



XILINX Open-Source Research

Cathal McCabe
Xilinx University Program
WRC HiPEAC 20 January 2020



Overview

- ▶ Open Source at Xilinx
- ▶ PYNQ
- ▶ FINN
- ▶ RapidWright



Open Source at Xilinx



Xilinx

GitHub.Com/Xilinx/

San Jose, CA <http://www.xilinx.com>

Repositories 140

Packages

People 222

Teams 35

Projects

Pinned repositories

 **linux-xlnx**

The official Linux kernel from Xilinx

 C  649  947

 **u-boot-xlnx**

The official Xilinx u-boot repository

 C  309  537

 **PYNQ**

Python Productivity for ZYNQ

 Jupyter Notebook  832  474

 **XRT**

Xilinx Run Time for FPGA

 C++  131  163

 **FPGA_as_a_Service**

 Go  54  16

 **Vitis_Libraries**

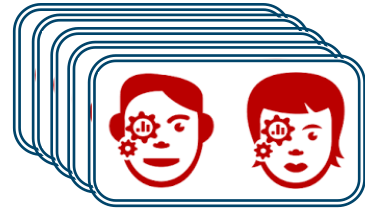
Vitis Libraries

 C++  126  40



PYNQ: Python productivity for Zynq

PYNQ™ Python Productivity on Zynq



Domain Experts
Software Engineers

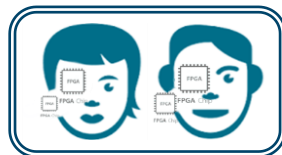
New users will not all be hardware designers,
or embedded systems designers



Embedded software
Engineers

PYNQ™

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

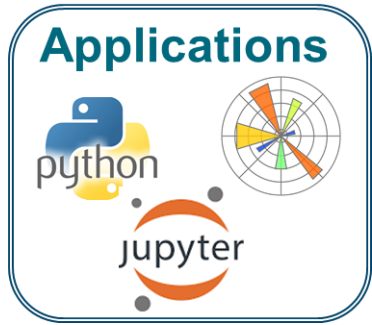


Hardware
Engineers

AND

*Offers more rapid development for hw
designers and embedded sw engineers*

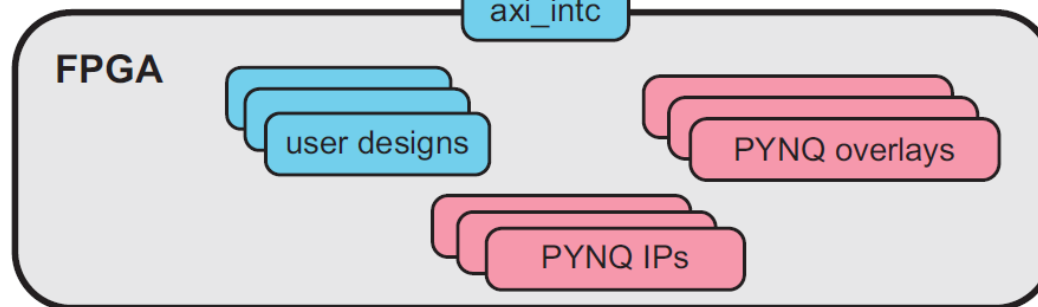
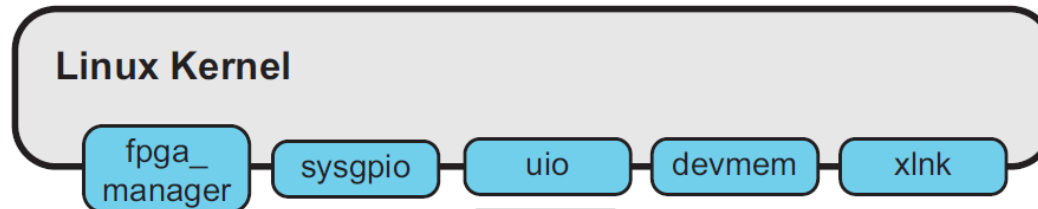
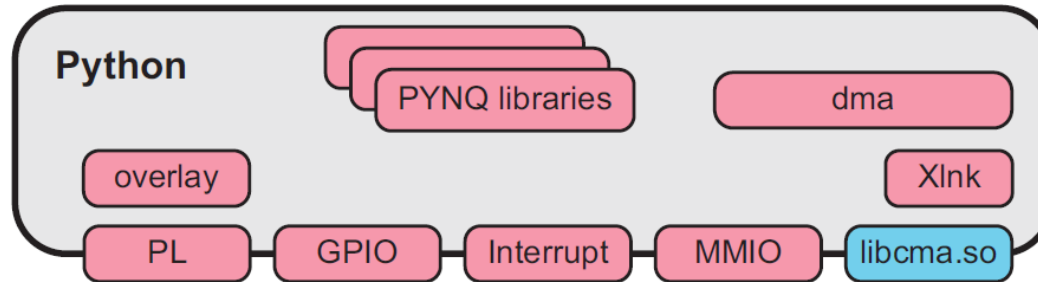
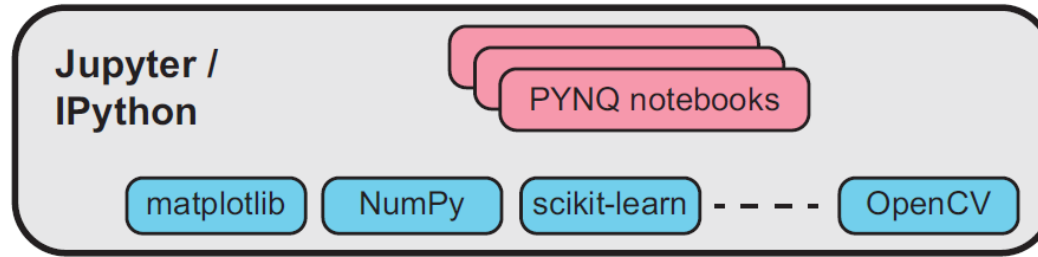
PYNQ™ is an open source *framework*



Applications

Software

Hardware



Apps

APIs

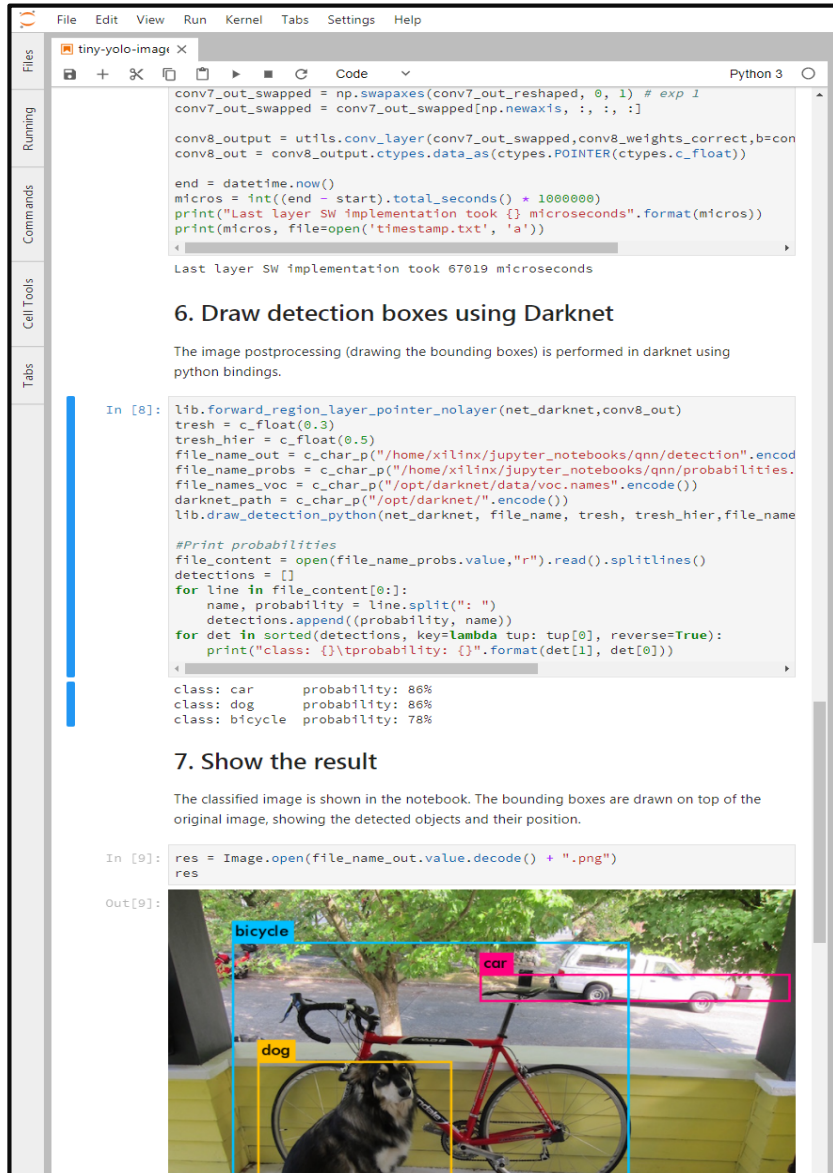
Drivers

Bitstreams

PYNQ™

Jupyter Notebooks: reproducible science

2017 ACM
Software System Award



```
conv7_out_swapped = np.swapaxes(conv7_out_resshaped, 0, 1) # exp 1
conv7_out_swapped = conv7_out_swapped[np.newaxis, :, :, :]

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

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

Last layer SW implementation took 67019 microseconds
```

6. Draw detection boxes using Darknet

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

```
In [8]: lib.forward_region_layer_pointer_nolayer(net_darknet, conv8_out)
tresh = c_float(0.3)
tresh_hier = c_float(0.5)
file_name_out = c_char_p("/home/xilinx/jupyter_notebooks/qnn/detection".encode())
file_name_probs = c_char_p("/home/xilinx/jupyter_notebooks/qnn/probabilities.".encode())
file_names_voc = c_char_p("/opt/darknet/data/voc.names".encode())
darknet_path = c_char_p("/opt/darknet/".encode())
lib.draw_detection_python(net_darknet, file_name, thresh, thresh_hier, file_name)

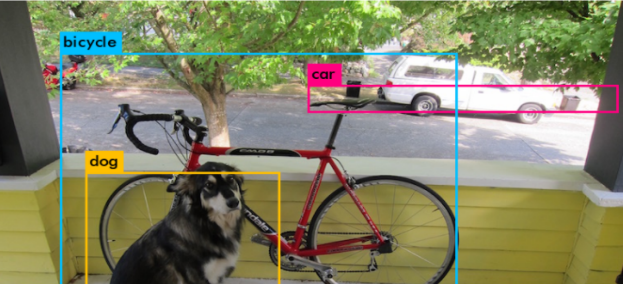
#Print probabilities
file_content = open(file_name_probs.value, "r").read().splitlines()
detections = []
for line in file_content[0:]:
    name, probability = line.split(": ")
    detections.append((probability, name))
for det in sorted(detections, key=lambda tup: tup[0], reverse=True):
    print("class: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

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

7. Show the result

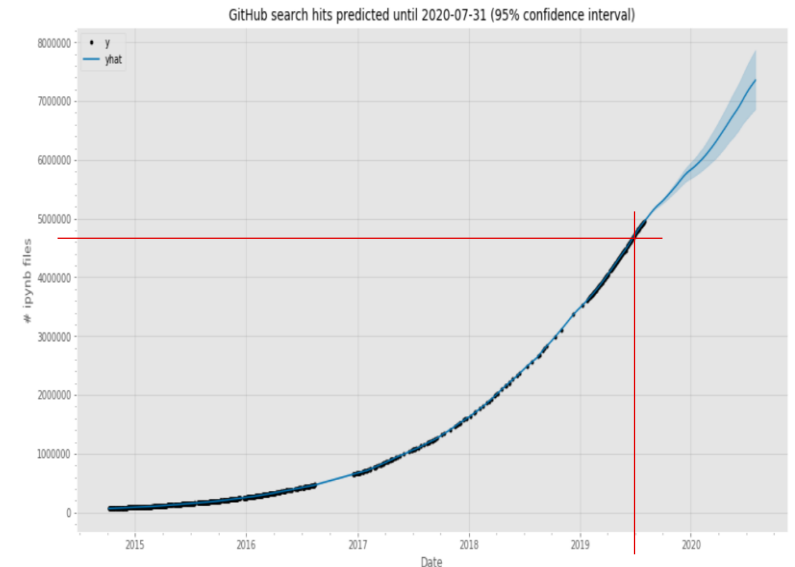
The classified image is shown in the notebook. The bounding boxes are drawn on top of the original image, showing the detected objects and their position.

```
In [9]: res = Image.open(file_name_out.value.decode() + ".png")
res
```



“Google Docs of data science”

Exponential adoption:
> 4 million notebooks
on GitHub



<https://nbviewer.jupyter.org/github/parente/nbestimate/blob/master/estimate.ipynb>

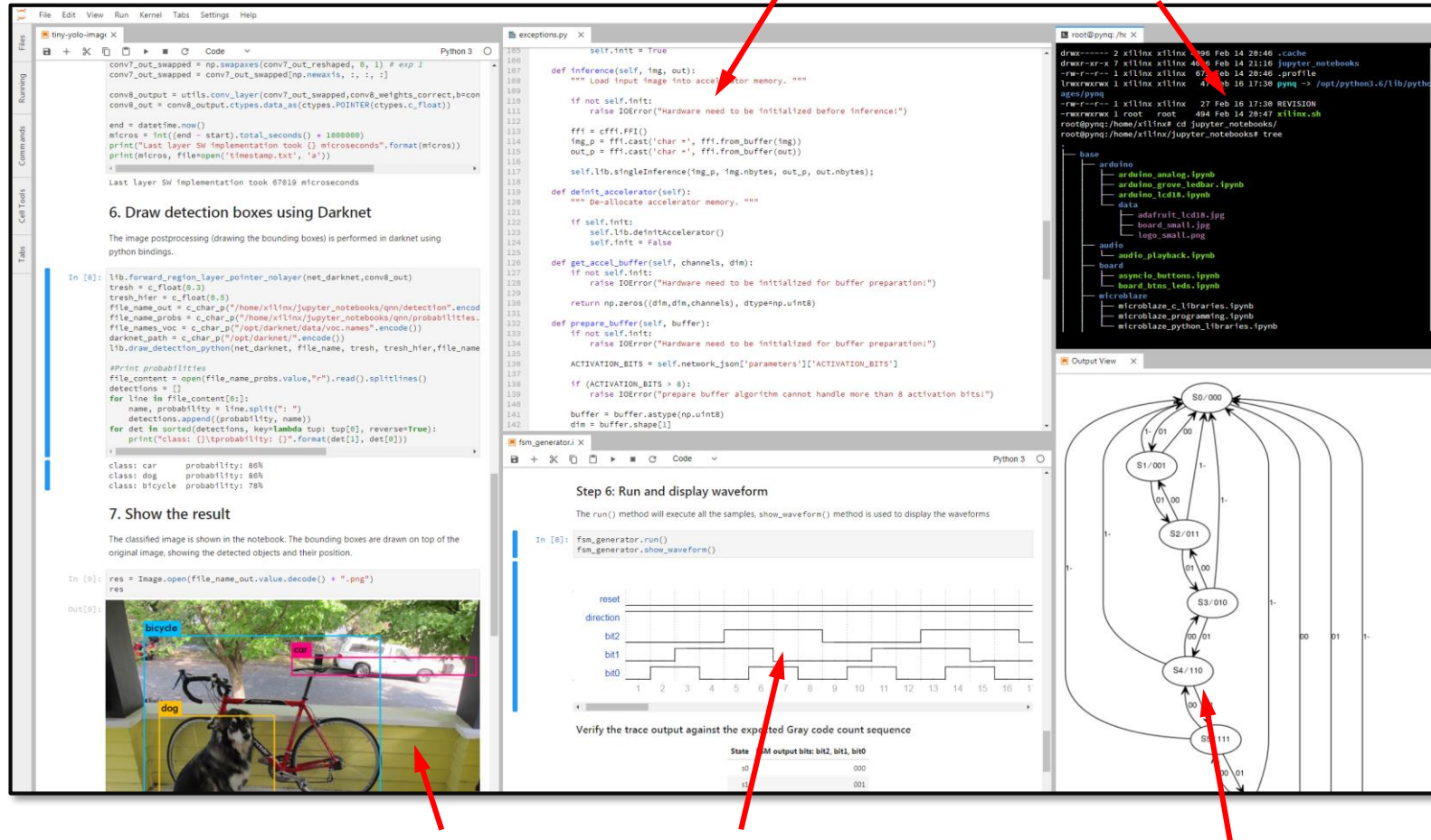
Used by IBM, AWS, Microsoft, Bloomberg, Netflix,
plus thousands of other companies ...

Jupyter Notebooks to JupyterLab IDE



Code editor

Terminal



Jupyter notebooks

Visualization

Next-gen browser IDE

Includes Jupyter Notebooks

Completely open source

Based on web standard

Fully extensible

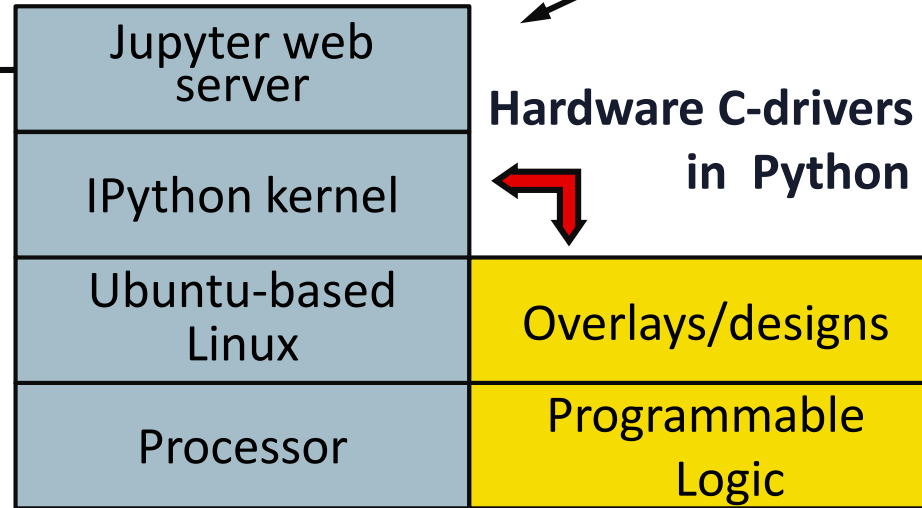
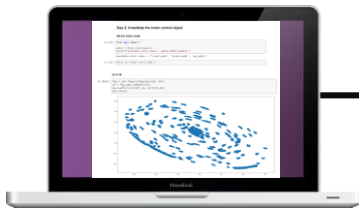




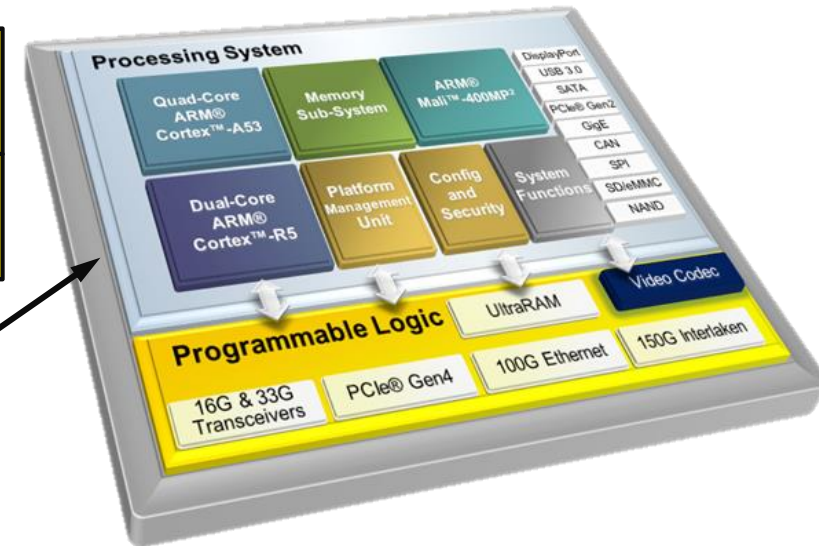
Python productivity for Zynq

Jupyter notebooks,
browser-based interface

PYNQ enables JupyterLab
on Zynq and ZU+, RFSoc and now Alveo

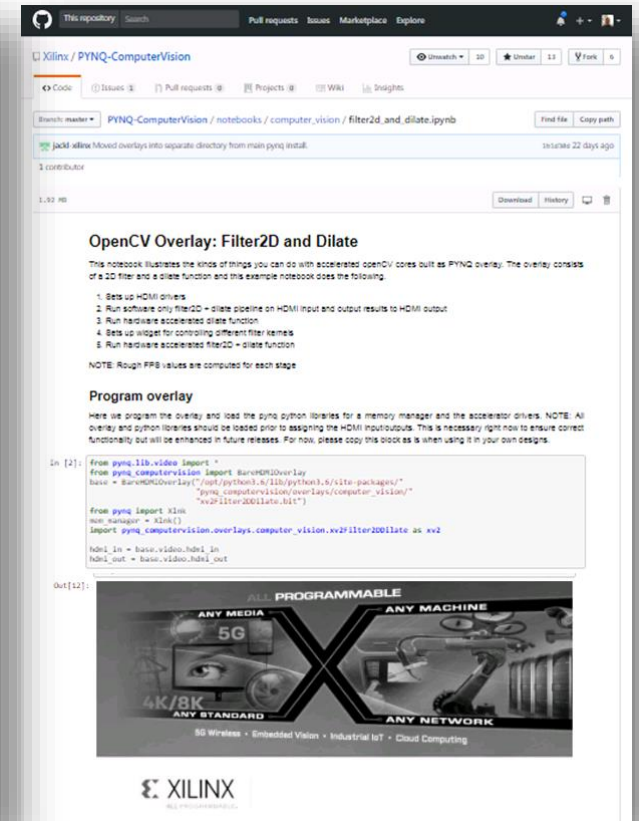
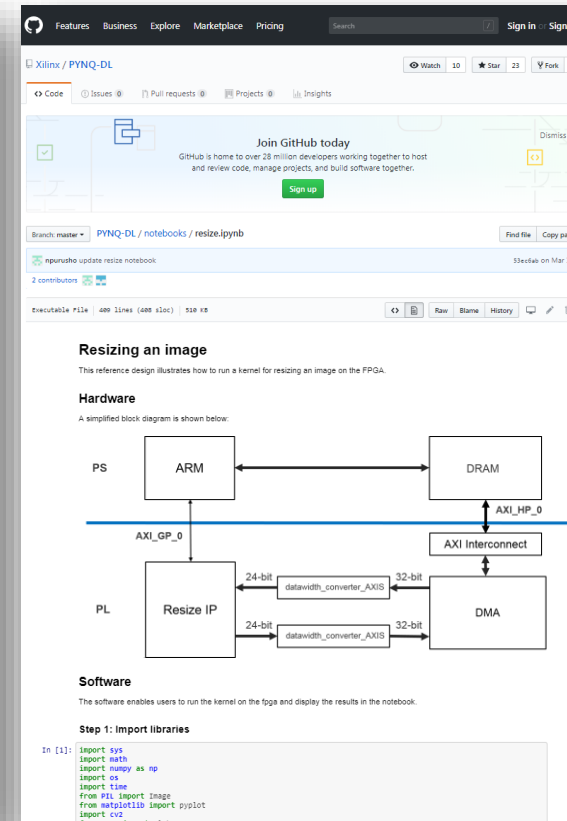
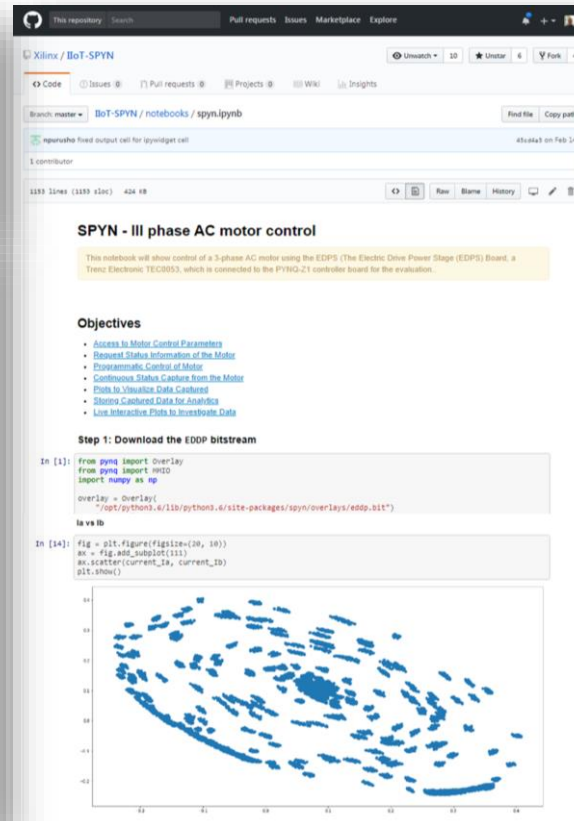
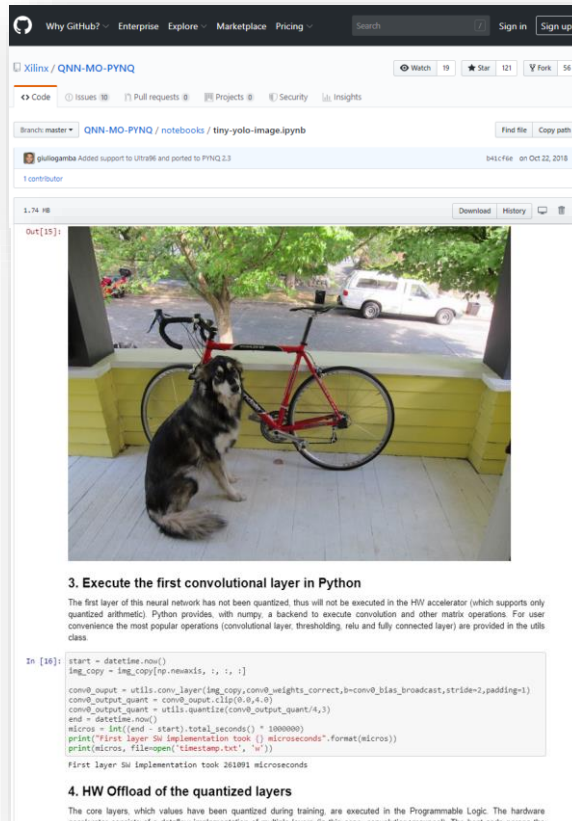


PYNQ runs natively on
processor



Software-style Packaging & Distribution of Overlays

Enabled by new *hybrid packages*



Download the design from GitHub with a single Python command:

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

'Resizer' Notebooks

Software only resizer

Display the image

```
[4]: input_image = Image.fromarray(input_array)
display(input_image)
old_width, old_height = original_image.size
print("Image size: {}x{} pixels.".format(old_width, old_height))
```



Image size: 640x360 pixels.

Resizing in Software

We will set image resize dimensions and use `resize()` method from the PIL library.

```
[5]: resize_factor = 2
new_width = int(old_width/resize_factor)
new_height = int(old_height/resize_factor)

resized_image = original_image.resize((new_width, new_height))
```

Display resized image

```
[6]: output_array = np.array(resized_image)
result = Image.fromarray(output_array)
display(result)
print("Resized image size: {}x{} pixels.".format(new_width, new_height))
```



Resized image size: 320x180 pixels.

100 loops, best of 3: 10.3 ms per loop

Hardware-accelerated resizer

Display the image

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



Image size: 640x360 pixels.

Resizing in Hardware

We will set image resize dimensions and use `resize()` method from the PIL library.

```
[11]: resizer.write(0x10, old_height)
resizer.write(0x18, old_width)
resizer.write(0x20, new_height)
resizer.write(0x28, new_width)

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

run_kernel()
```

Display resized image

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

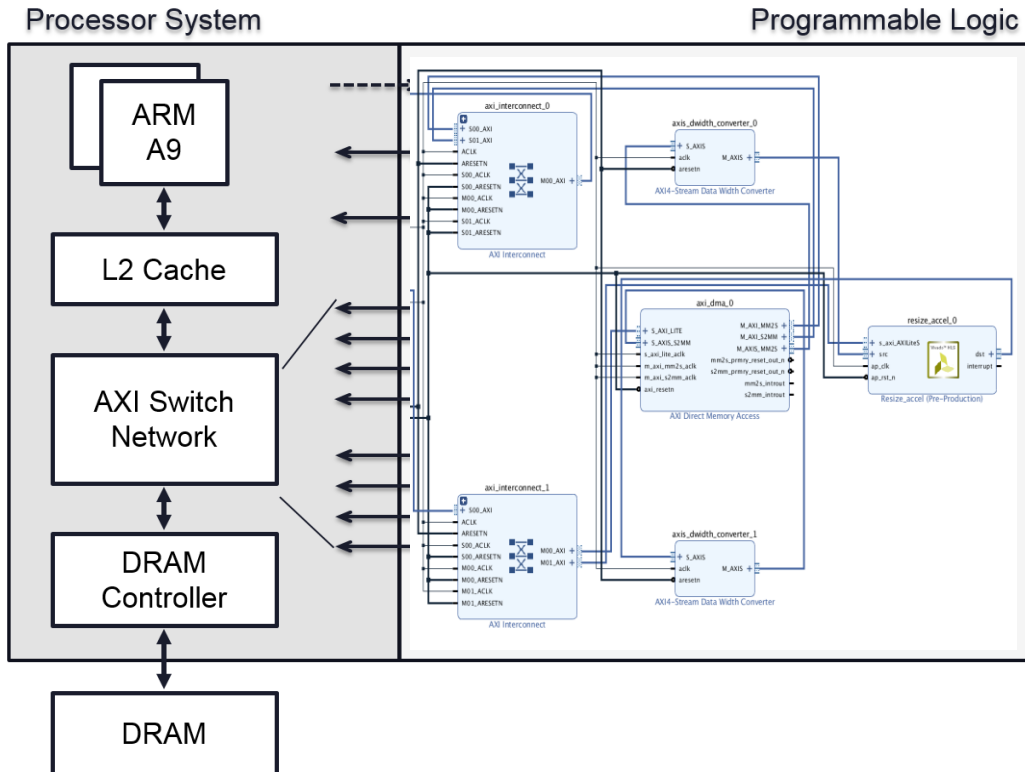


Resized in Hardware(PL): 320x180 pixels.

100 loops, best of 3: 3.09 ms per loop

Load the downloaded resizer design into Zynq

Zynq device



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

PYNQ's self-aware overlays

- Overlay bitstream **and** its metadata are loaded
- Creates a Python dictionary of the metadata
 - IP, memory-map, registers, drivers ...
- IPs without Linux drivers are automatically assigned MMIO drivers & Python APIs

PYNQ™ on Alveo



PYNQ runs on Alveo Accelerators
in the Data Center

```
[ ]: import pynq
    from pynq_alveo_demo.overlays import VADD

    ol = pynq.Overlay(VADD)
    add = ol.krn1_vadd_1
```

...

```
[2]: allocator = pynq.Xlnk()

    in_a = allocator.cma_array((1024,1024), 'u4')
    in_b = allocator.cma_array((1024,1024), 'u4')
    out_c = allocator.cma_array((1024,1024), 'u4')
```

...

```
[3]: import numpy as np

    in_a[:] = np.random.randint(low=0, high=100, size=(1024,1024), dtype='u4')
    in_b[:] = 200
```

...

```
[4]: %%timeit
    in_a.flush()
    in_b.flush()

    add.call(in_a, in_b, out_c, 1024*1024)
    add.wait()

    out_c.invalidate()
```

...

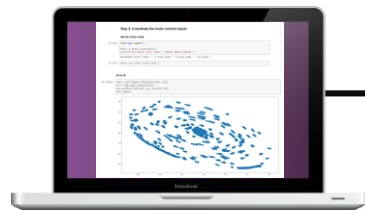
...

```
[5]: np.array_equal(out_c, in_a + in_b)
```

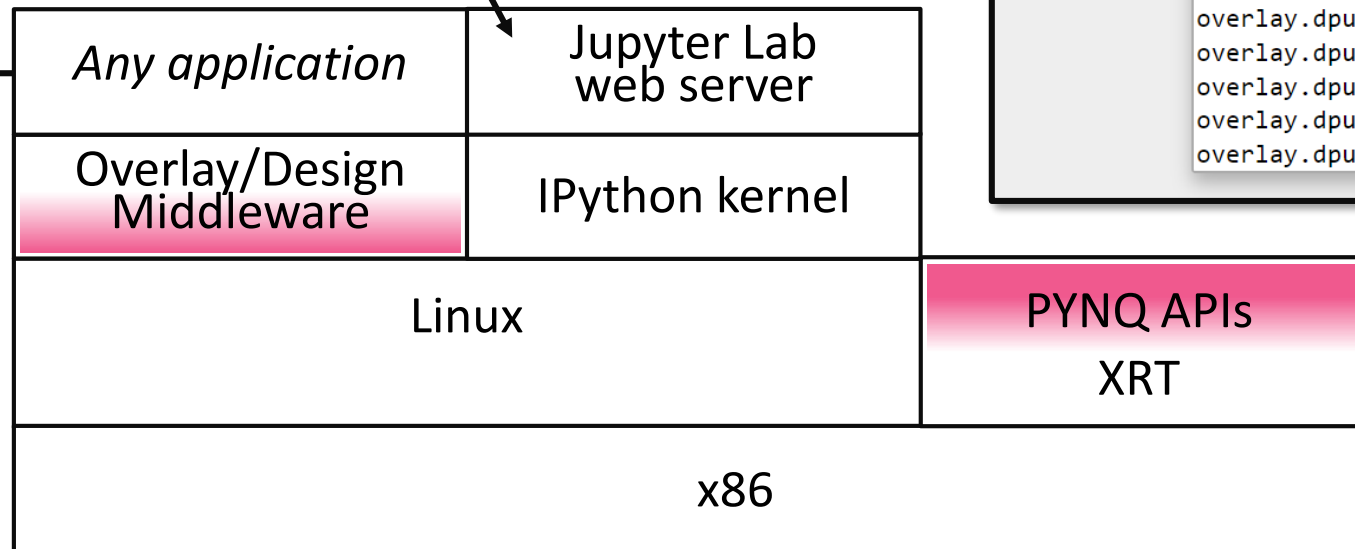
...



PYNQ Overlay Developer Run-time



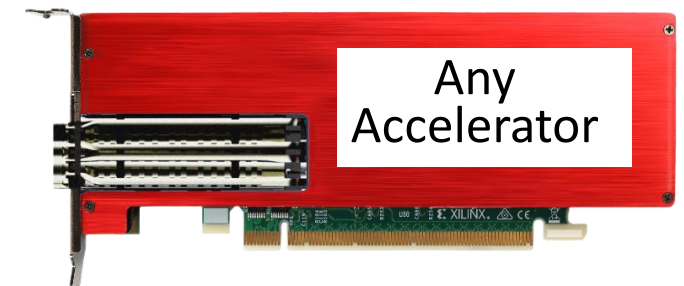
JupyterLab: Browser IDE
Including Notebooks



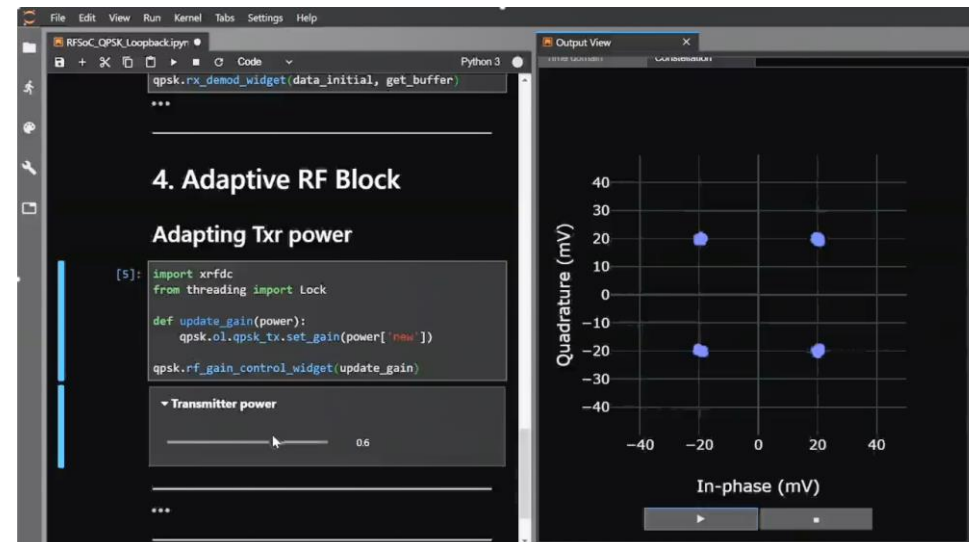
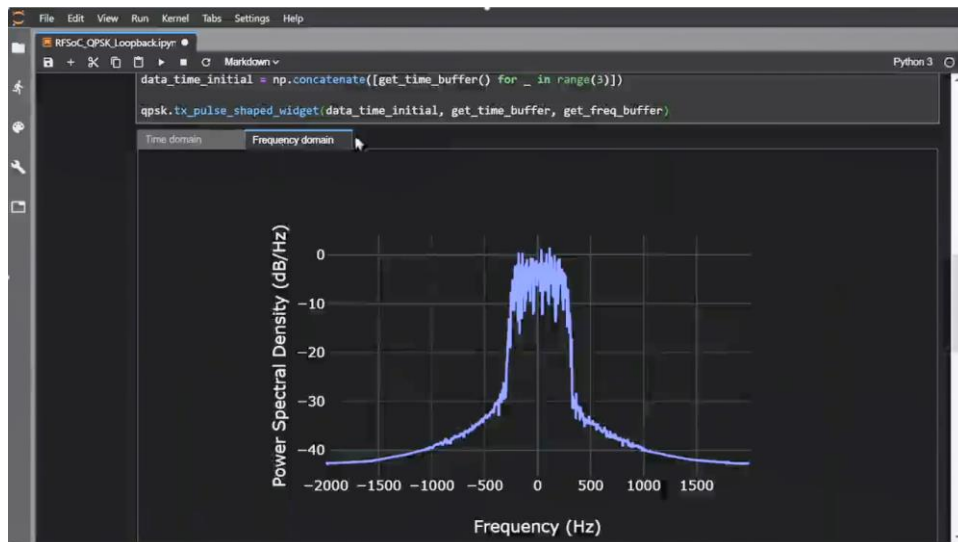
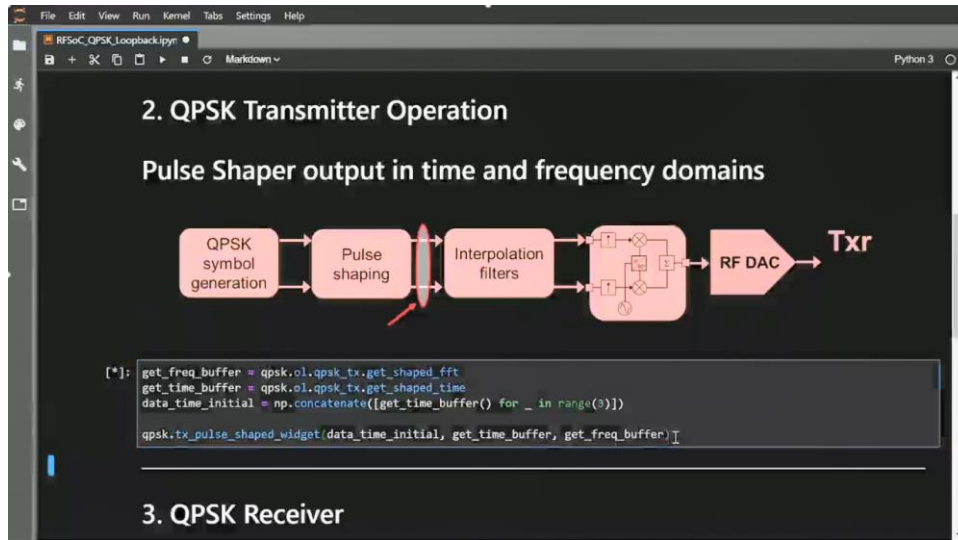
```
In [1]: from pynq import Overlay
        overlay = Overlay('object_recognition.bit')

In [ ]: overlay.dpu_0.
        overlay.dpu_0.call
        overlay.dpu_0.device
        overlay.dpu_0.interrupt
        overlay.dpu_0.mmio
        overlay.dpu_0.read
        overlay.dpu_0.register_map
        overlay.dpu_0.signature
        overlay.dpu_0.start
        overlay.dpu_0.start_sw
        overlay.dpu_0.write
```

Design.
xclbin



Visualization of Instrumented Accelerators (RFSoc)



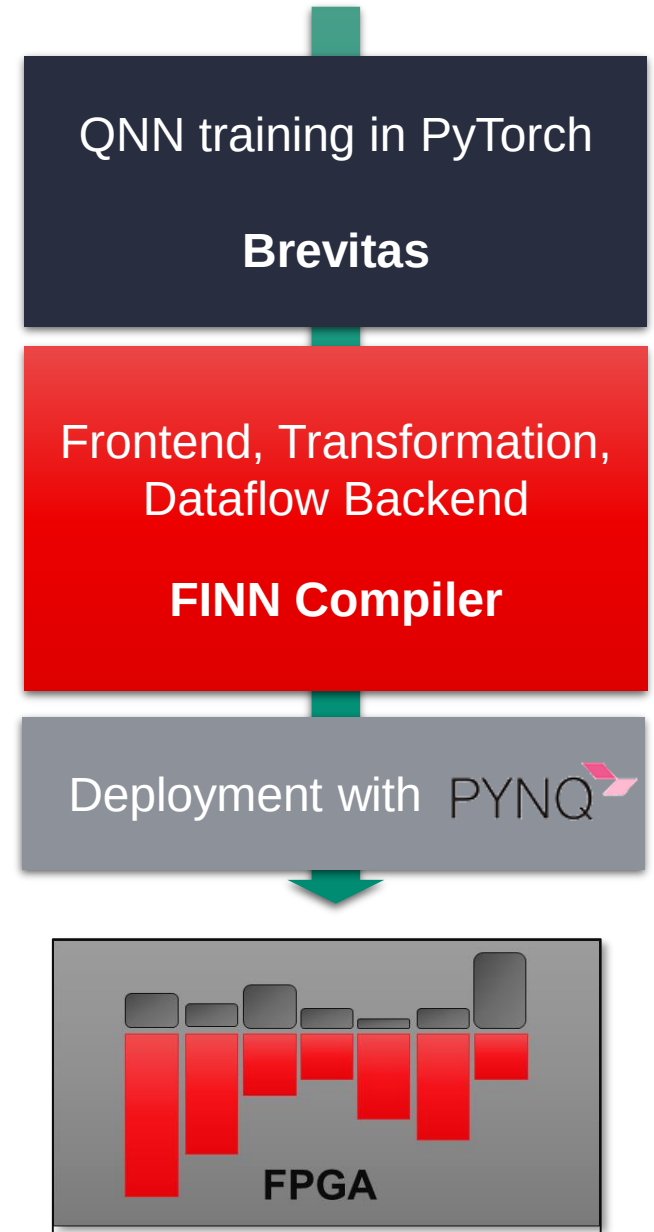
FINN



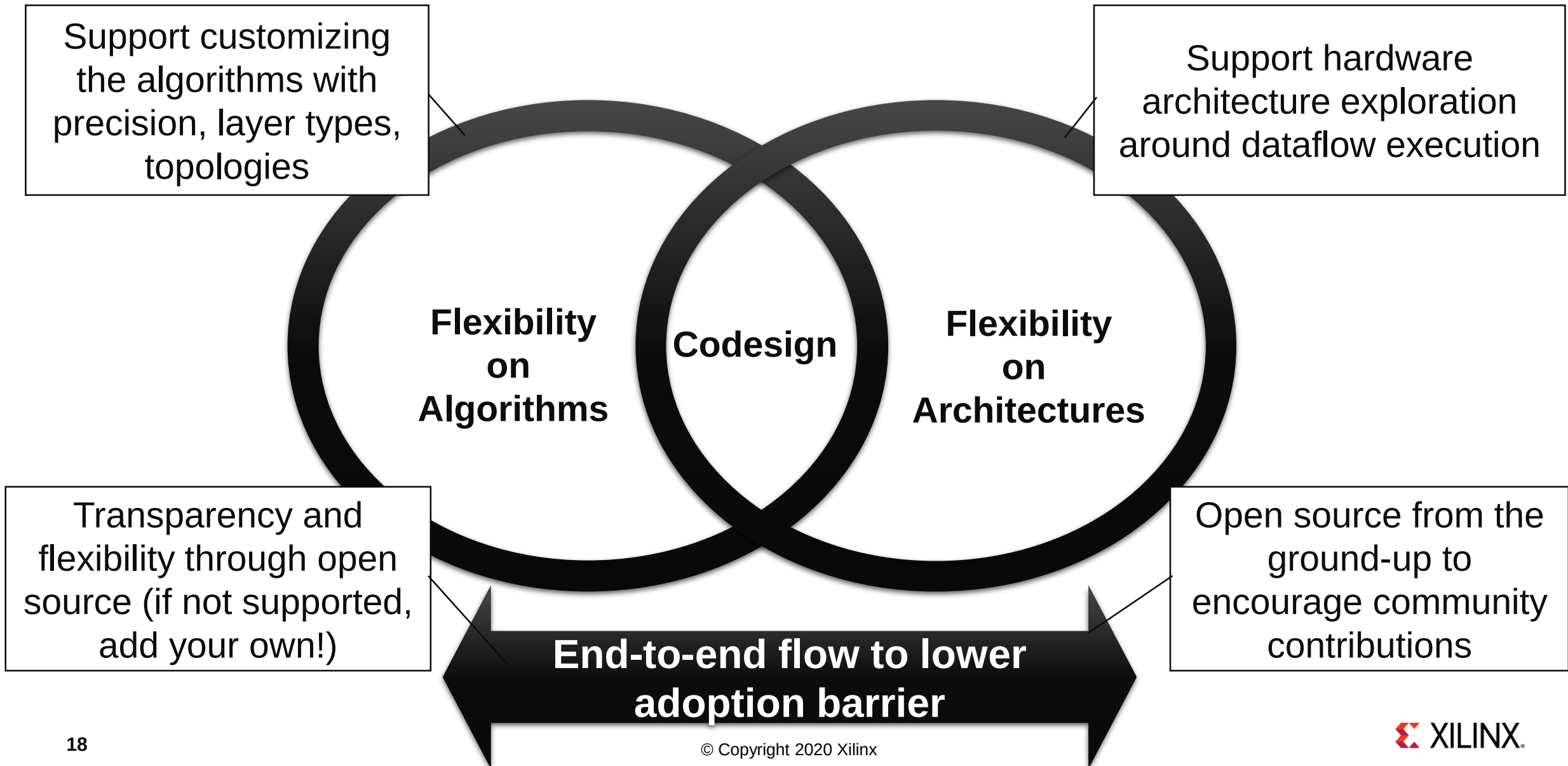
The FINN Project

- ▶ A framework for DNN co-design on Xilinx FPGAs
 - Focus on QNNs and per-network tailored HW architecture
- ▶ End-to end design flow
 - Go all the way from training framework to hardware
 - Demos on PYNQ platforms
- ▶ Revamped from ground-up for open source
 - To enable innovation throughout a growing community
 - Transparent and customizable

<https://xilinx.github.io/finn>

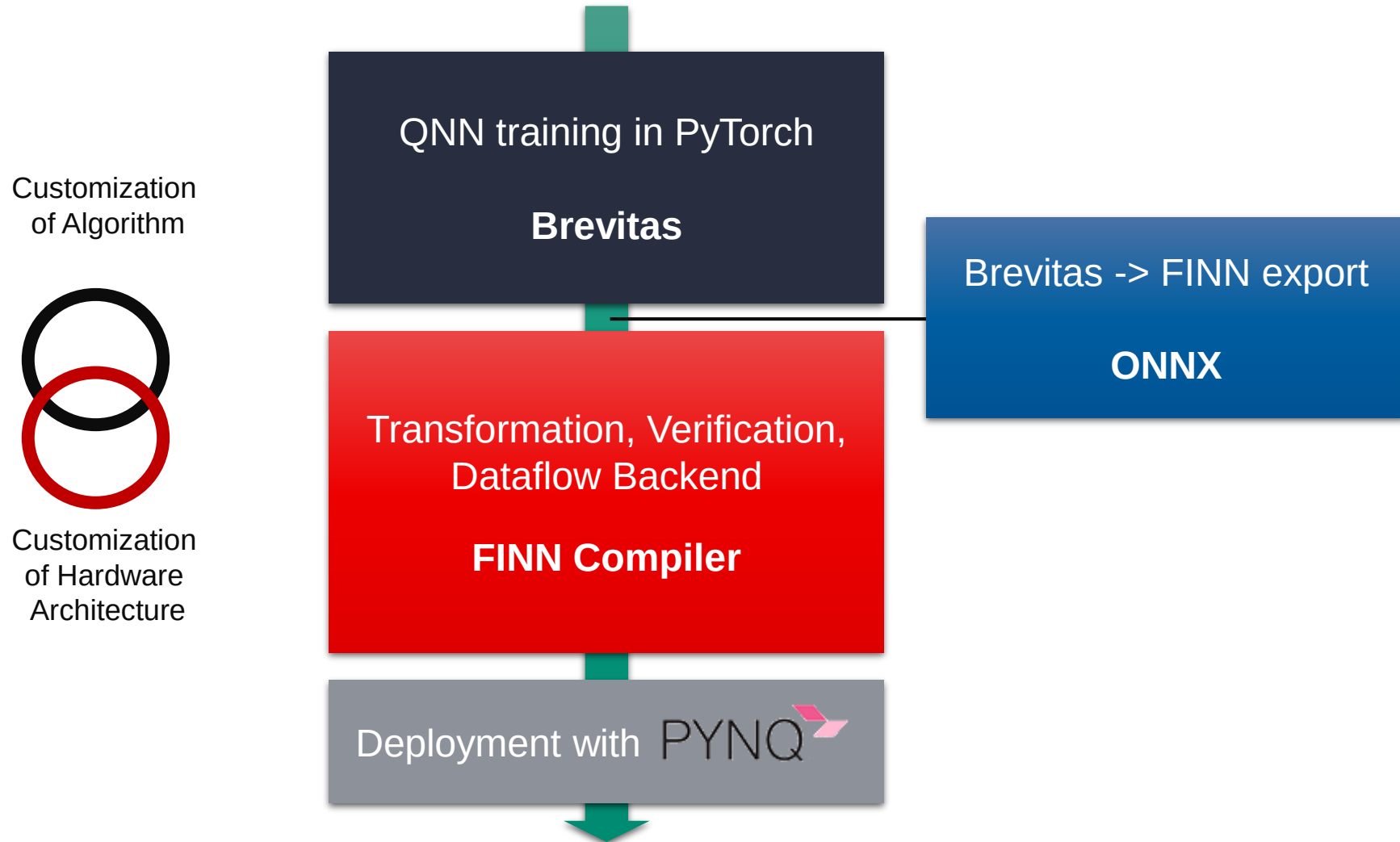


The FINN Project: Mission

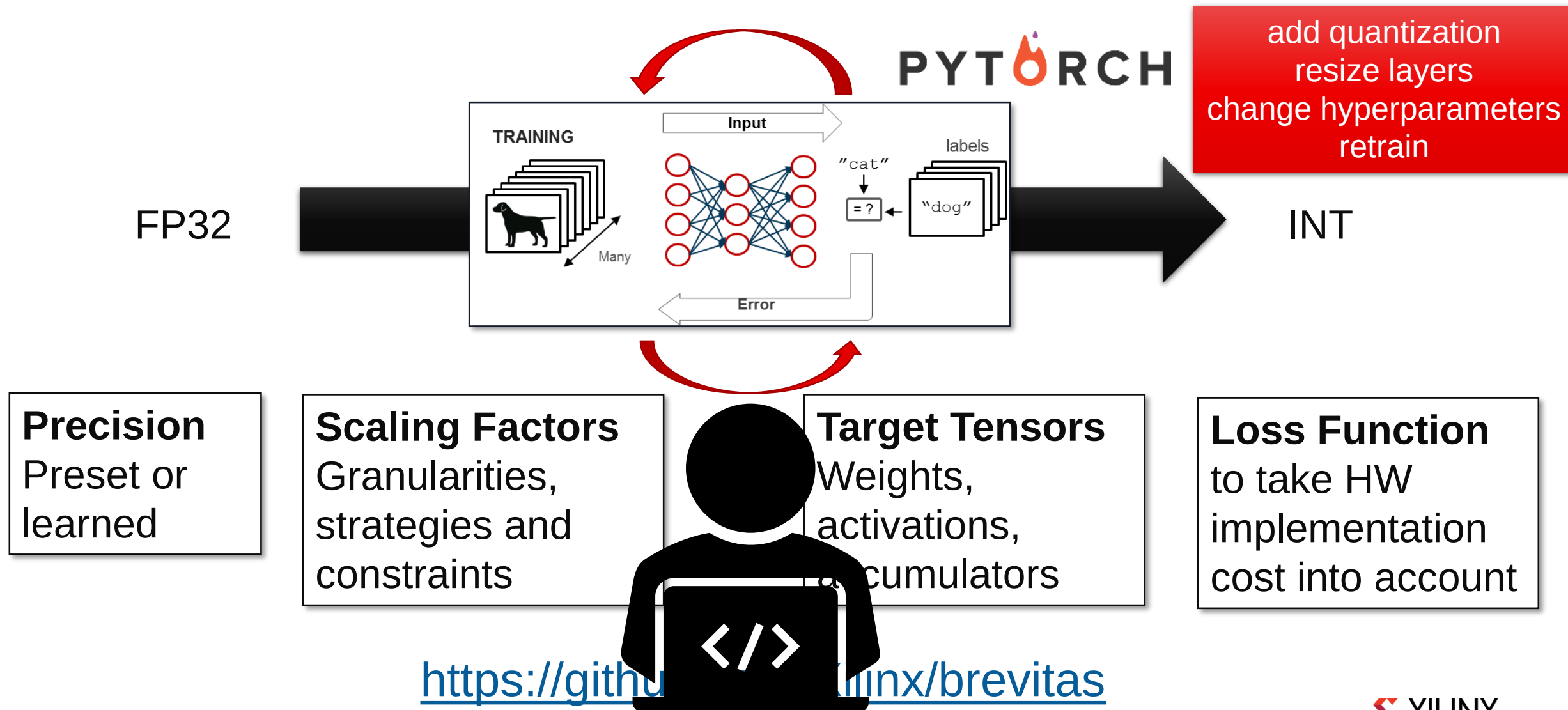


The FINN Project: Components of the Stack

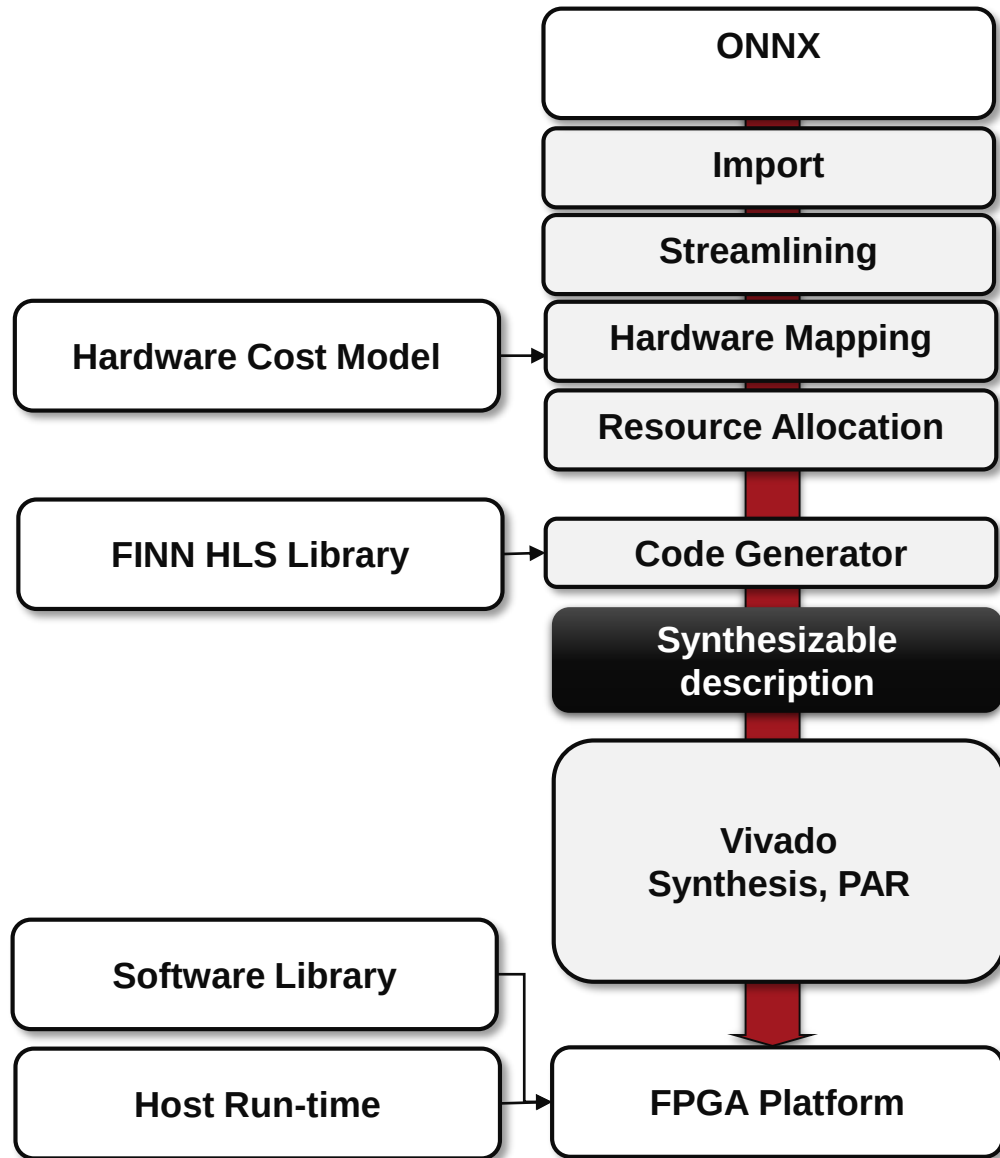
From Framework to FPGA



Brevitas: A PyTorch library for quantization-aware training



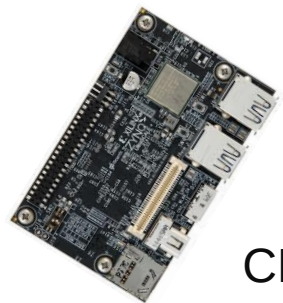
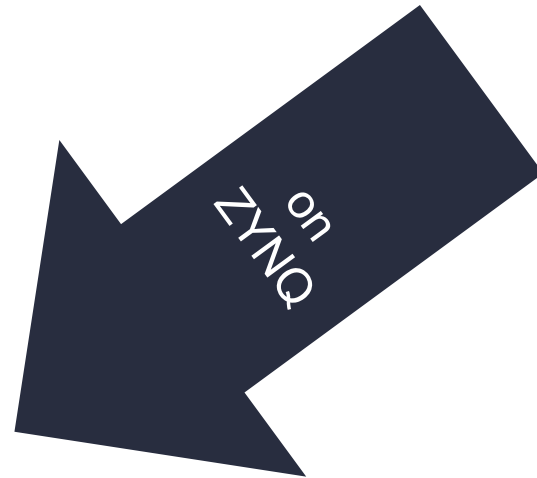
An Overview of the FINN Compiler



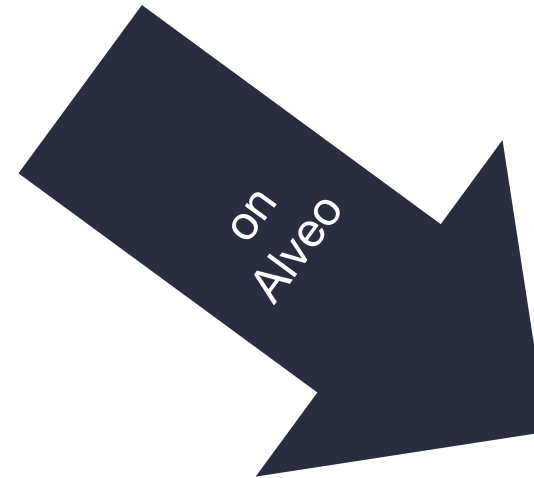
```
void DoCompute(ap_uint<64> * in, ap_uint<64> * out) {  
    #pragma HLS DATAFLOW  
    stream<ap_uint<64> > memInStrm("memInStrm");  
    stream<ap_uint<64> > InStrm("InStrm");  
    .  
    .  
    stream<ap_uint<64> > memOutStrm("memOutStrm");  
  
    Mem2Stream<64, inBytesPadded>(in, memInStrm);  
    StreamingMatrixVector<L0_SIMD, L0_PE, 16, L0_MW, L0_MH, L0_WMEM, L0_TMEM>  
        (InStrm, inter0, weightMem0, thresMem0);  
    StreamingMatrixVector<L1_SIMD, L1_PE, 16, L1_MW, L1_MH, L1_WMEM, L1_TMEM>  
        (inter0, inter1, weightMem1, thresMem1);  
    StreamingMatrixVector<L2_SIMD, L2_PE, 16, L2_MW, L2_MH, L2_WMEM, L2_TMEM>  
        (inter1, inter2, weightMem2, thresMem2);  
    StreamingMatrixVector<L3_SIMD, L3_PE, 16, L3_MW, L3_MH, L3_WMEM, L3_TMEM>  
        (inter2, outstream, weightMem3, thresMem3);  
    StreamingCast<ap_uint<16>, ap_uint<64> >(outstream, memOutStrm);  
    Stream2Mem<64, outBytesPadded>(memOutStrm, out);  
}
```

Planned FINN Demos

PYNQ™

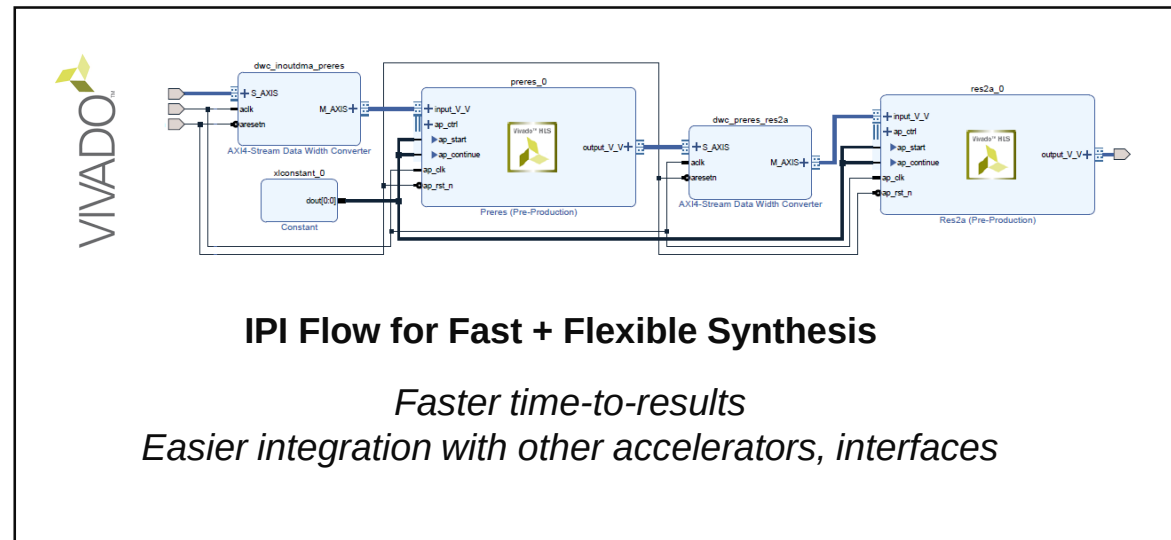
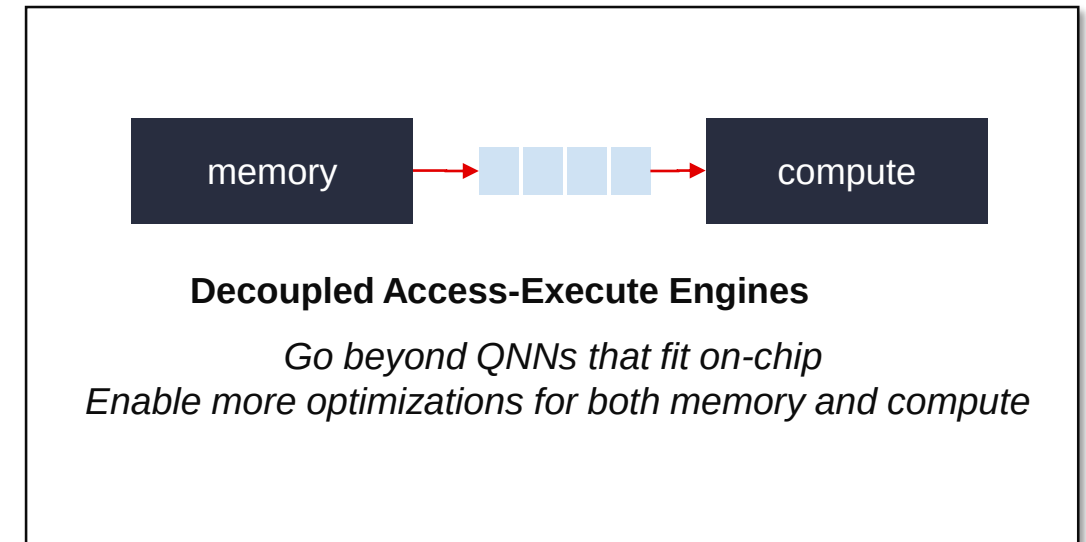
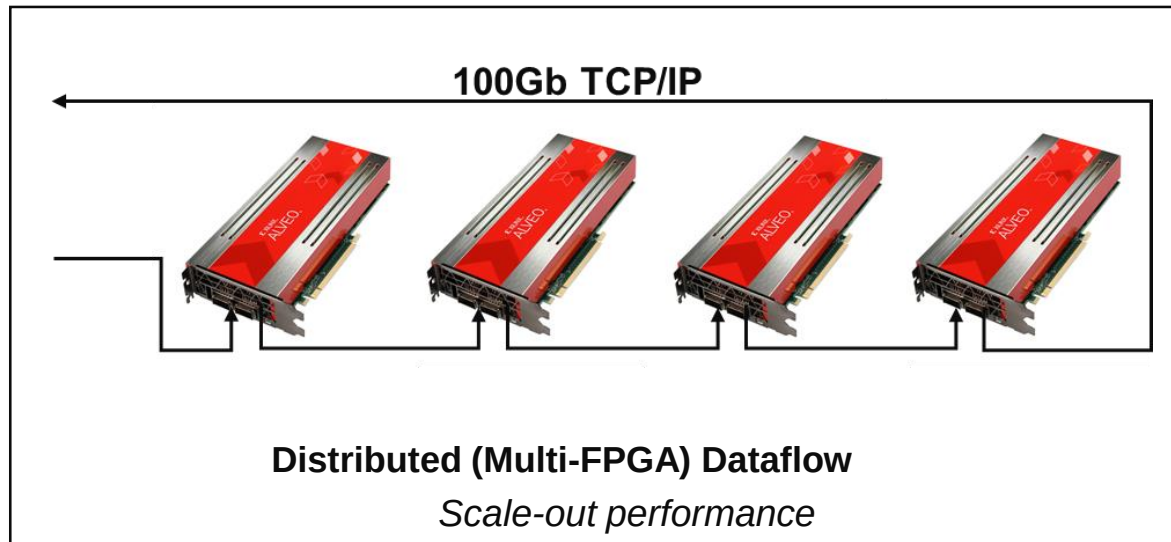


CNV, LFC variants
on MNIST and CIFAR-10
10Ks to Ms of frames per second



MobileNet-v1, ResNet-50
on ImageNet
1Ks of frames per second

Planned Features for the FINN Compiler



The FINN Project: Open Source



GitHub

<https://xilinx.github.io/finn>

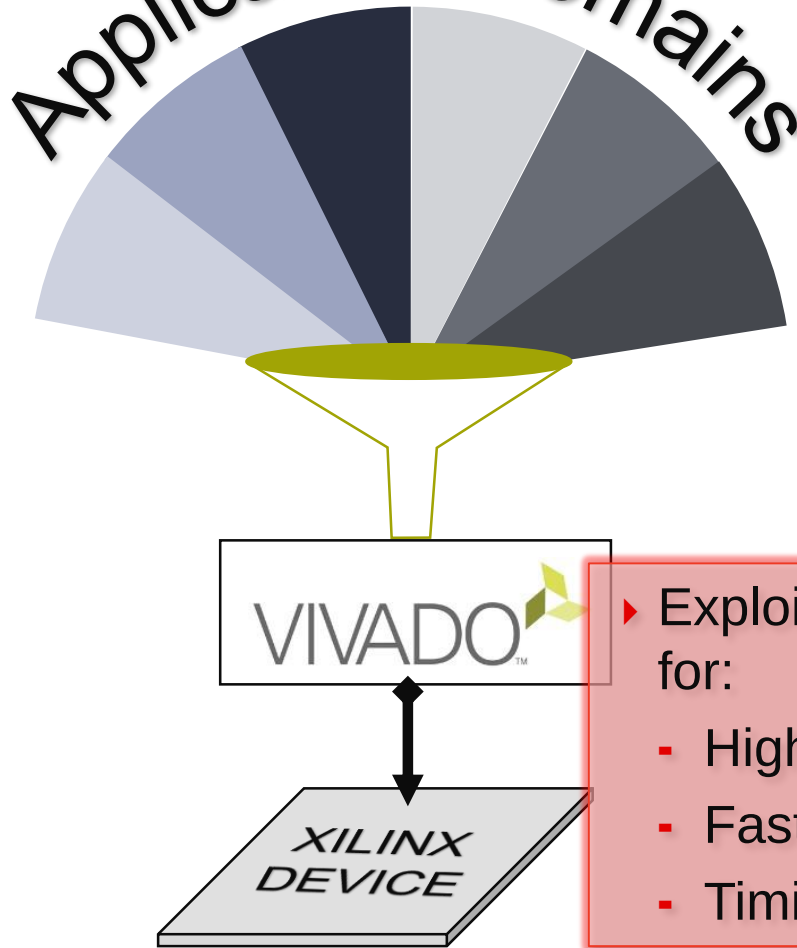
- ▶ Enable the community to train, customize and deploy their own QNNs
- ▶ Some components and demonstrators already open-source
- ▶ New clean-slate version in progress, open development on GitHub
 - **Join our growing community!**

Enabling Domain-specific Backend Productivity for FPGAs



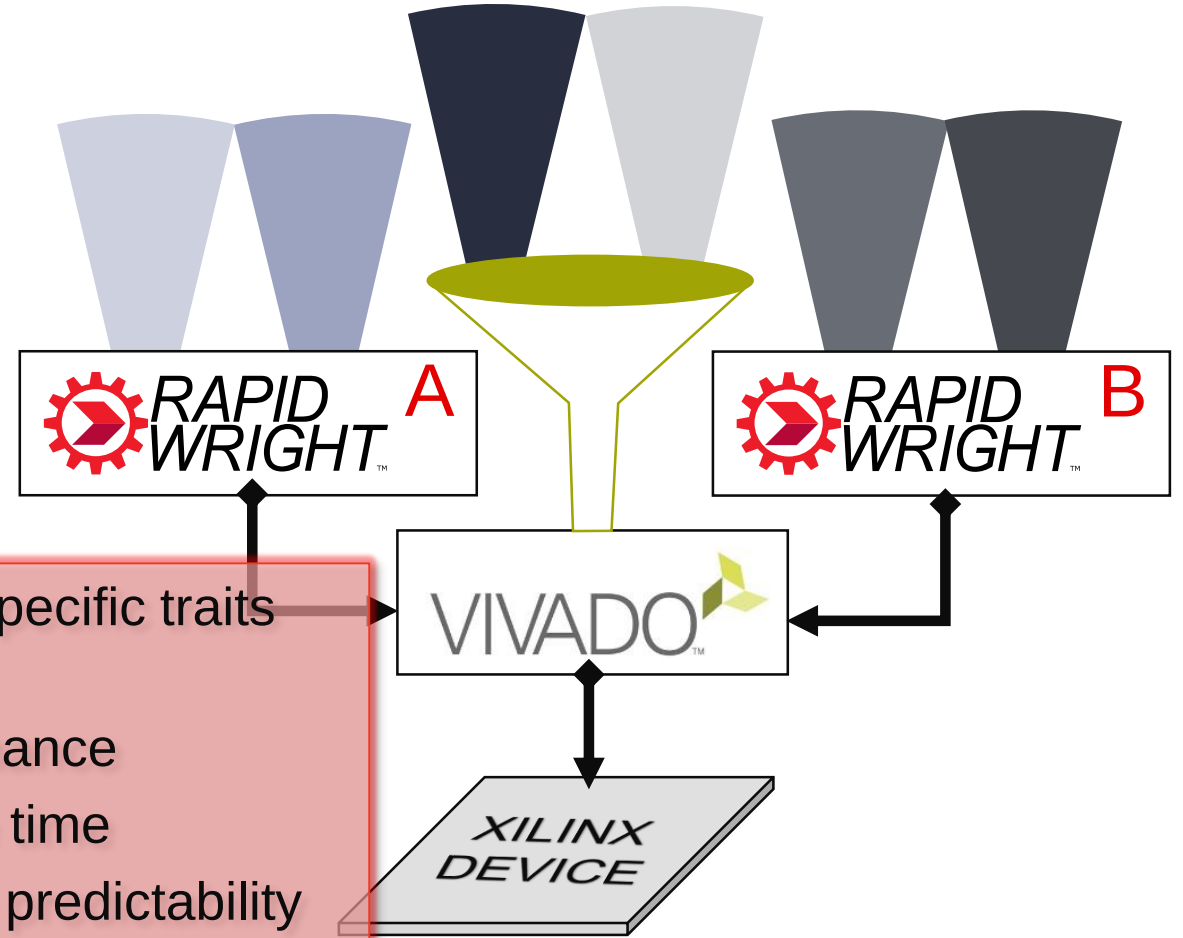
Domain Specific Era Needs Domain Specific Backends

Application Domains



- ▶ Exploit domain-specific traits for:
 - Higher performance
 - Faster compile time
 - Timing closure predictability

Application Domains

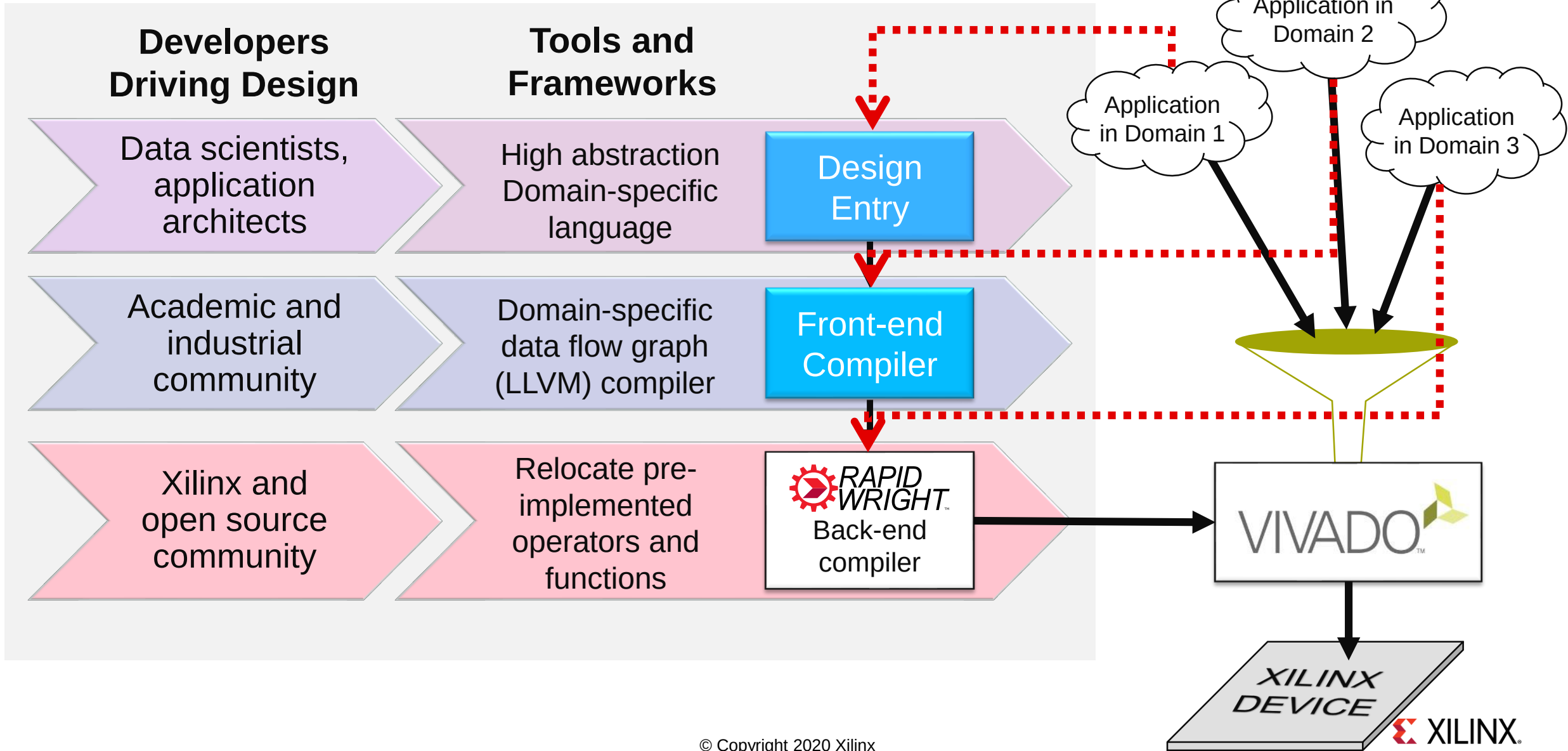


Vivado must **generalize** solutions

RapidWright can **specialize**

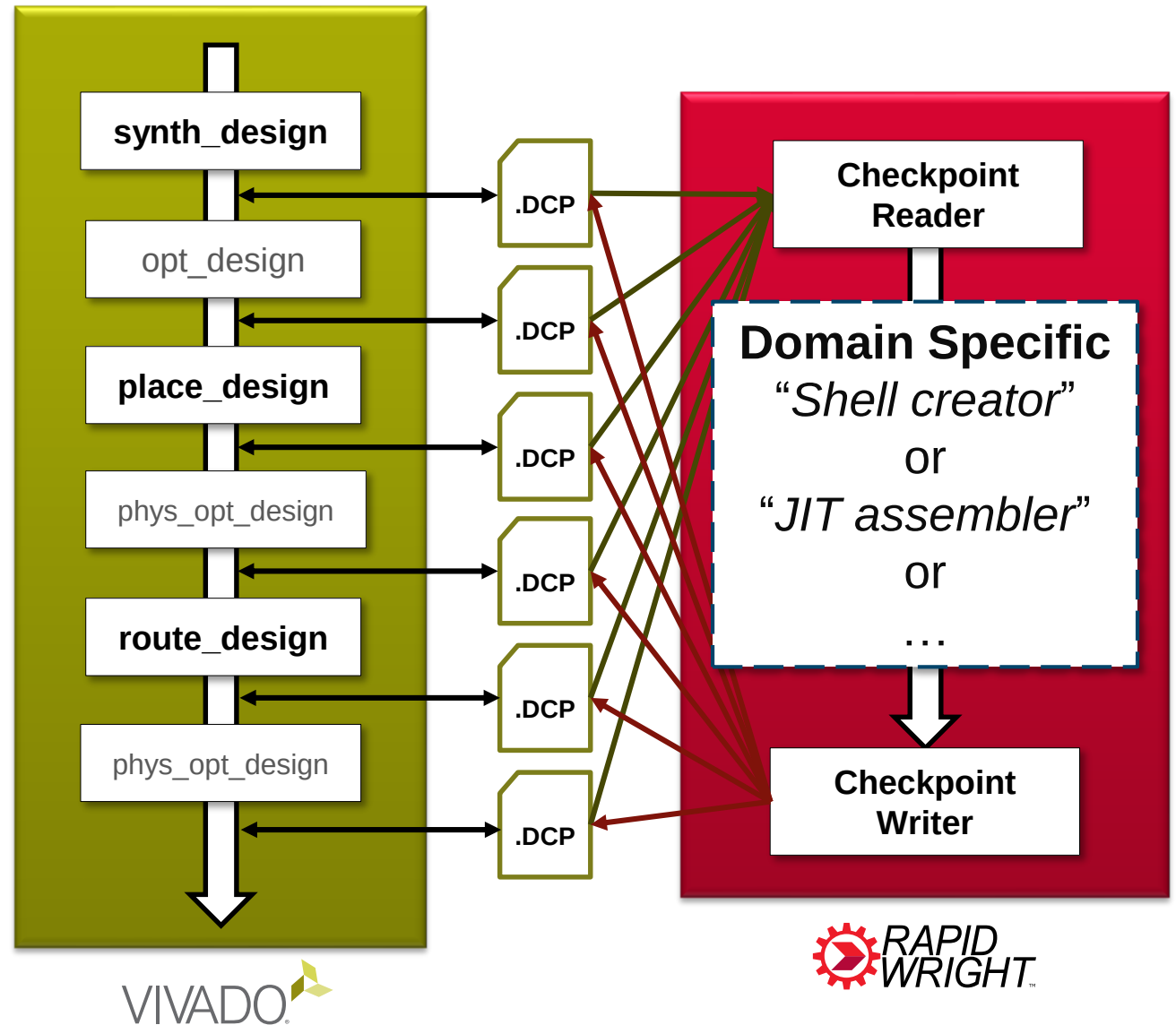


Proposed Domain-specific Tool Flows



RapidWright Overview

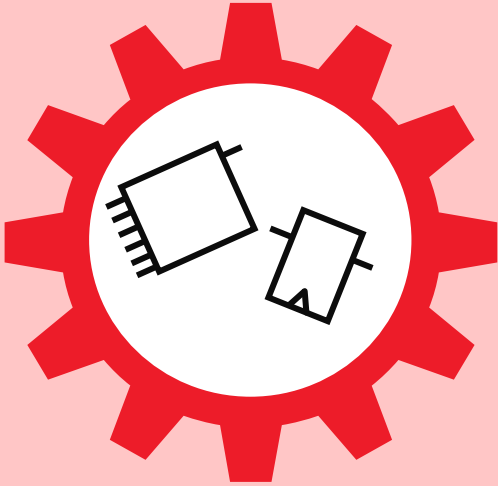
- ▶ Companion framework for Vivado
 - Fast, light-weight, open source
 - Communicates through Design CheckPoints¹
 - Java code, Python scripting
- ▶ Enables targeted solutions
 - Reuse & relocate pre-implemented modules
 - Just-in-time implementations
 - Create shells & overlays
- ▶ Power user ecosystem
 - Academic algorithm validation
 - Rapid prototyping of CAD concepts



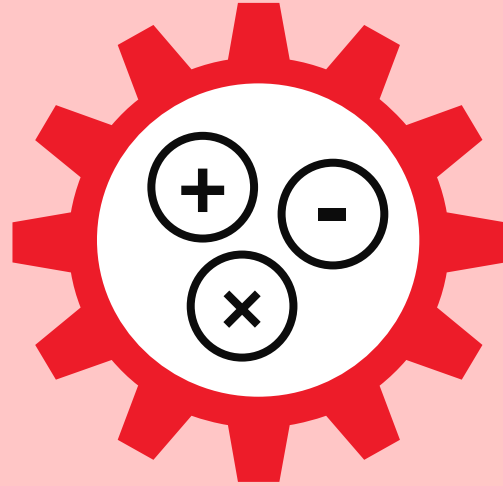
¹DCP = netlist + P&R data + constraints

4 Ways to Design in RapidWright

BUILD ROUTED CIRCUITS



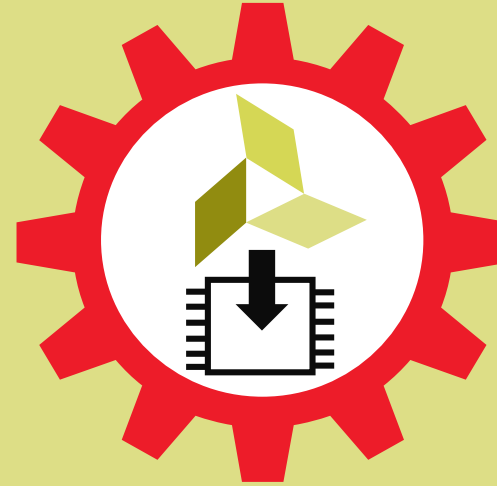
FROM SCRATCH



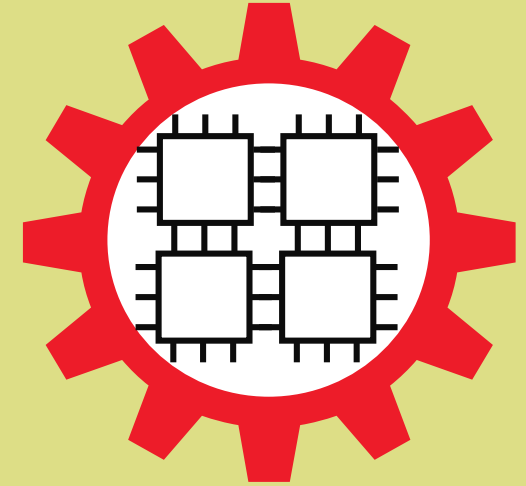
GENERATORS

- ▶ Well-defined circuits in seconds
- ▶ Parameterizable library of generators

REUSE P&R CIRCUITS



FROM VIVADO



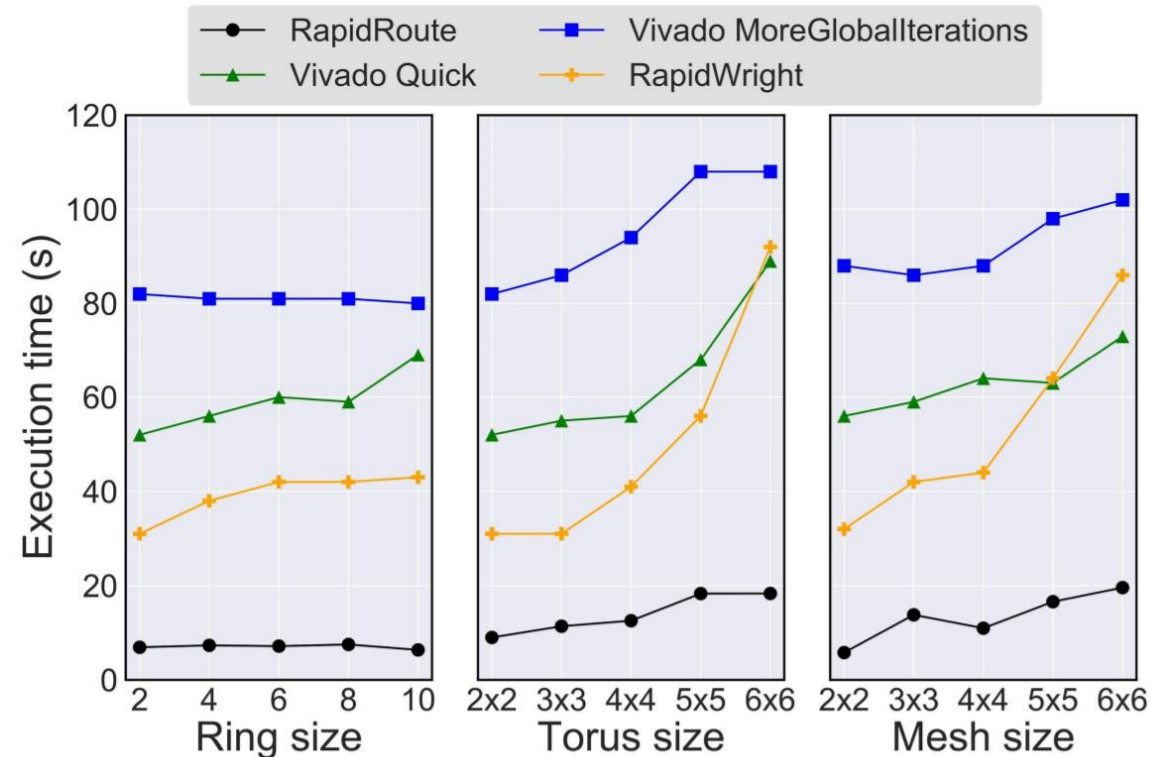
SHELLS & OVERLAYS

- > Reuse/relocate P&R circuits from Vivado
- > Combine P&R circuits together

RapidRoute – Domain-specific Fast Router

- ▶ Timing-aware router for Xilinx FPGAs
 - Competitive timing closure ($\pm 0.05\text{ns}$)
- ▶ 5-8X faster runtime than Vivado
- ▶ Targets replicated structures (NoCs)
- ▶ Open Source:
 - <https://git.uwaterloo.ca/watcag-public/RapidRoute>

Routing Runtime



References

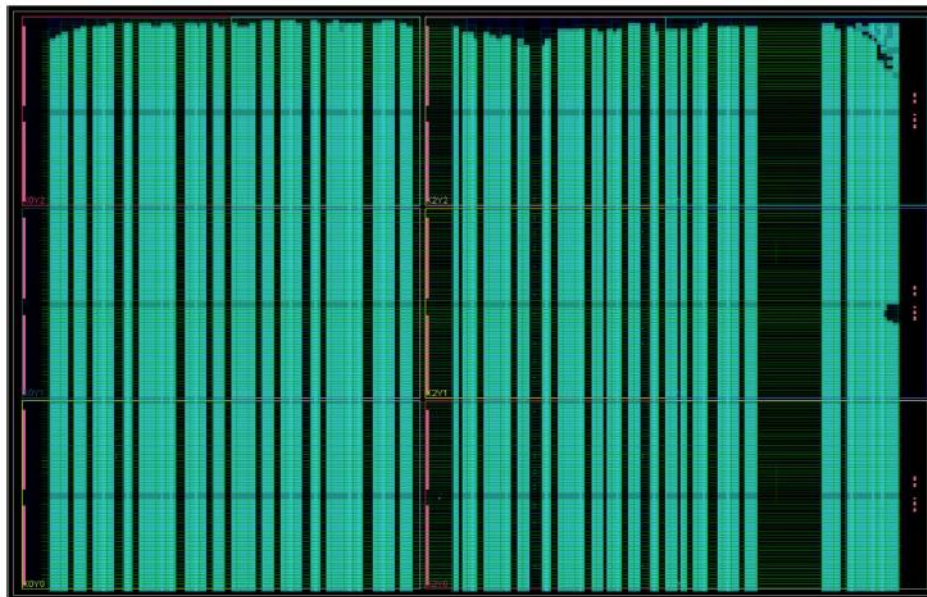
- Leo Liu and Nachiket Kapre, FPL'19, Timing-aware routing in the RapidWright framework
- Leo Liu, Jay Weng and Nachiket Kapre, FCCM'19, RapidRoute: Fast Assembly of Communication Structures for FPGA Overlays



Distributed Memory FPGA Debug

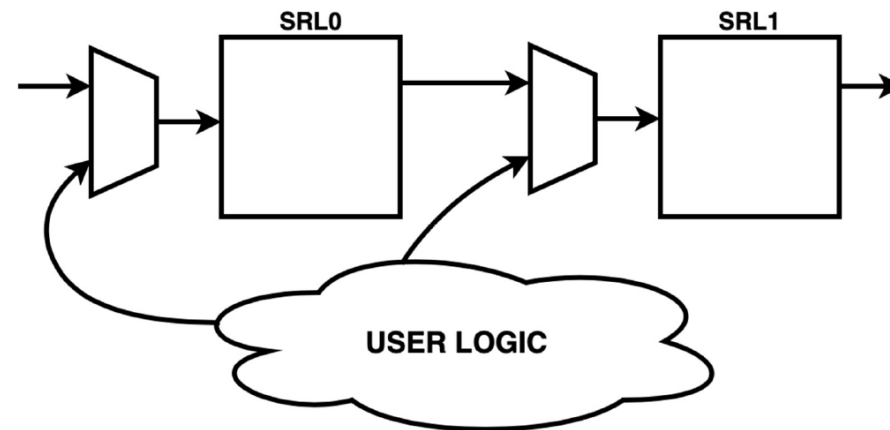
FPGA with
94% of LUT
resources
utilized:

Xilinx ILA:



Where will the embedded logic analyzer fit?

Distributed Memory (DIME) Debug



- ▶ No Block RAM needed
- ▶ 16-256 bit trace depth
- ▶ Exploring post P&R and pre-allocating



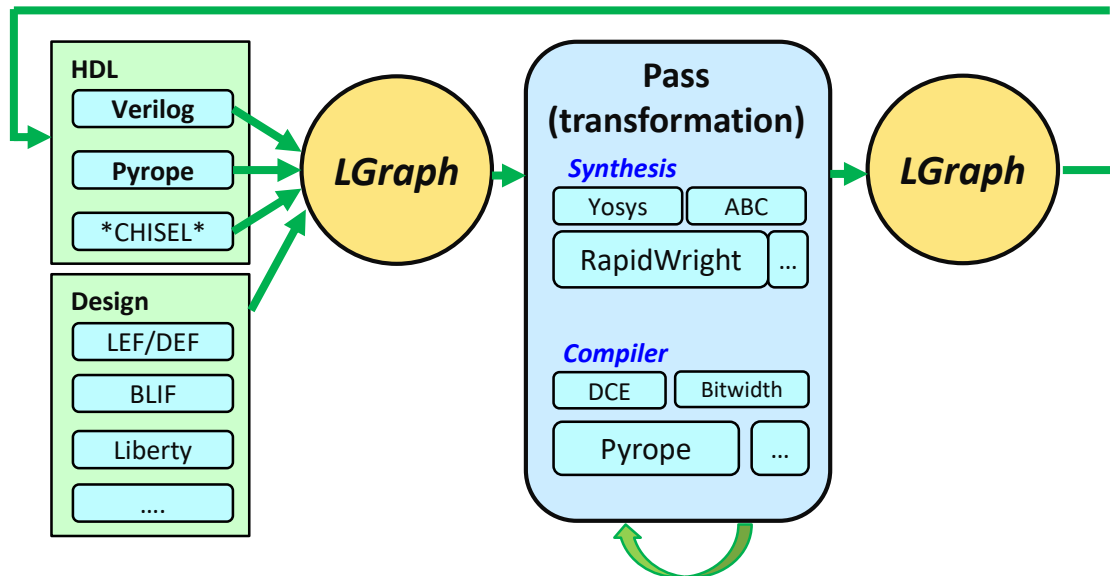
References

- Robert Hale & Brad Hutchings,
 - FPL'19, Pre-allocating Resources for Distributed Memory based FPGA Debug
 - FPT'18, Distributed-memory Based FPGA Debug: Design Timing Impact
 - FPL'18, Enabling Low Impact, Rapid Debug for Highly Utilized FPGA Designs

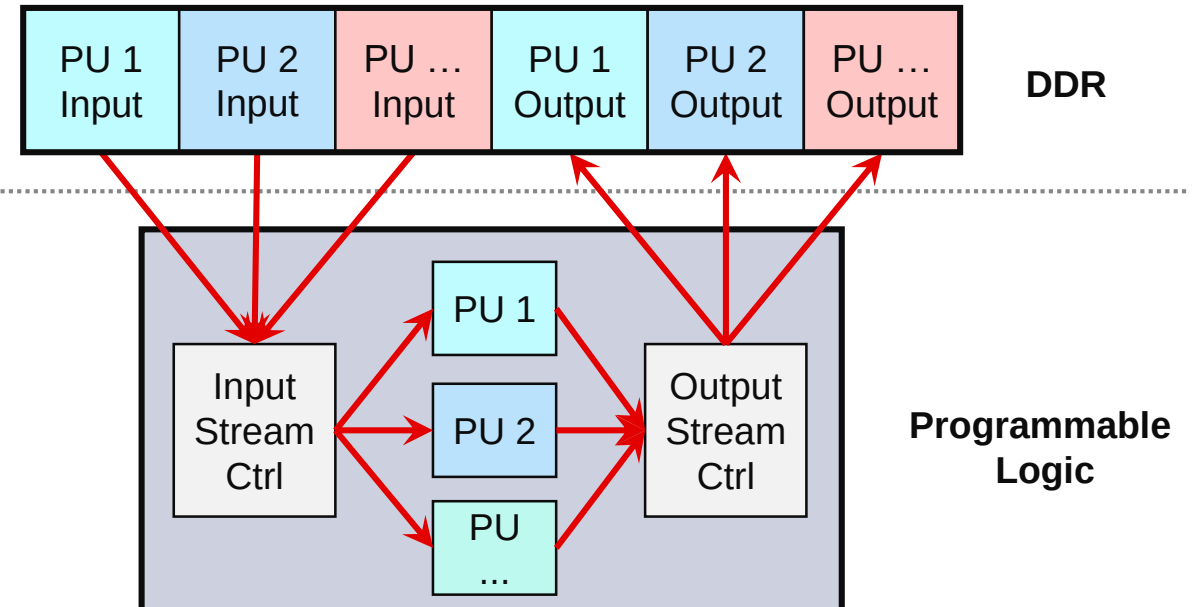
Projects Using RapidWright



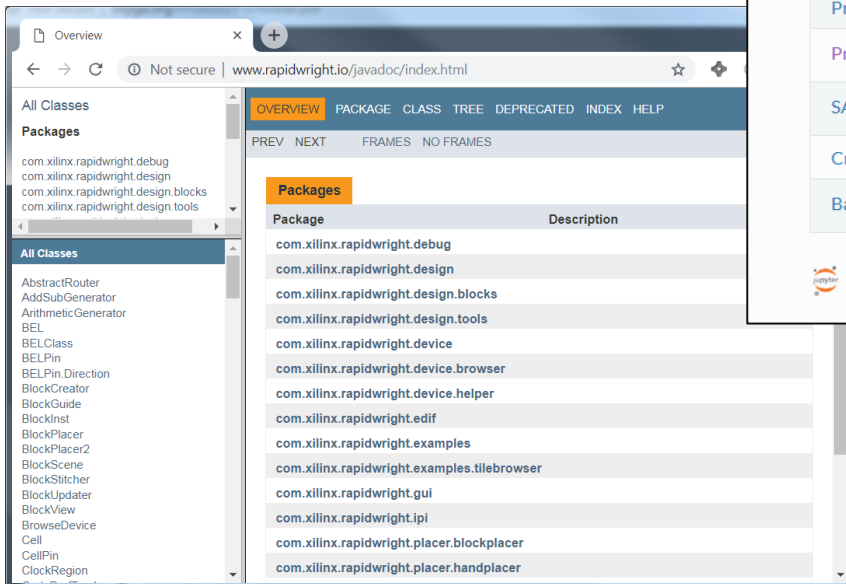
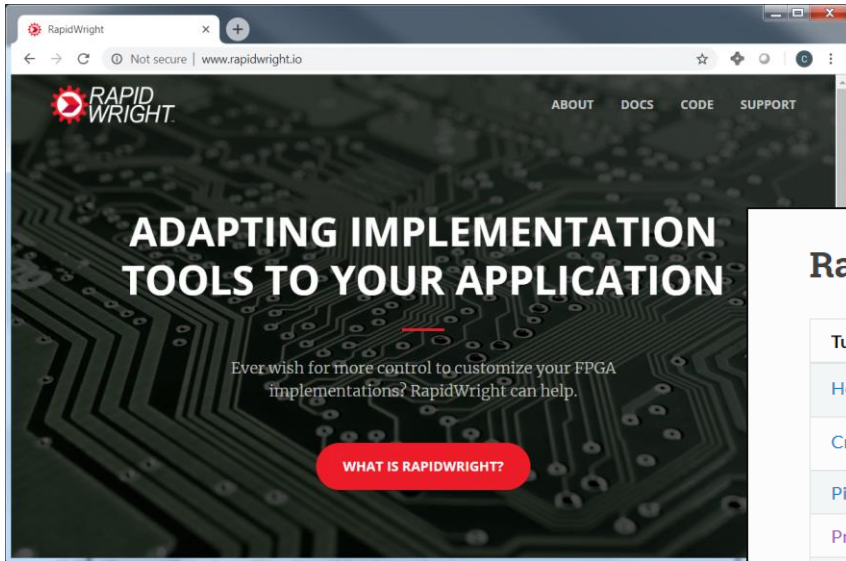
LGraph: “A hardware LLVM compiler”



StreamRTL: Productivity Language for Streaming on FPGAs



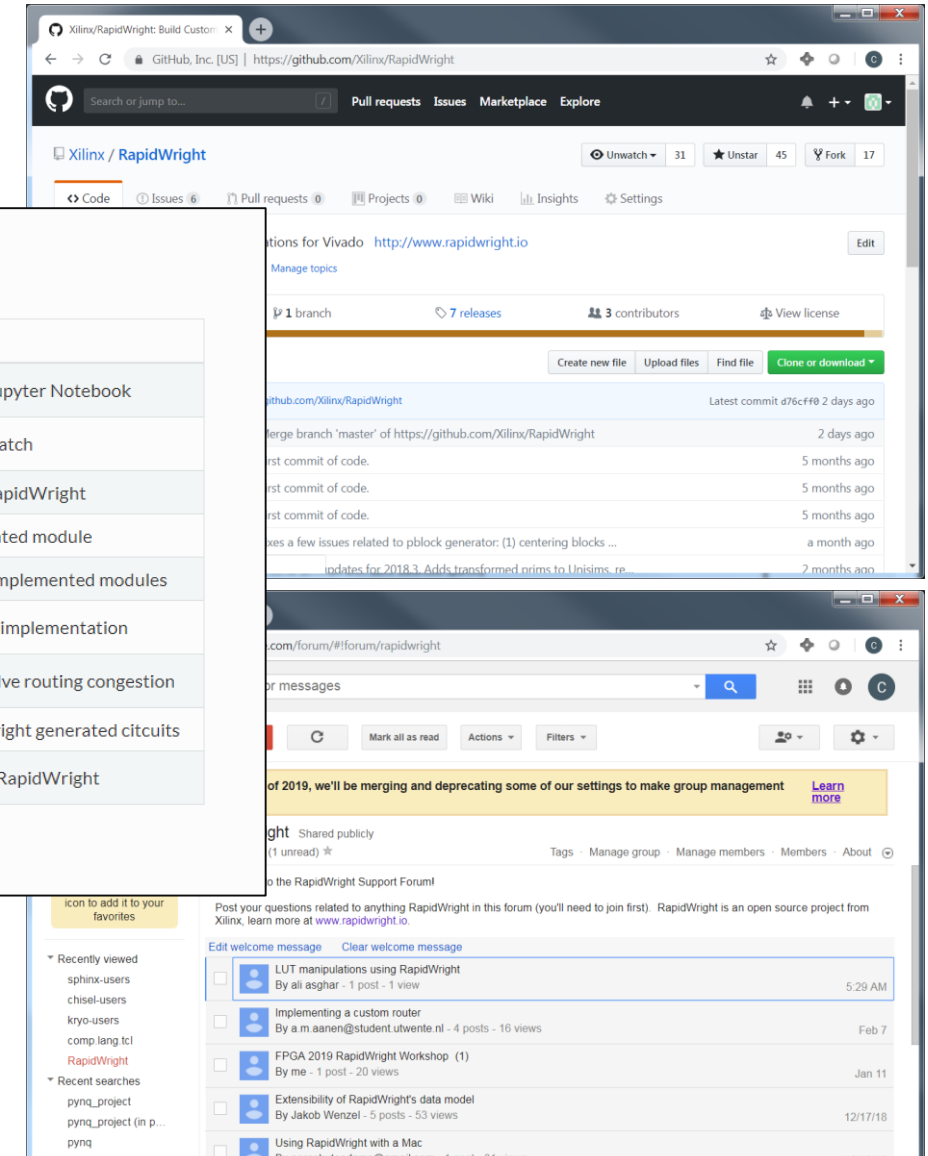
RapidWright Resources: www.rapidwright.io



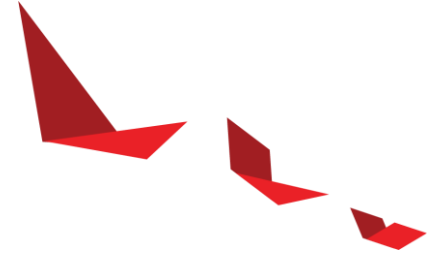
RapidWright FPL 2019 Tutorial

Tutorial Segment	Time	Purpose
Hello, World	5 mins	Intro to RapidWright within Jupyter Notebook
Create Netlist from Scratch	10 mins	How to build a netlist from scratch
Pipeline Generator	15 mins	How to generate a circuit in RapidWright
Pre-implemented Modules: Part I	15 mins	How to create a pre-implemented module
Pre-implemented Modules: Part II	15 mins	How to use and relocate pre-implemented modules
Probe Re-router	20 mins	Fast probe routing on existing implementation
SAT Router (Linux only)	15 mins	How to use a SAT engine to solve routing congestion
Create and Use an SLR Bridge	20 mins	Combine Vivado and RapidWright generated circuits
Basic Routing	20 mins	How to build a basic router in RapidWright

 = Jupyter Notebook Tutorial



Summary



▶ Xilinx CTO open-source research projects

- PYNQ: www.pynq.io
- FINN: xilinx.github.io/finn
- RapidWright: www.rapidwright.io

▶ Open to contributions, collaborations ...

▶ Contact me: cathal.mccabe@xilinx.com



cathalmccabe



www.linkedin.com/in/cathal-mccabe-24a03318



Thank You

