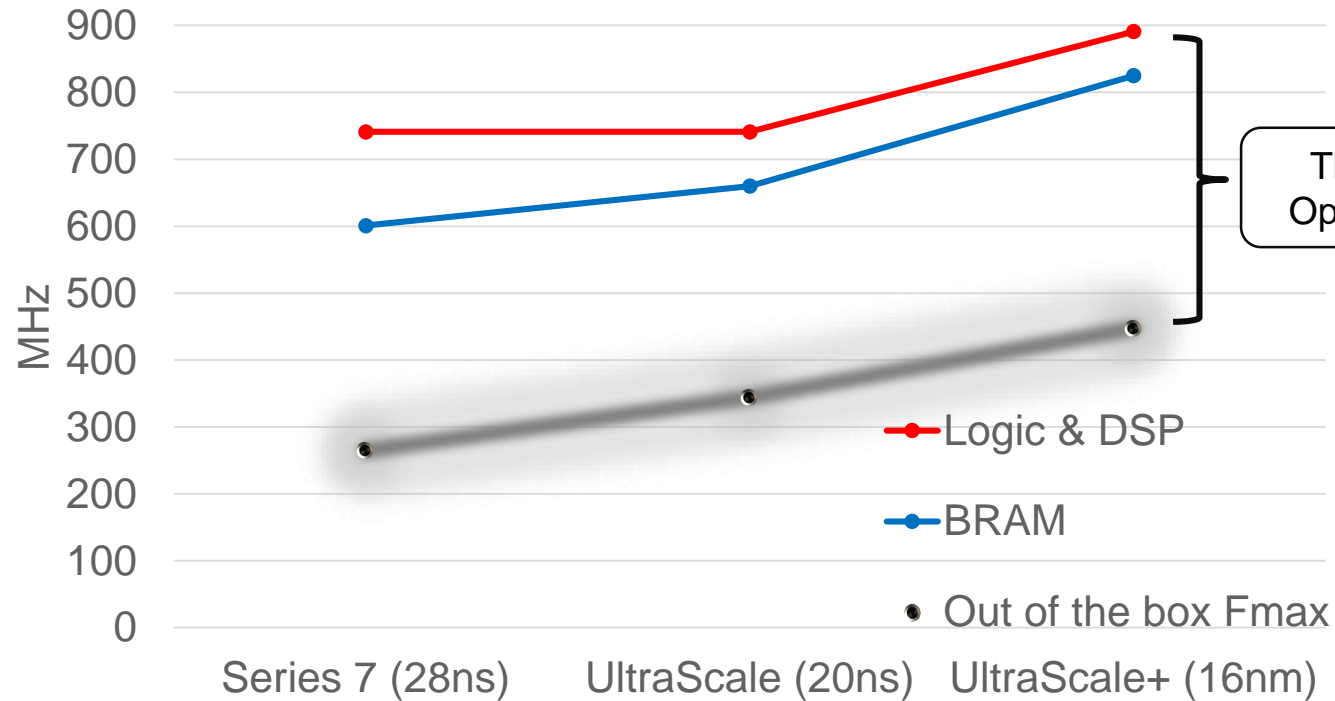
3. P&R modules cached:

- Relocatable
- Reusable
- Timing predictable

4. Run implementation



Building relocatable domain-specific shells



> Fact

>> Advances in silicon have created QoR opportunity

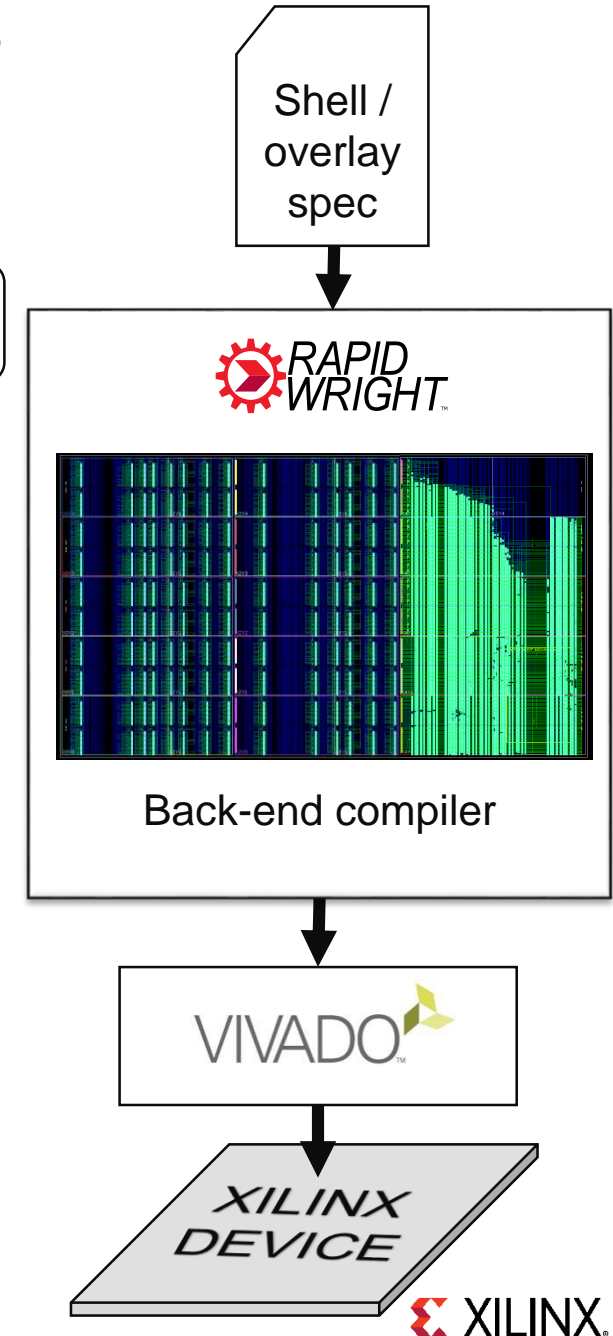
> Community role

>> Domain-specific shell design or overlays

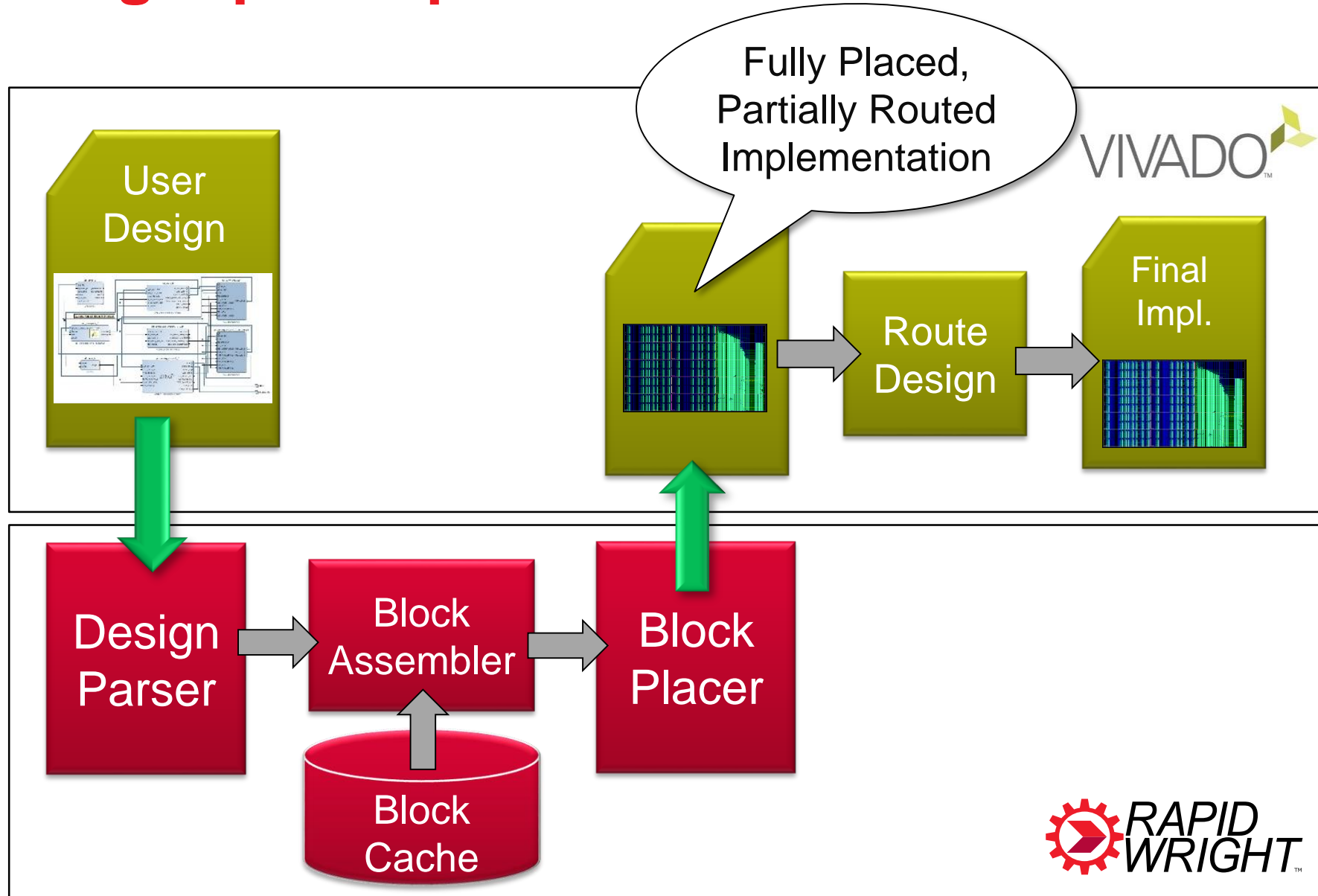
> RapidWright value proposition

>> Achieve near-spec performance

© Copyright 2018 Xilinx



RapidWright pre-implemented module flow



Design performance results

Design	Target Device	Baseline (initial design)	RapidWright ¹ Flow	Gain
Seismic	KU040	270MHz	390MHz	41%
FMA	KU115	270MHz	417MHz	54%
GEMM	KU115	391MHz	462MHz	16%
ML overlay	ZU9EG	368MHz	541MHz	50%

Speed Grade: -2

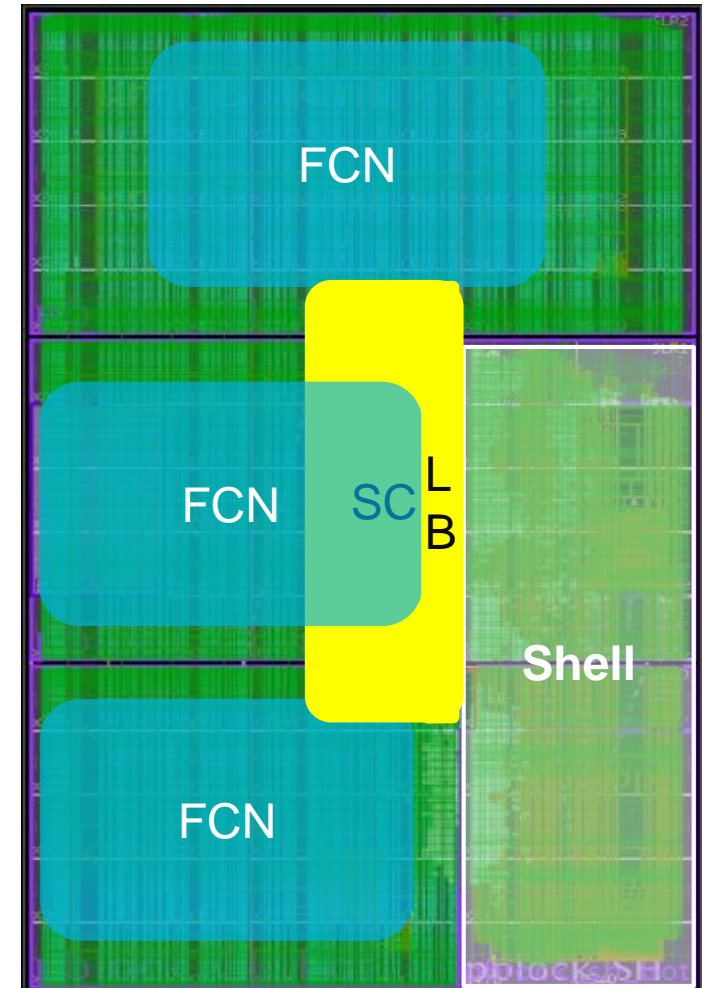
Utilization table

Design	LUT	FF	DSP	BRAM
Seismic	93%	5%	-	-
FMA (HPC design)	25%	50%	97%	6%
GEMM	19%	20%	87%	-
ML overlay	46%	29%	42%	96%

1: RapidWright: Enabling Custom Crafted Implementations for FPGAs, FCCM 2018

Fully Connected Network (FCN) accelerator

- > **Fully Connected Network Accelerator (FCN)**
 - >> GEMM + ReLU (activation function)
 - >> BRAM and DSP Utilization higher than 80%
 - >> Goal: fit four compute kernels on F1
- > **Regular Host Interconnect**
 - >> 2 compute Kernels (@ 200 MHz) fit
 - Three kernels **does not route**, due to overhead of data movement
- > **LinkBlaze¹ Host Interconnect**
 - >> Three kernels (@ 200 MHz) **fully placed & routed**



1: LinkBlaze: Efficient global data movement for FPGAs., Reconfig 2017

Fully Connected Network (FCN) accelerator

> Fully Connected Network Accelerator (FCN)

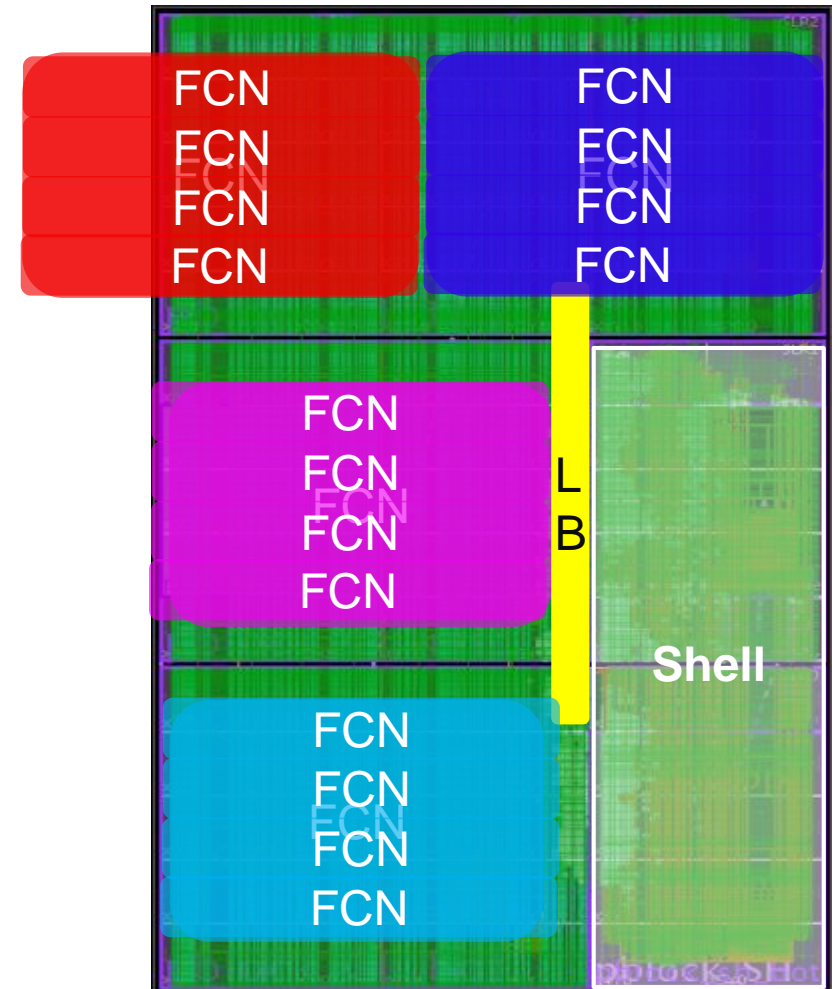
- >> GEMM + ReLU (activation function)
- >> BRAM and DSP Utilization higher than 80%
- >> Goal: fit four compute kernels on F1

> Regular Host Interconnect

- >> 2 compute Kernels (@ 200 MHz) fit
 - Three kernels **does not route**, due to overhead of data movement

> LinkBlaze¹ Host Interconnect

- >> Three kernels (@ 200 MHz) **fully placed & routed**
- >> 4x Kernels with relocatable modular design will fit



Enabling 33% More Compute

1: LinkBlaze: Efficient global data movement for FPGAs., Reconfig 2017

Pre-implemented data movement shell

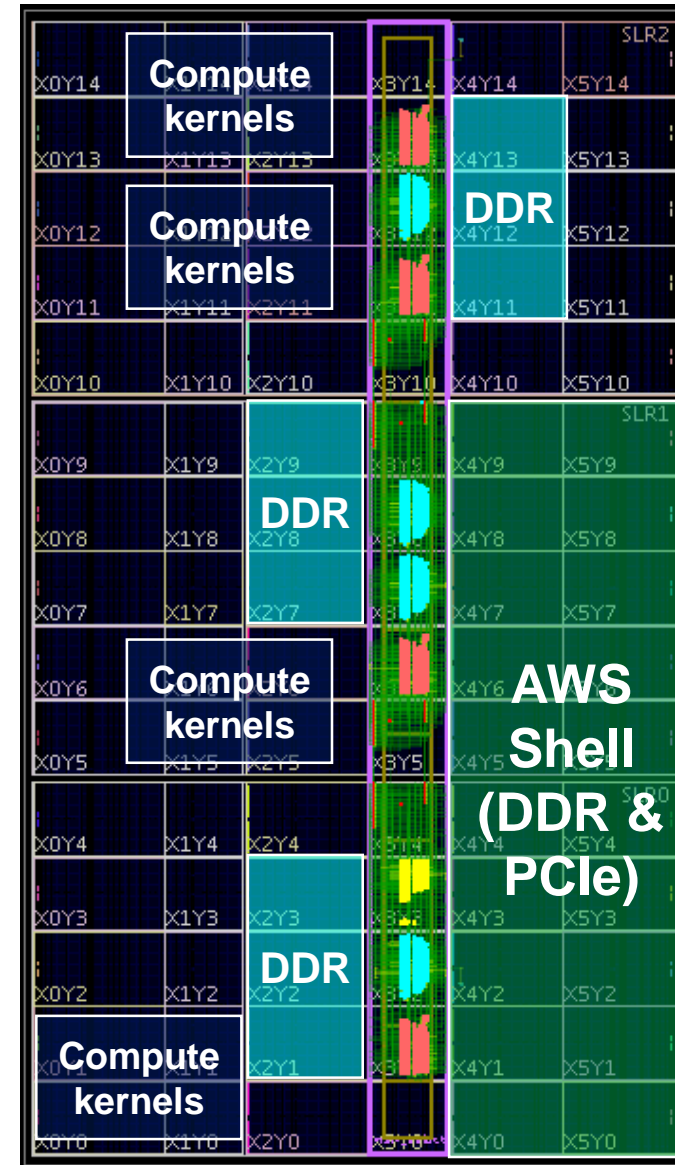
> Goals

- >> Minimize overhead of compute (and overlays)
- >> Prove shell assembly model

> Build-to-order LinkBlaze shell

- >> 512 bit, bi-directional
- >> RapidWright Pre-implemented modules

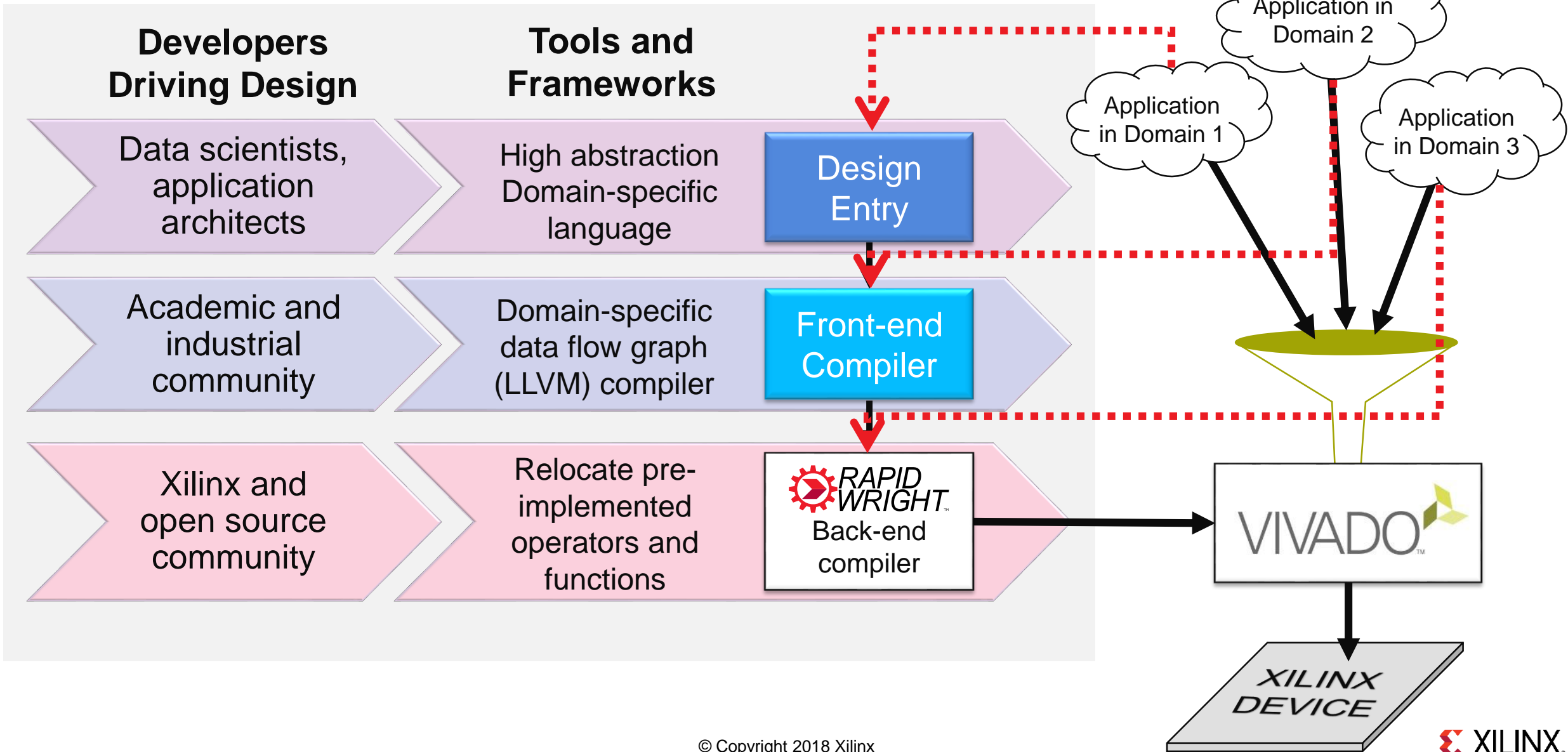
Vivado	RapidWright
516MHz	620MHz (+20%)



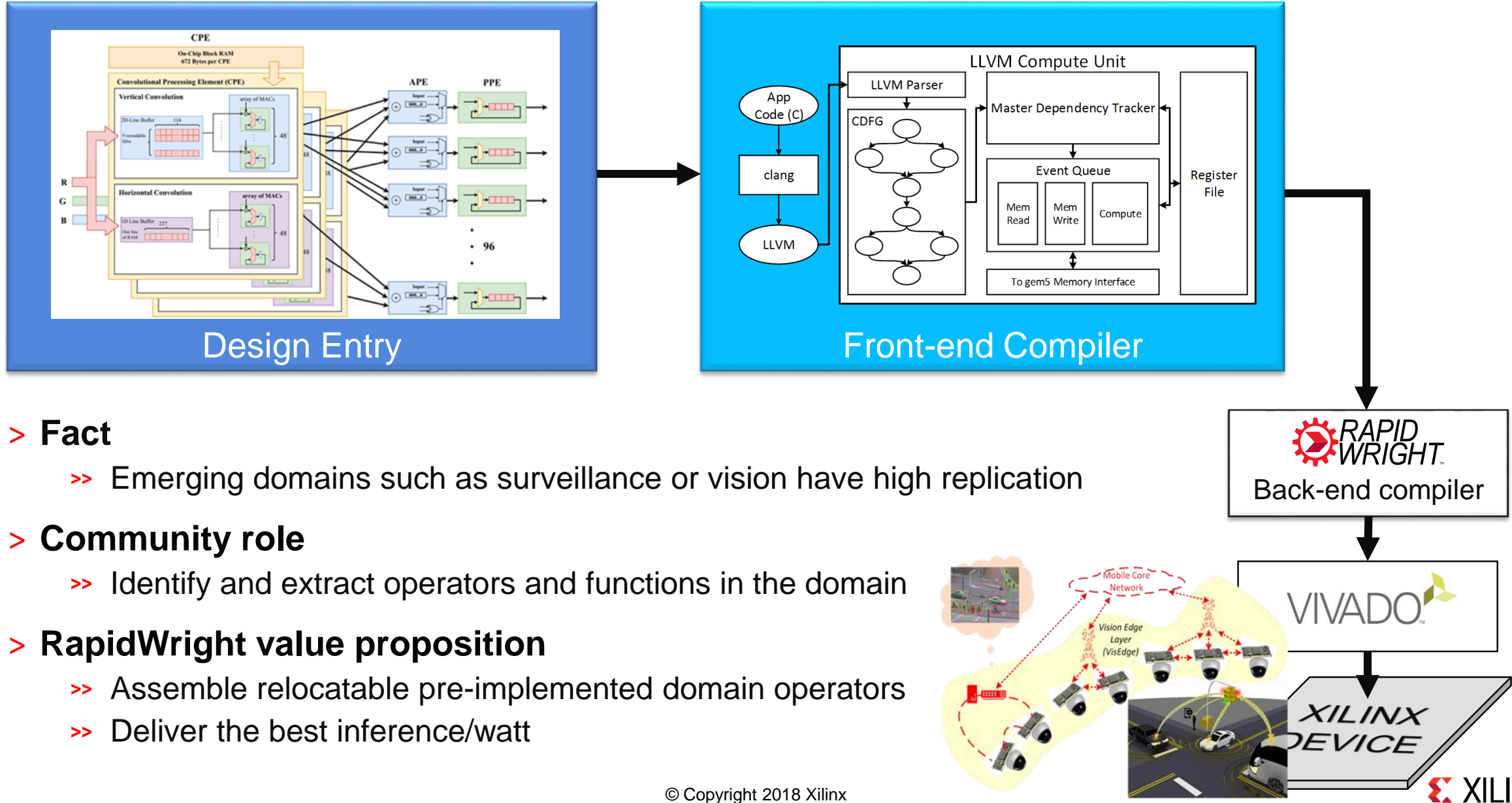
Open Source Community Call for Action



Proposed domain-specific tool flows



Domain tool flow example



> Fact

>> Emerging domains such as surveillance or vision have high replication

> Community role

>> Identify and extract operators and functions in the domain

> RapidWright value proposition

>> Assemble relocatable pre-implemented domain operators

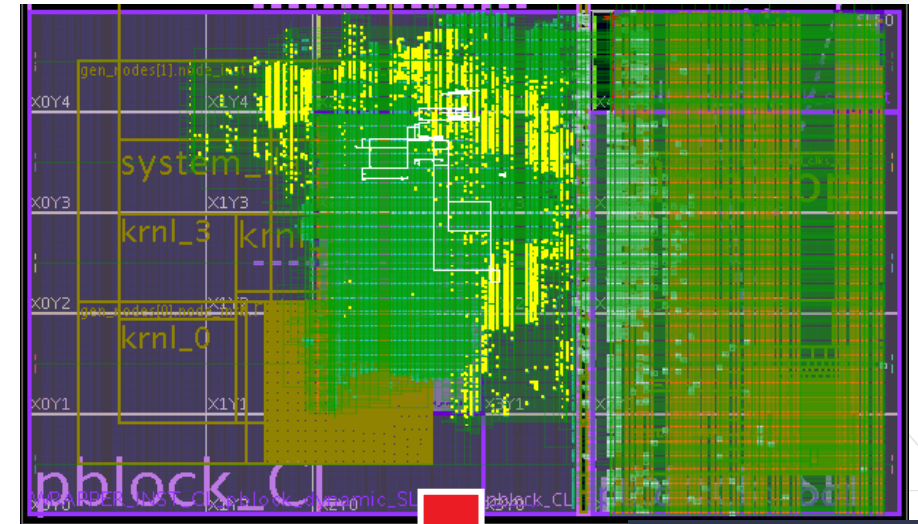
>> Deliver the best inference/watt

Beyond a pre-implemented methodology

> RapidWright probe router enables higher productivity

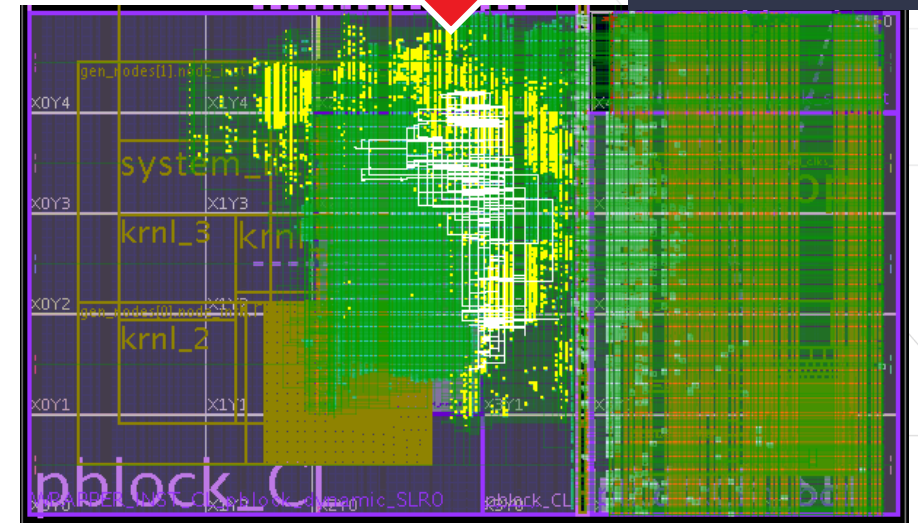
- >> 21X more debug turns per day
- >> Highest level of routing preservation possible
- >> Future innovation:
 - iteration with extra probe inputs
 - Automatic insertion of pipeline flops to manage timing

Vivado modify_debug_probes	RapidWright ProbeRouter	Δ
130 mins	6.3 mins	21X



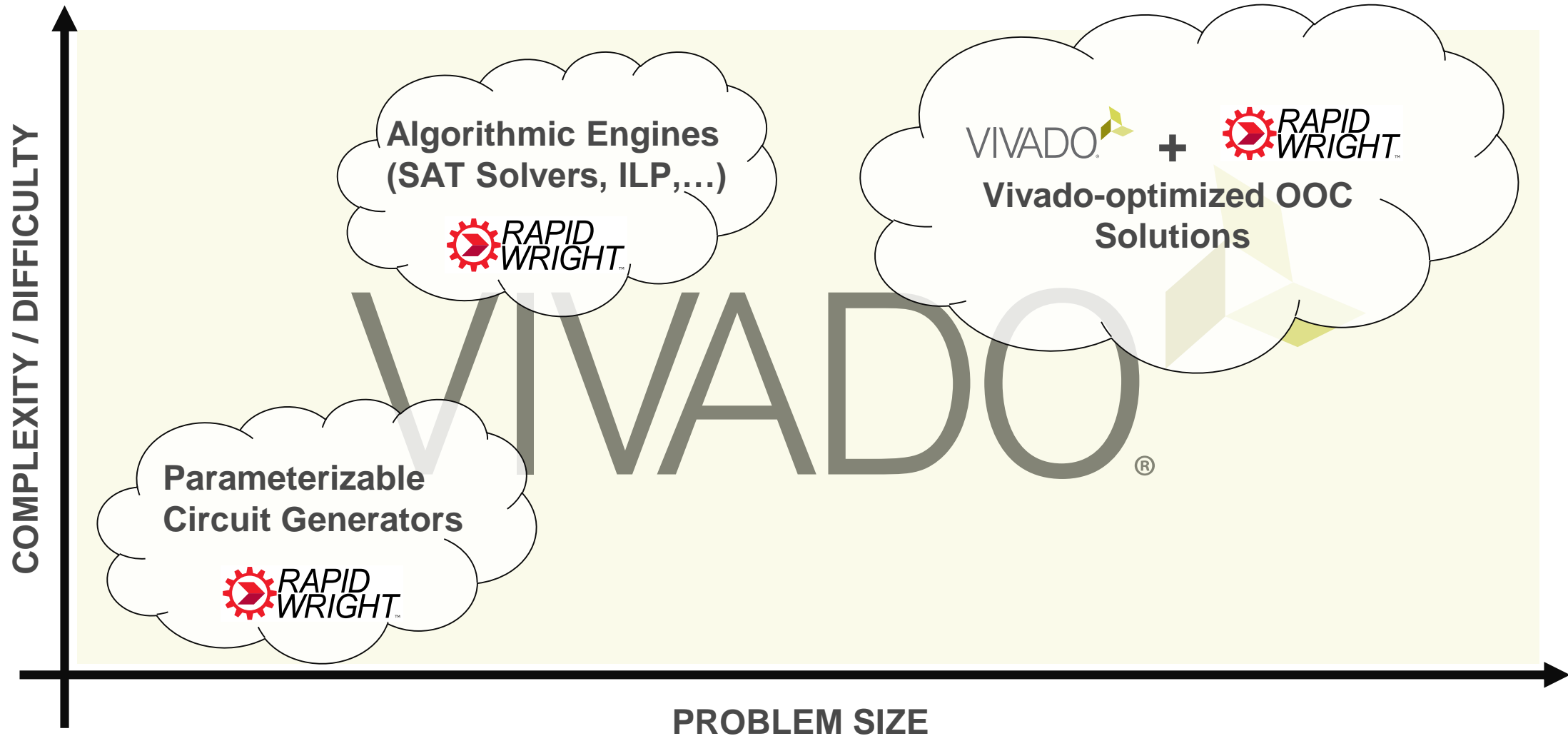
Original

■ ILA Cells
— Probe Routes



RapidWright Probes Rerouted

Vision: Pre-implemented modules



www.rapidwright.io

© Copyright 2018 Xilinx



| UNIVERSITY PROGRAM

Presented By

Hugo Andrade

Director, Xilinx University Program



What we do

Empower academic teaching, research, and entrepreneurship with Xilinx technologies

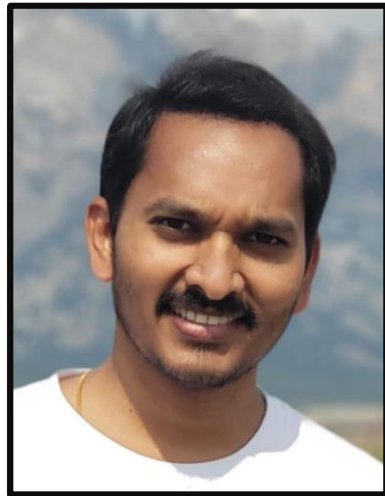


Who we are

- > **Global university engagement**
 - >> Americas/RoW, EMEA, APAC
- > **Americas/RoW and world-wide contacts**



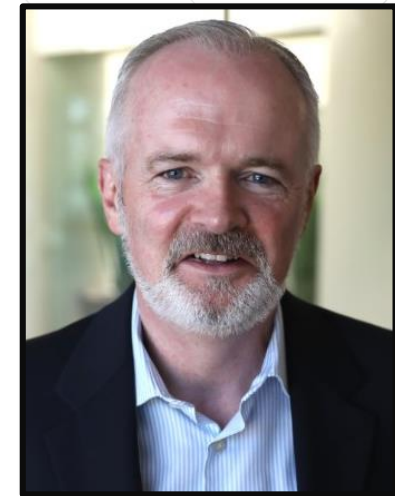
Dr. Parimal Patel



Naveen Purushotham



Hugo A. Andrade



Patrick Lysaght

How we can help

Access to tool and IP licenses, academic boards and chips

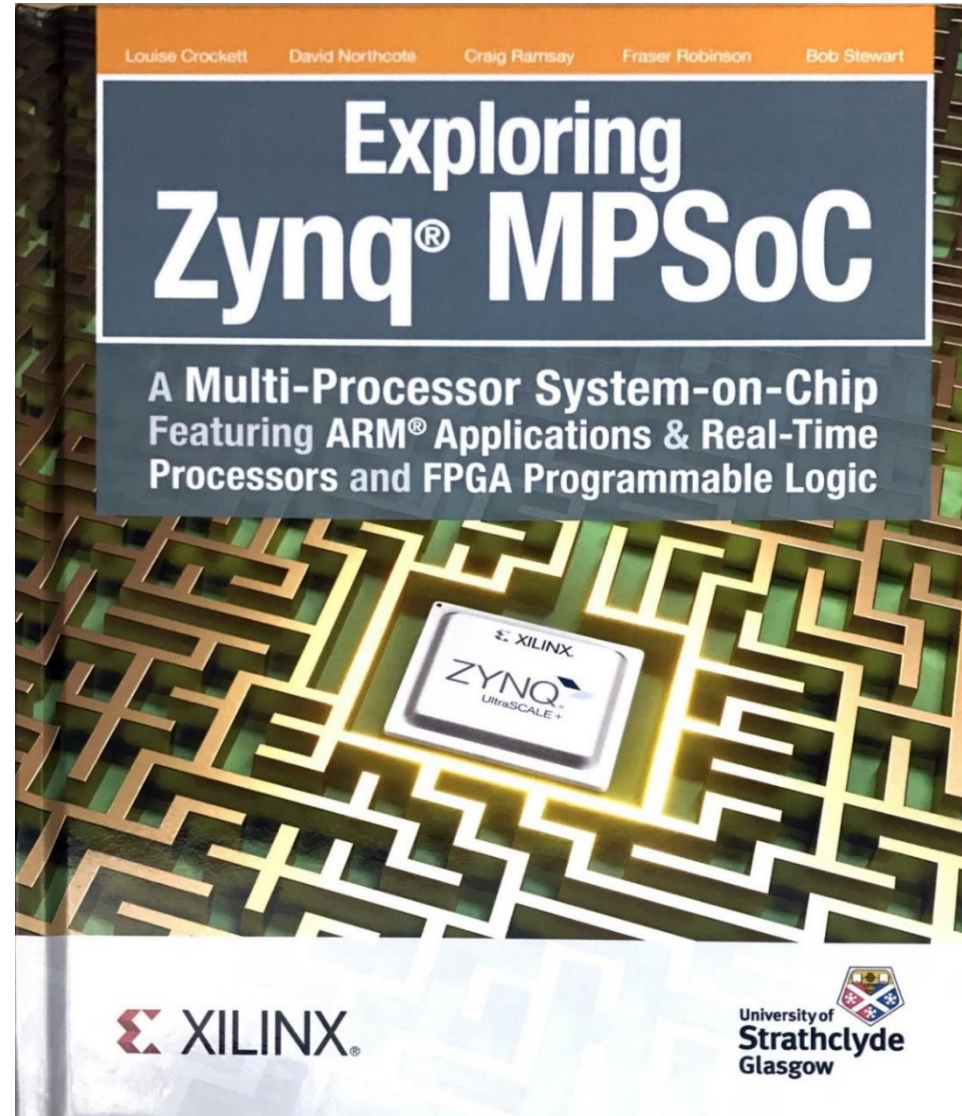
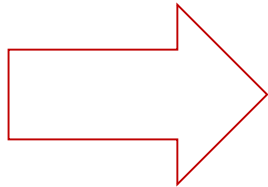
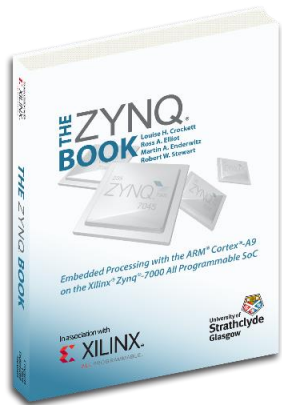
- > Vivado
- > HLS
- > SDx: SDSoC, SDaccel
- > Zynq
- > MPSoC, RFSoc
- > Ultrascale+

- > Research enablement
- > Teaching Material
- > Reference designs
- > Technical Support

- > Conferences
- > Workshops
- > Summer Schools

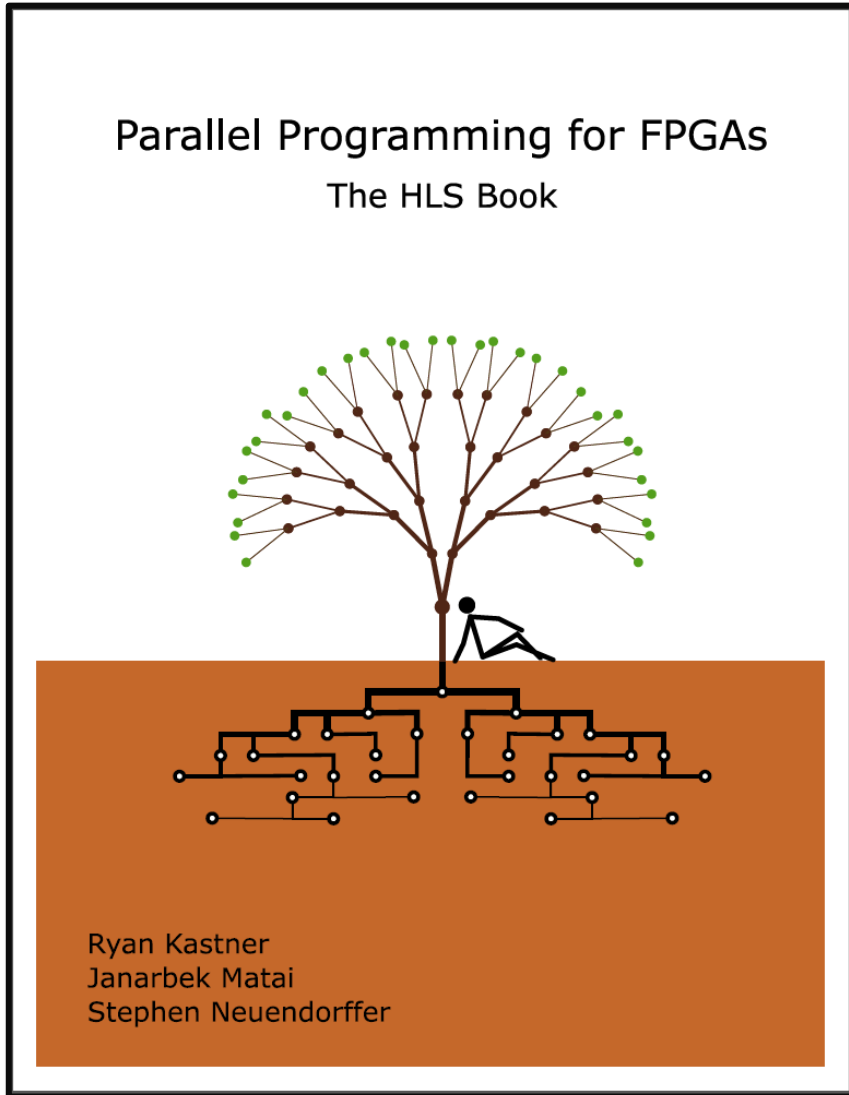
- > Design Contests
- > Hackathon
- > Startup program
- > Cloud access

New Zynq Ultrascale+ MPSoC book with ML



Coming in 2019

New 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



<http://kastner.ucsd.edu/hlsbook/>

Global engagement and collaboration

BYU

MIT HANLAB
Hardware, AI and Neural-nets

AMC: AutoML for Model Compression and Acceleration

Ji Lin^{1*} Yihui He^{2*} Zhijian Liu¹ Hanrui Wang¹ Li-Jia Li³ Song Han¹
¹ Massachusetts Institute of Technology ² Xi'an Jiaotong University ³ Google (* equal contributions)

XDF



Democratizing Customizable Computing via Automated Accelerator Generation

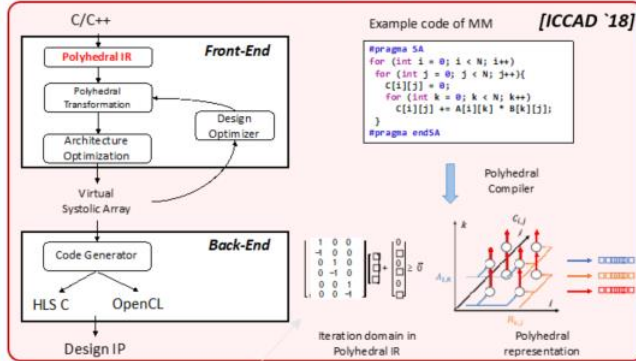
Yuze Chi¹, Jason Cong^{1,2}, Jie Wang¹, Peng Wei¹ and Cody Hao Yu^{1,2}
 University of California Los Angeles¹, Falcon Computing Solutions²

XDF

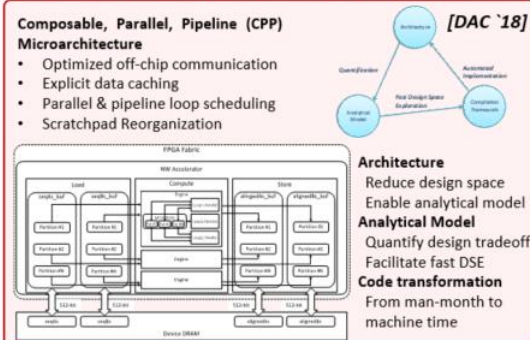
Abstract

- High-performance & ease-of-programming: we want both**
- Designing high-performance accelerators demands a great deal of programming effort
 - Transforming software programs directly into FPGA circuits becomes feasible via HLS, but the QoR is horrible without heavy reconstruction of the software code
- From C to high-quality HLS-C: dealing with large design space**
- A large number of pragmas in many legal insertion points
 - Complex performance-resource trade-offs
 - Long evaluation time (tens of minutes per design points)
- Automated accelerator generation: automatically producing a high-performance design in a reasonable amount of time**
- Using *machine learning* to decrease the number of design points that need to be evaluated
 - Using *microarchitecture templates* to confine the design choices, and, more importantly, enable analytical-model-based fast design space exploration

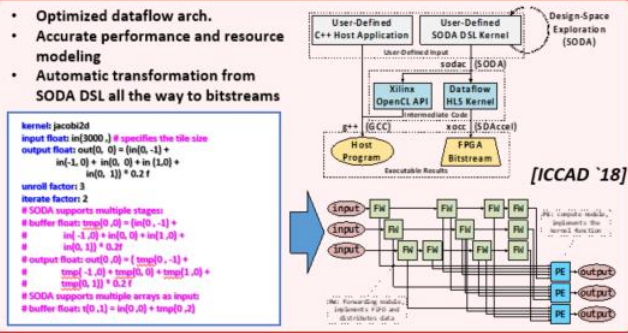
PolySA: Polyhedral-Based Systolic Array Auto-Compilation



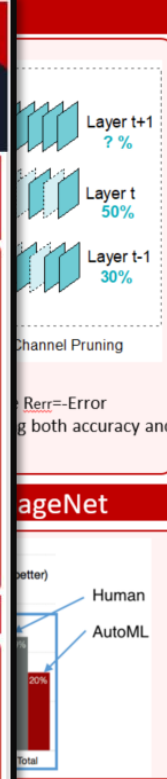
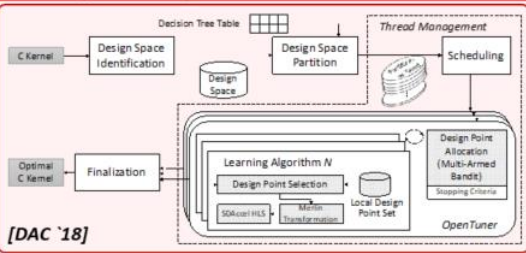
AutoAccel: Automated Accelerator Generation w/ CPP Microarchitecture



SODA: Stencil with Optimized Dataflow Arch.



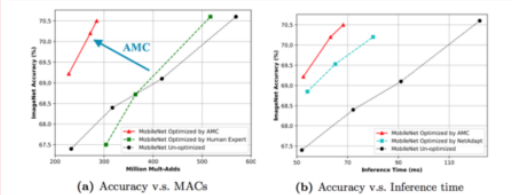
AutoDSE: Learning-Based Design Space Exploration Framework



AMC Details

- DDPG Agent
- DDPG Agent for continuous action space (0-1)
 - Input state embedding of each layer and output sparse ratio
- Compression Methods
- Fine-grained Pruning for model size compression
 - Coarse-grained/Channel Pruning for faster inference
- Search Protocols
- Resource-Constrained Compression to reach a desired compression ratio while getting highest possible performance.
 - Accuracy-Guaranteed Compression to fully preserve the original accuracy while maintain smallest possible model size.

Speedup Mobile-Net on ImageNet



Pros and Cons of Approaches

- Learning-based approach**
- Not being constrained to a specific microarchitecture
 - Demanding a great deal of time to achieve the optimal solution due to time-consuming design point evaluation
- Microarchitecture-based approach**
- Coming up with a high-quality design in much shorter time if the input program fits into the *microarch* well
 - May not work well for all kinds of compute domains

References

Cong et al, Automated Accelerator Generation and Optimization with Composable, Parallel and Pipeline Architecture, DAC '18
 Yu et al, S2FA: An Accelerator Automation Framework for Heterogeneous Computing in Datacenters, DAC '18
 Chi et al, SODA: Stencil with Optimized Dataflow Architecture, ICCAD '18
 Cong et al, PolySA: Polyhedral-Based Systolic Array Auto-Compilation, ICCAD '18

UNIVERSITY of VIRGINIA



Key Initiatives

- FPGA-based accelerators in the cloud
- PYNQ: Python productivity for Zynq



Promote & support AWS EC2 F1 in academic community

- > **An AWS EC2 compute instance with Xilinx FPGAs which can be programmed to create custom hardware accelerated applications**
- > **F1 instances are easy to program and come with everything needed to develop, simulate, debug, and compile hardware accelerators**
- > **Can be registered as an Amazon FPGA Image (AFI) and marketed**



Latest technology training at top conferences

FPGA18, Monterey

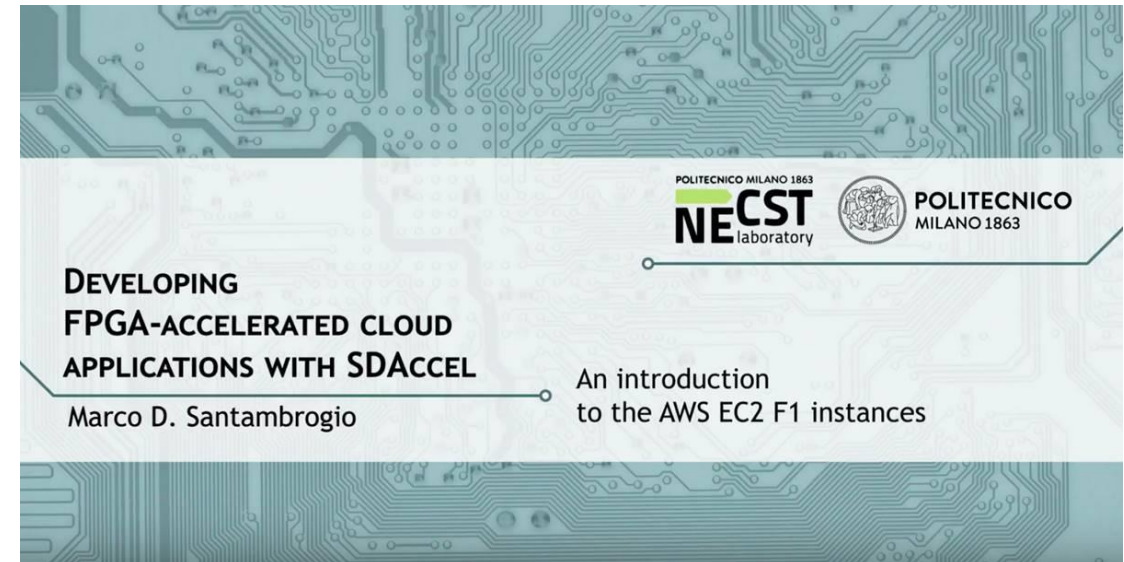


Expanded conference categories covered

- > **FPGA**
- > **Computer Architecture and HPC**
- > **Applications**

Promote & support AWS EC2 F1 based courses

- > **UC Berkeley, Prof. Krste Asanović**
 - >> Computer Architecture & Engineering
 - >> “An important part is lab assignments using real microprocessor designs implemented in Chisel, running as simulators and FPGA emulators in the Amazon cloud as F1 instances.”
- > **Cornell, Prof. Zhiru Zhang**
 - >> High-level Digital Design Automation
- > **UCLA, Prof. Jason Cong**
 - >> Customizable Computing for Big Applications
 - >> Parallel and Distributed Computing
- > **Coursera, Politecnico di Milano, Prof. Marco Domenico Santambrogio**
 - >> FPGA-accelerated Cloud Applications with SDaccel



Let's chat further about:

- Opportunities in teaching, research and entrepreneurship using AWS EC2 F1
- How to get AWS credit vouchers
- Hand-on training



Updated 2018.x workshop material now on PYNQ boards

Advanced Embedded System Design on Zynq using Vivado

Course Description	This workshop provides professor the necessary skills to develop complex embedded systems using Vivado design suite; understand and utilize advanced development techniques of embedded systems design for architecting a complex system in the Zynq® System on a Chip (SoC).
Level	Intermediate
Duration	2 Days
Who should attend?	Professors who are familiar with embedded system design using Vivado and want to explore advanced design techniques using Xilinx SoC in Zynq.
Pre-requisites	<ul style="list-style-type: none">• Digital logic and FPGA design experience• Experience with Vivado software• Experience with Embedded System design• Have attended XUP Embedded System Design workshop or has an equivalent experience

Skills Gained

After completing this workshop, you will be able to:

- Assemble an advanced embedded system
- Explore various features of Zynq Soc for hardware-software co-design
- Design and integrate peripherals using interrupts
- Analyze system performance
- Utilize hardware debugging technique
- Design a bootable system ready for deployment in field

Course Overview

Quick Links

- [Workshops Schedule](#)
 - [Vivado Design Suite](#)
 - [Vivado-Based Workshops](#)
 - [ISE Design Suite](#)
 - [ISE-Based Workshops](#)
- [+ More](#)

2018x Workshop Material

Common to PYNQ-Z1 and PYNQ-Z2

- [Labdocs \(PDF\)](#)
- [Lab Source File](#)
- [Labdocs and Presentation \(docx and pptx\)*](#)

PYNQ-Z1

- [README](#)
- [Board Files \(required to do the labs\)](#)
- [Labsolution*](#)

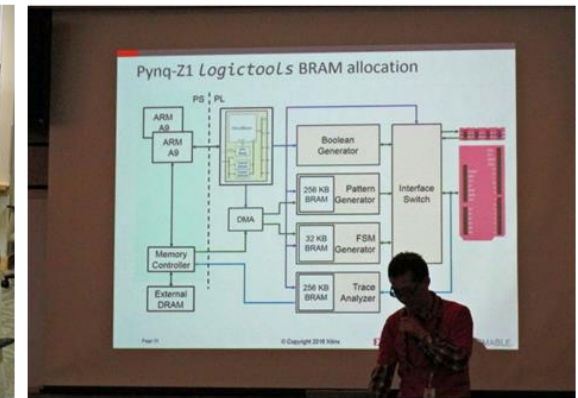
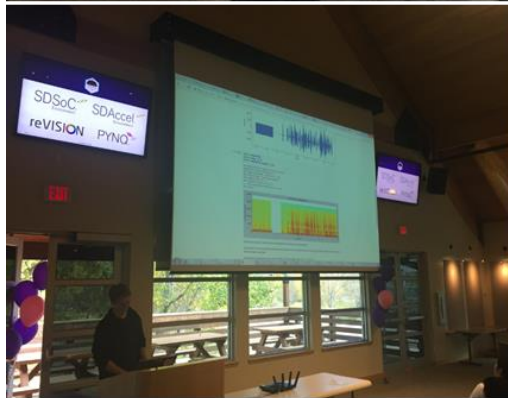
PYNQ-Z2

- [README](#)
- [Board Files \(required to do the labs\)](#)
- [Labsolution*](#)

Promote and support PYNQ hackathons



Industry and academia



Reproducible Hacking with Jupyter Notebooks

Reaching XUP

www.xilinx.com/xup

xup@xilinx.com



 **XDF** XILINX
DEVELOPER
FORUM

 **XILINX**
| UNIVERSITY PROGRAM

 **XILINX**[®]

PYNQ[™] 