Fast, Scalable Quantized Neural Network Inference on FPGAs with FINN & LogicNets

@ H2RC at Supercomputing, 2020-11-10

Yaman Umuroglu, Senior Research Scientist
Xilinx Research Labs

Xilinx Research, Dublin

- Established over 14 years ago
  - Slowly expanding and increasingly leveraging external funding (IDA, H2020)
  - 6 full-time researchers + interns
- Applications & Architectures
  - Quantifying the value proposition of Xilinx devices in machine learning
- In collaboration with Partners, Customers and Universities

Lucian Petrica, Giulio Gambardella, Alessandro Pappalardo,
Ken O’Brien, Michaela Blott (leader), Nick Fraser, Yaman Umuroglu
(from left to right)

DNNs in Extreme-Throughput Applications

Source: Thomas James, CERN

CERN CMS Experiment

Network Intrusion Detection

- How do we mix DNNs into *extreme-throughput* applications?
  - Need DNNs running at 100Ms of FPS, sub-microsecond latency

XILINX

How Efficient Does Your DNN Need To Be?
A Spectrum of FPGA Inference Alternatives

How Efficient Does Your DNN Need To Be?
A Spectrum of FPGA Inference Alternatives

XILINX VITIS™ | AI

FINN

less efficient
generic
broad scope

DPU, overlays
(10k+ FPS)

FINN
(10M+ FPS)

more efficient
co-designed
specialized

Layer-by-layer compute
(Matrix of Processing Engines)

Optimizing compiler/scheduler
Down to 4-bit

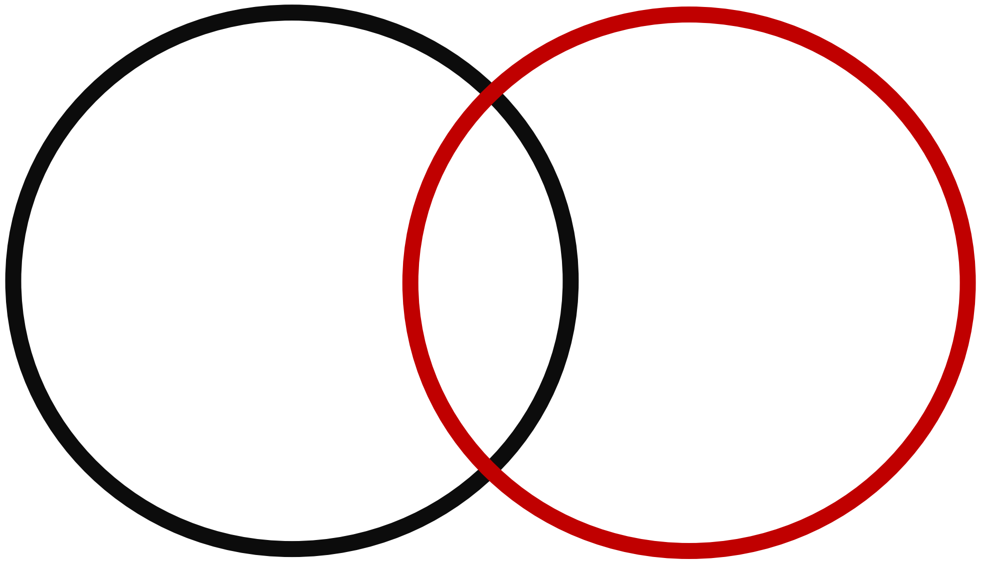
Generated heterogeneous
streaming architecture

Custom topologies,
arithmetic and hardware

XILINX

Customization for Efficient Inference

Customization
of Algorithm

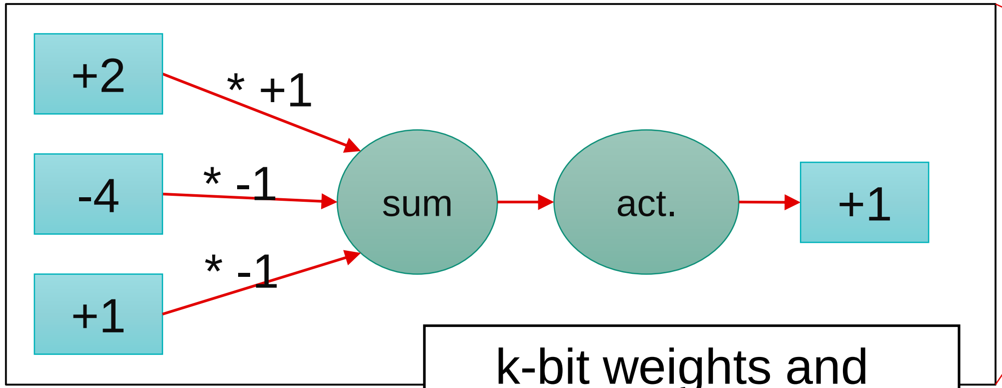


Customization
of Hardware
Architecture

XILINX

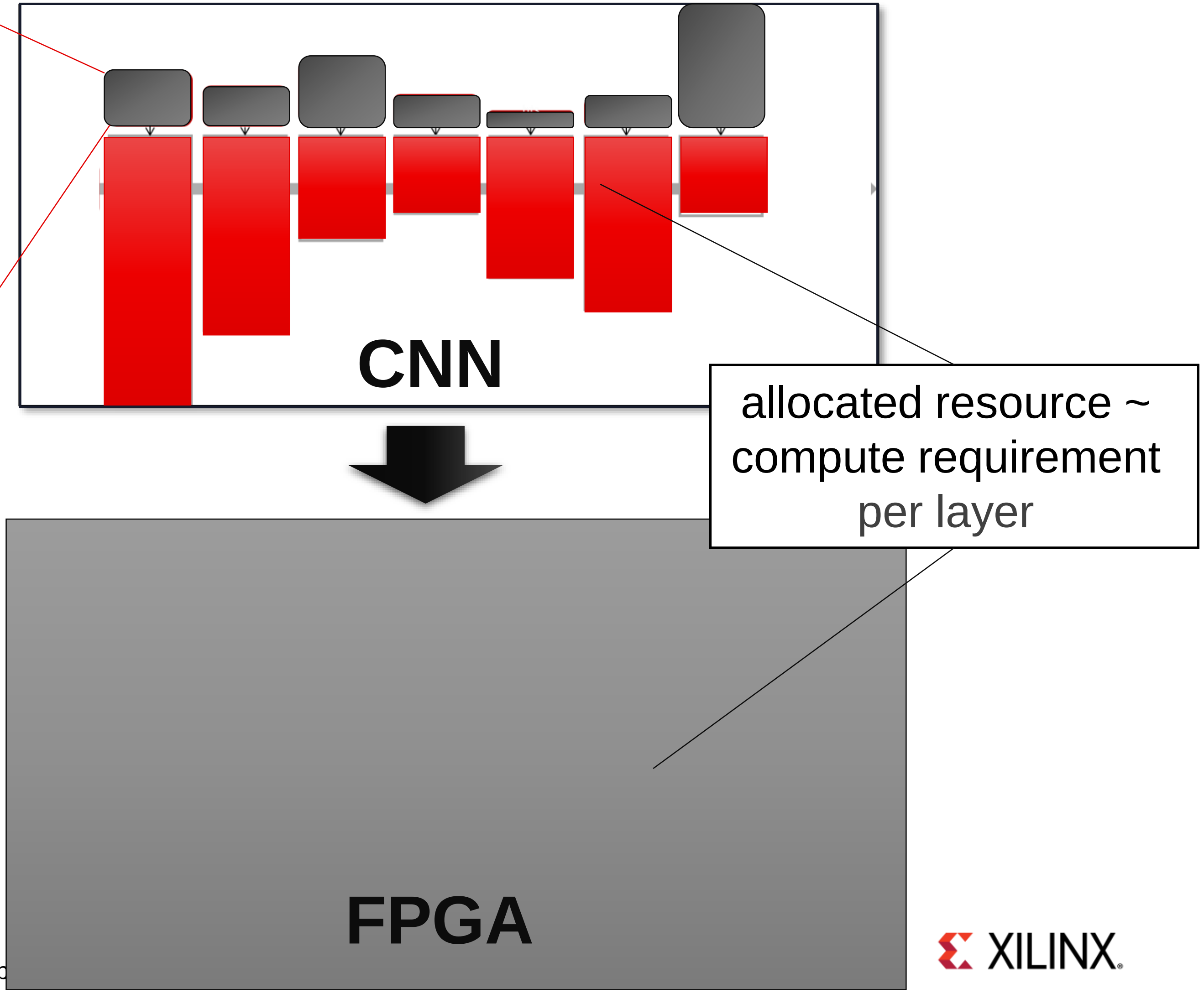
Two Key Techniques for Customization

Few-bit weights & activations (tailored to requirements) + **Streaming dataflow architecture (tailored to requirements)**



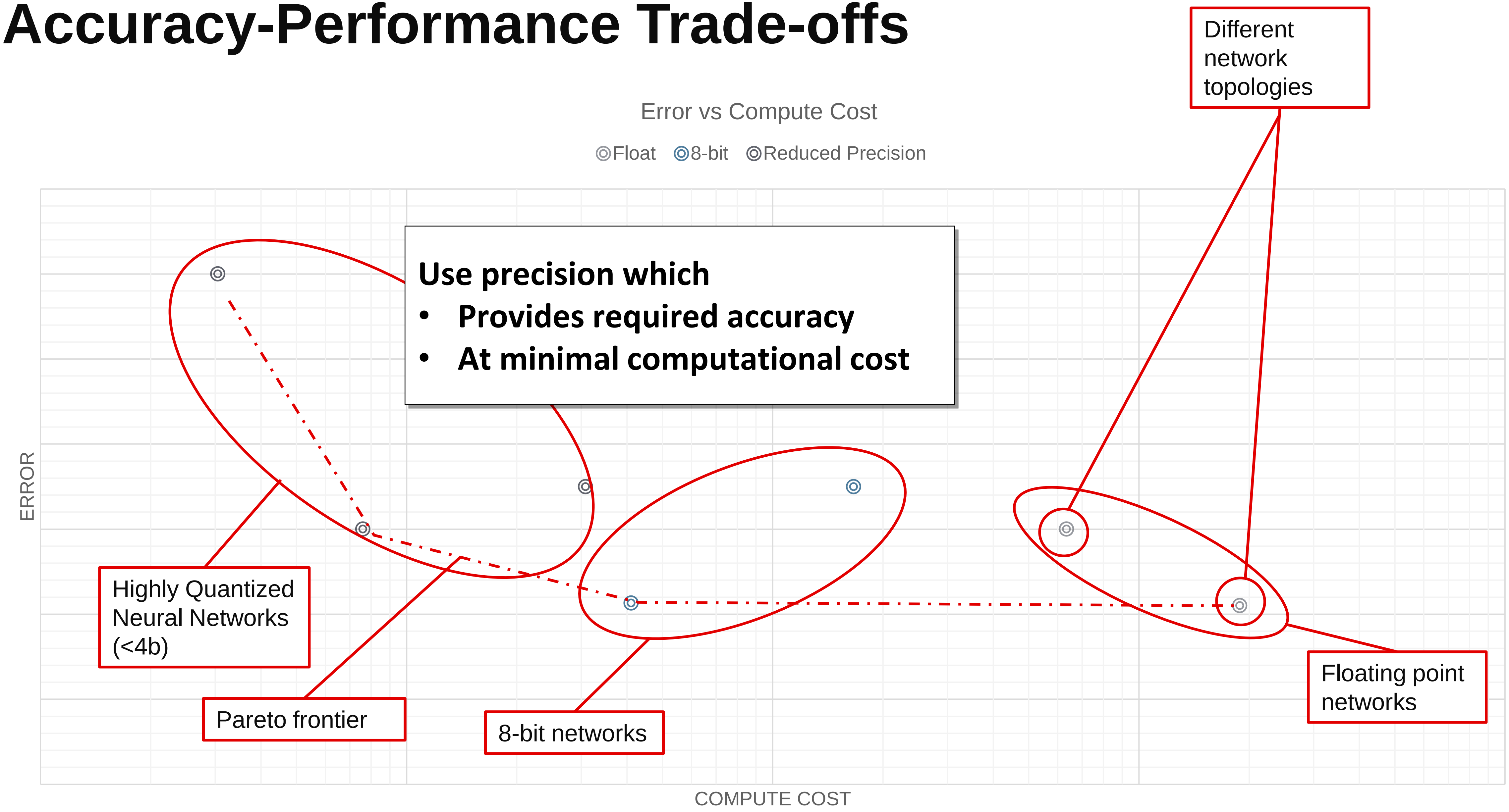
k-bit weights and activations, k <= 4

keep all on-chip!



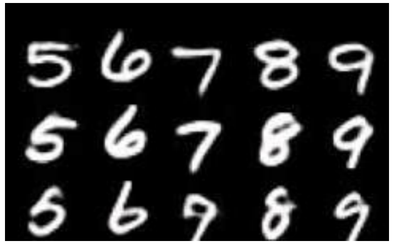
XILINX.

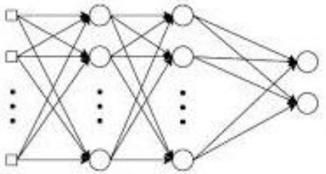
Accuracy-Performance Trade-offs



Few-bit QNNs + FPGA Dataflow: Showcases

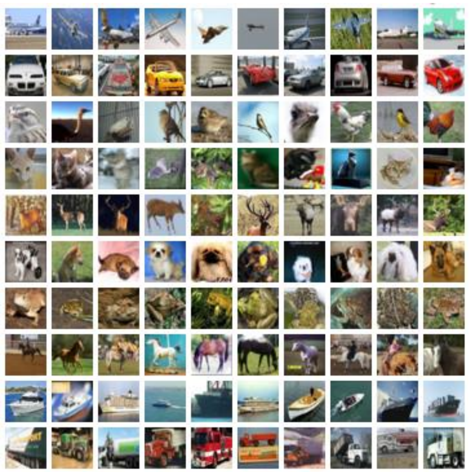
High Throughput & Low Latency





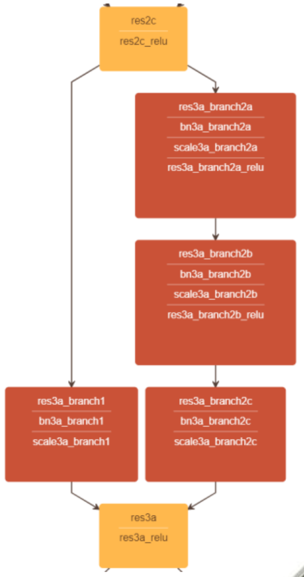
MNIST MLP on ZC706
12.3 M FPS @ 20 W
310 ns latency

Low-Power, Real-Time Image Classification



CIFAR-10 CNV on Pynq-Z1
3000 FPS @ 2.5 W
1 ms latency

Complex Topologies



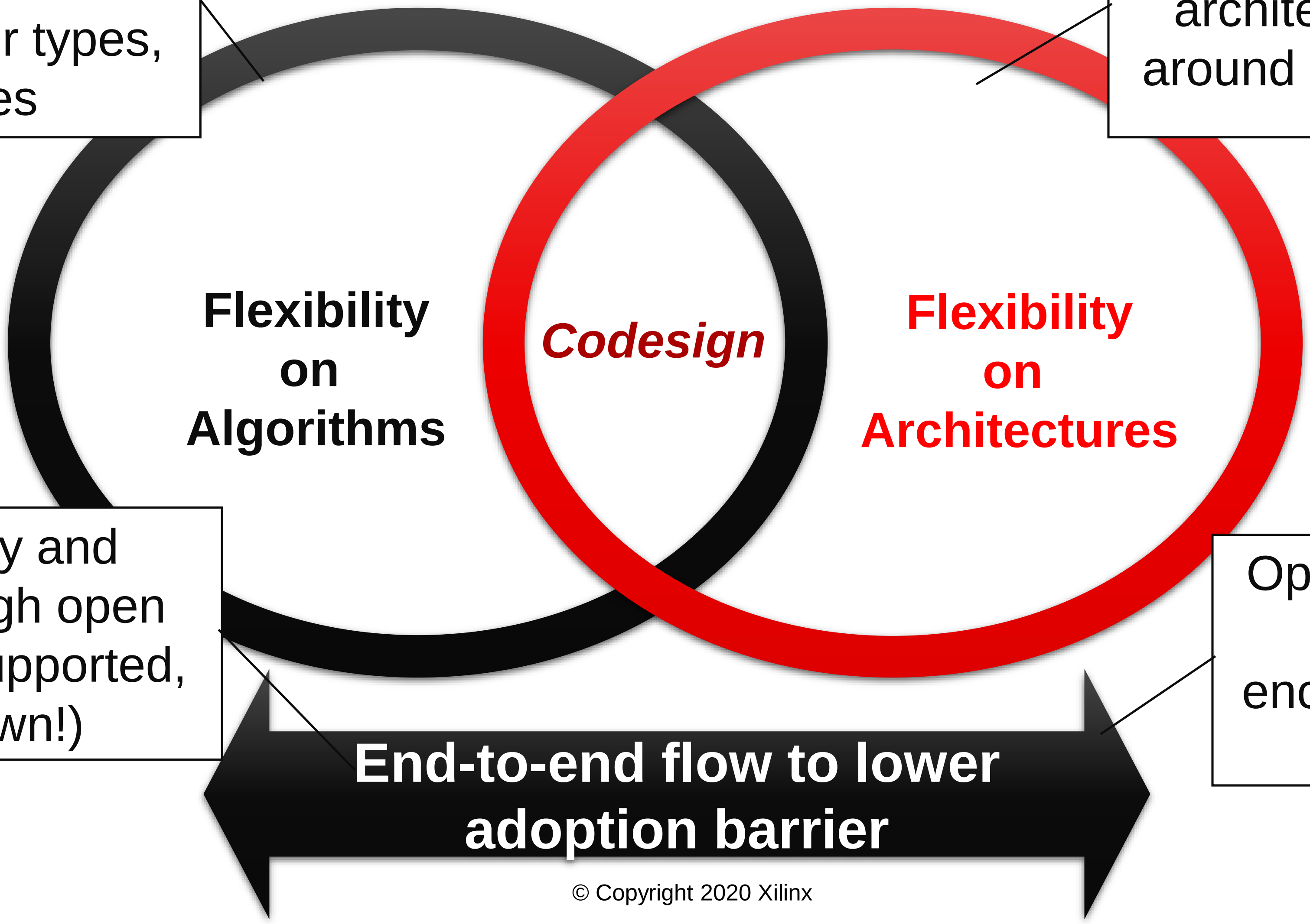
ResNet-50 on Alveo U250
2000 FPS @ 70 W
2 ms latency

XILINX

The FINN Project: Mission

Support customizing the algorithms with precision, layer types, topologies

Support hardware architecture exploration around dataflow execution



Transparency and flexibility through open source (if not supported, add your own!)

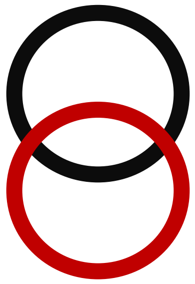
Open source from the ground-up to encourage community contributions

XILINX

The FINN Project: Components of the Stack

From PyTorch to FPGA

Customization of Algorithm



Customization of Hardware Architecture

QNN training in PyTorch

Brevitas

Frontends, Transformation, Dataflow Backend

FINN Compiler

Deployment with PYNQ

for Alveo or Zynq

Support

- Gitter Channel
- Jupyter Notebooks
- readthedocs
- End2End examples

FINN

Quantization-Aware Training in PyTorch with Brevitas

QNN training in PyTorch

Brevitas

Frontends, Transformation, Dataflow Backend

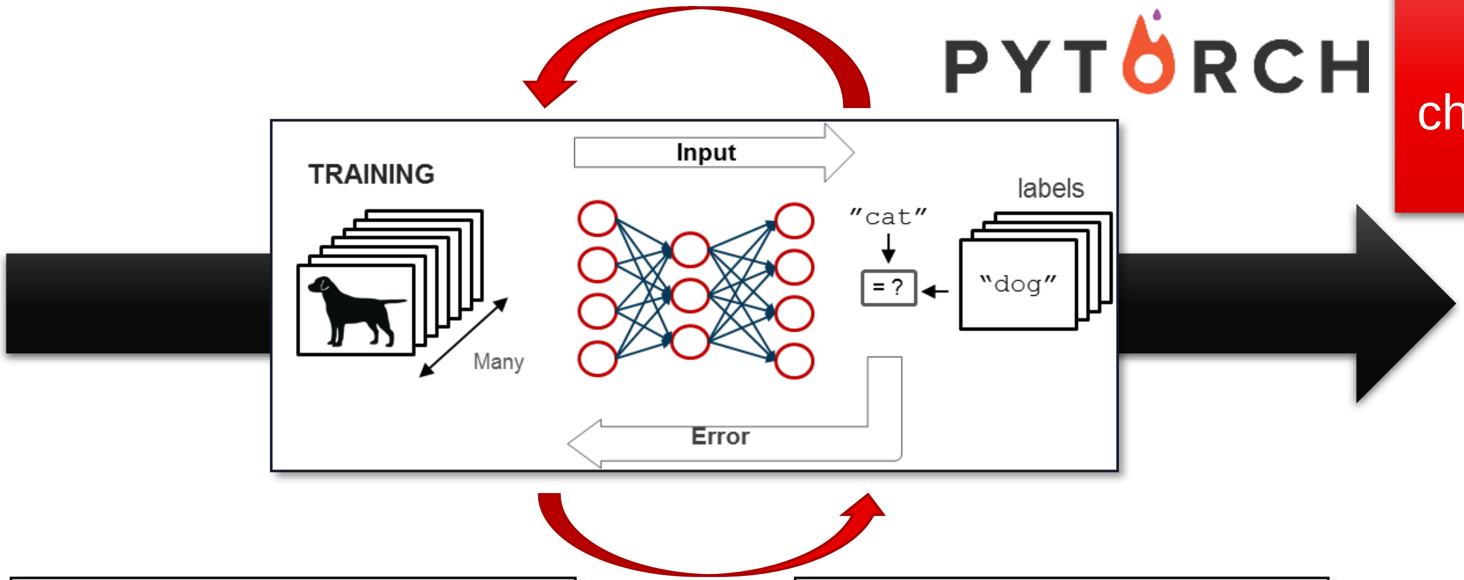
FINN Compiler

Deployment with PYNQ

Brevitas:
A PyTorch library for Quantization-Aware Training

add quantization
resize layers
change hyperparameters
retrain

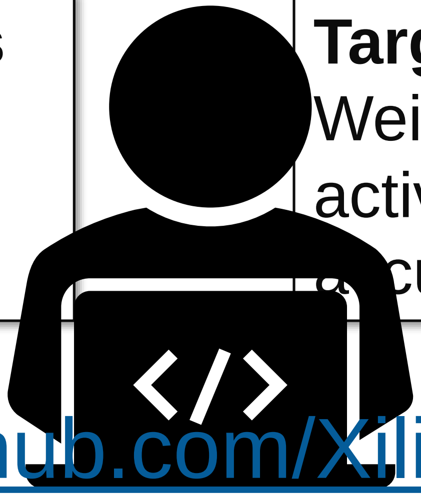
FP32



INT

Precision
Preset or learned

Scaling Factors
Granularities, strategies and constraints



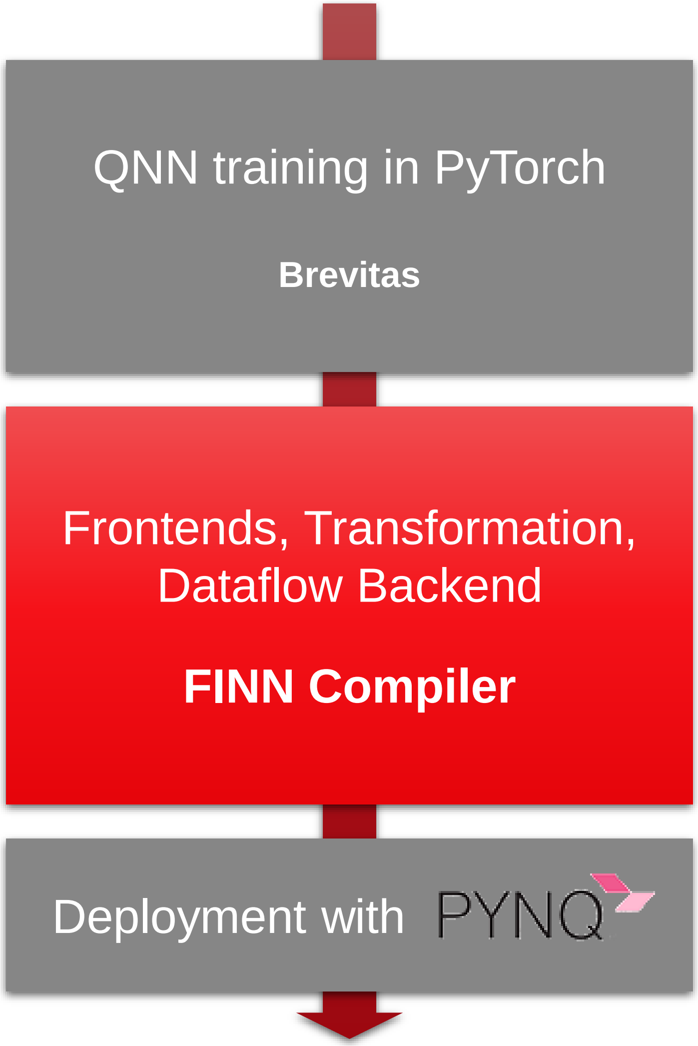
Target Tensors
Weights, activations, accumulators

Loss Function
to take HW implementation cost into account

GitHub https://github.com/Xilinx/brevitas

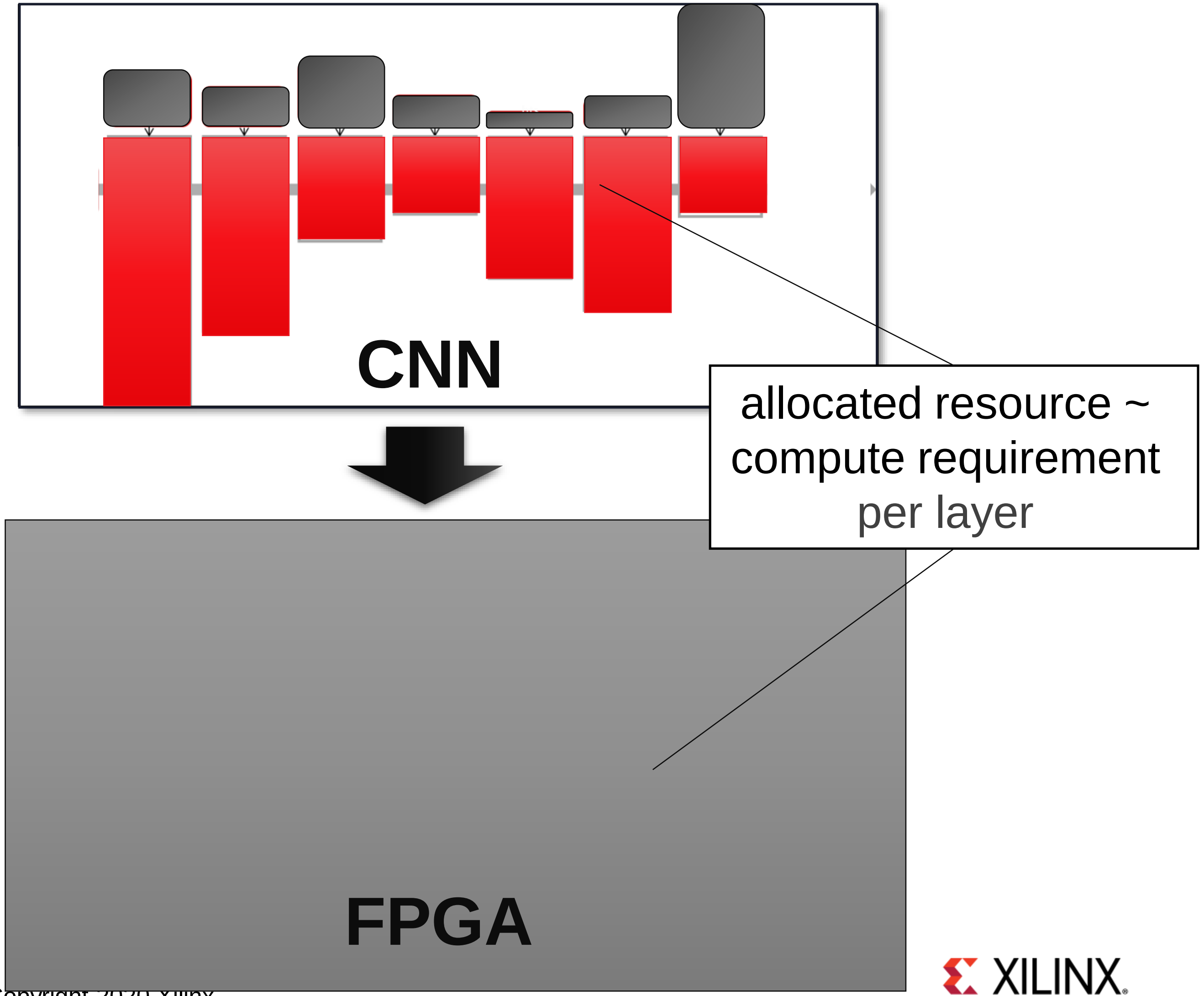
XILINX.

The FINN Compiler



Goal of the FINN compiler:
Transform QNN into custom dataflow architecture

- Map each layer to HLS description
- Connect with FIFOs/streams
- Stitch together in IPI

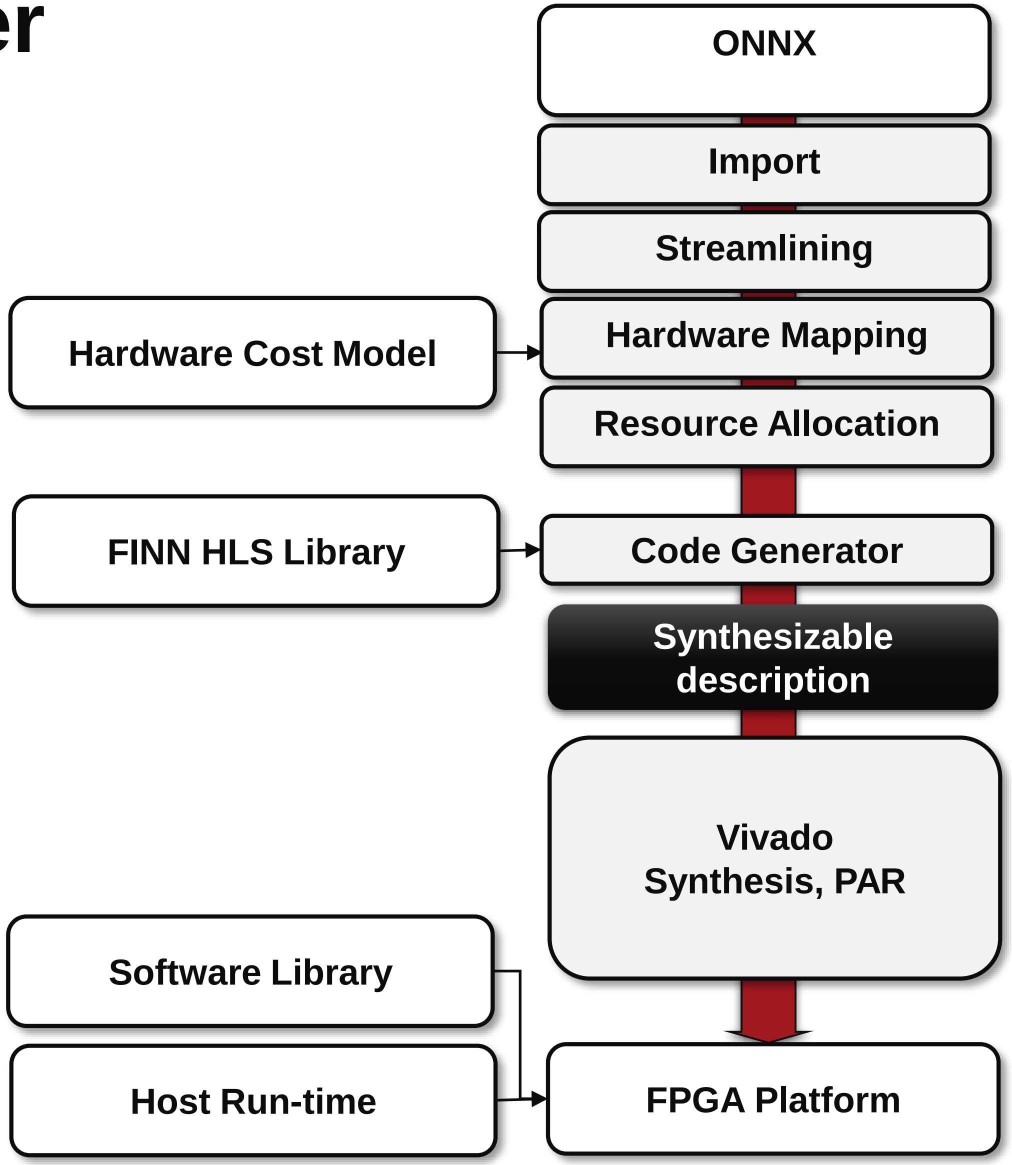


An Overview of the FINN Compiler

› Python library of graph transformations
  » Each consumes and produces an ONNX graph

› User calls sequence of transformations to create their own flow
  » Example end-to-end flows to get started

```
model = ModelWrapper("fpga4hep-bw%d.onnx" % bw)
model = model.transform(InferShapes())
model = model.transform(FoldConstants())
model = model.transform(GiveUniqueNodeNames())
model = model.transform(GiveReadableTensorNames())
model = model.transform(InferDataTypes())
model = model.transform(Streamline())
model = model.transform(ConvertBipolarMatMulToXnorPopcount())
model = model.transform(absorb.AbsorbAddIntoMultiThreshold())
model = model.transform(absorb.AbsorbMulIntoMultiThreshold())
model = model.transform(RoundAndClipThresholds())
model = model.transform(to_hls.InferBinaryStreamingFCLayer())
model = model.transform(to_hls.InferQuantizedStreamingFCLayer())
```



GitHub

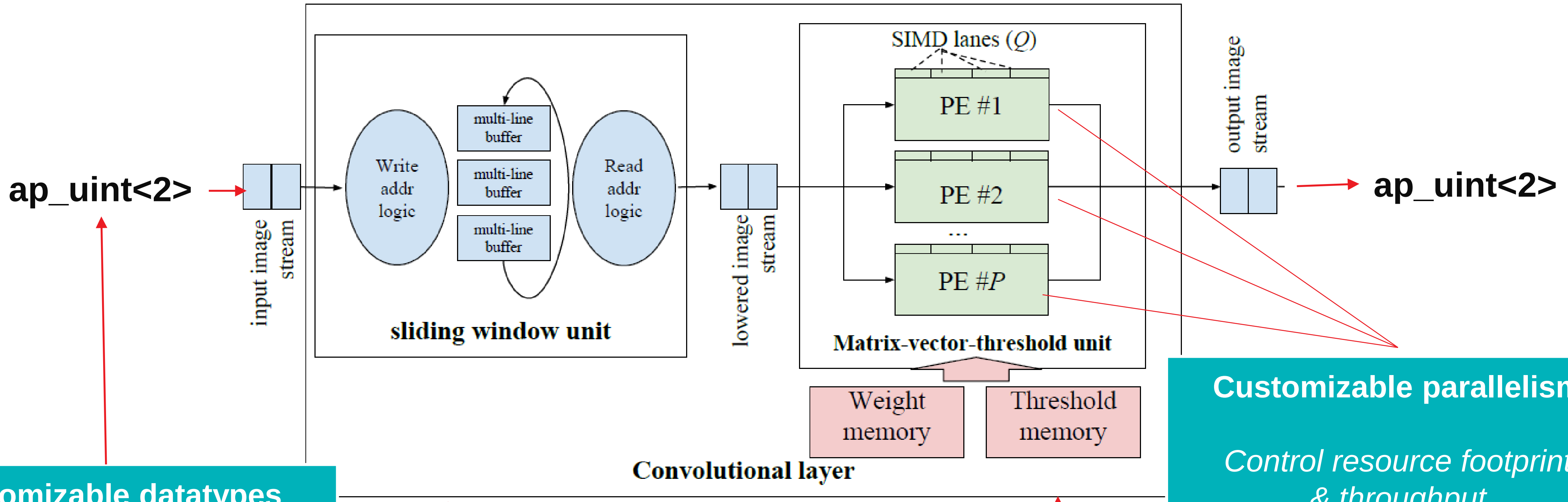
https://github.com/Xilinx/finn

XILINX

The FINN HLS Library

https://github.com/Xilinx/finn-hlslib

- An optimized, templated Vivado HLS C++ library of 10+ common DNN layers
- Key component: MVTU (Matrix Vector Threshold Unit)



Customizable datatypes

Flexibility through C++ templates

Streaming I/O

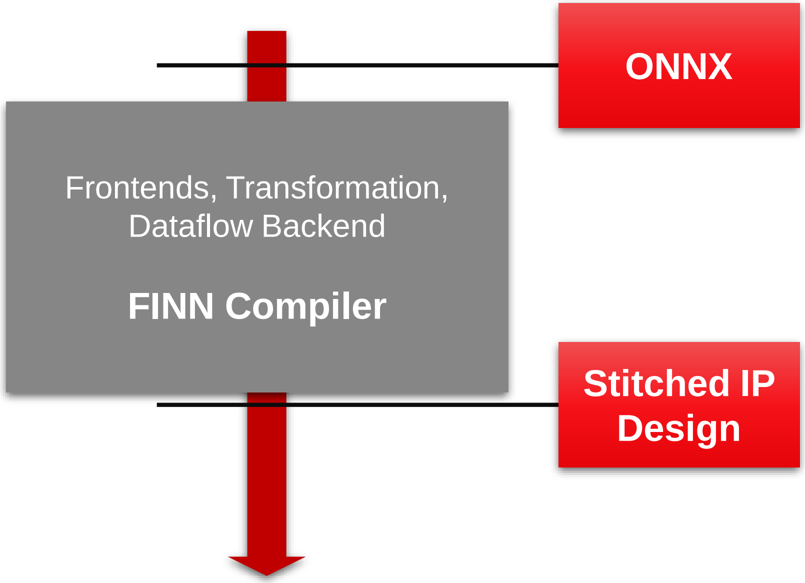
Easily compose modules together, low latency

Customizable parallelism

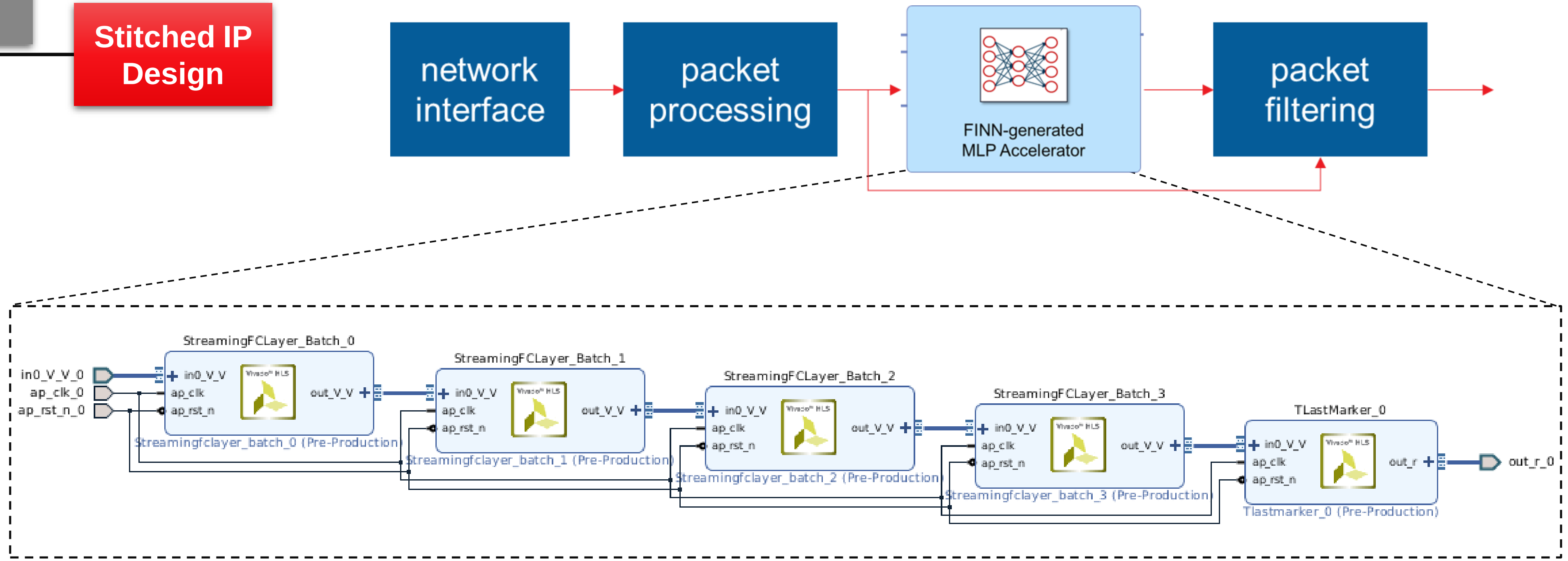
Control resource footprint & throughput

XILINX

How does the generated architecture look?



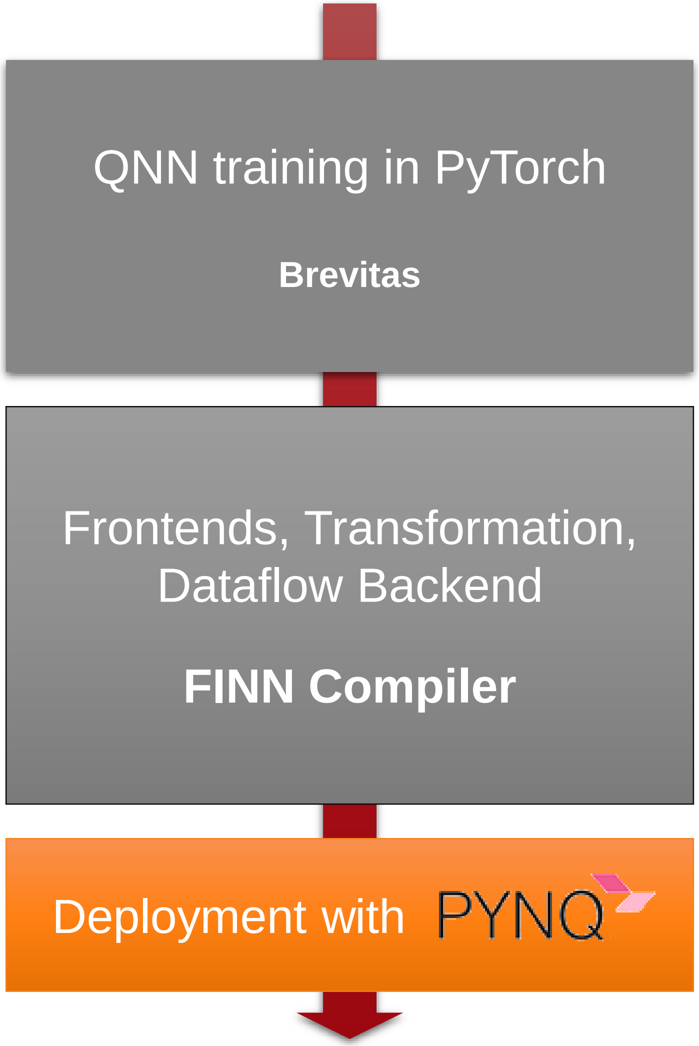
- › Stream-in, stream-out FPGA IP block
  - » Easy "bump-in-the-wire" integration into streaming systems
  - » Simple data movement, fully deterministic



XILINX

FINN

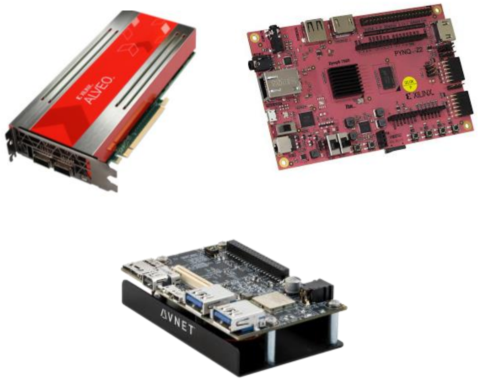
Deployment with PYNQ



XILINX

Deployment with PYNQ™ for Python Productivity

```
# numpy.ndarray shapes for i/o
ishape_packed = (1, 49, 2)
oshape_packed = (1, 1, 40)
# set up the DMA
dma.sendchannel.transfer(in_buf : numpy.ndarray)
dma.recvchannel.transfer(out_buf : numpy.ndarray)
# wait until all transfers complete
dma.sendchannel.wait()
dma.recvchannel.wait()
```



- Use PYNQ-provided Python abstractions and drivers
- User provides Numpy array in, calls driver, gets Numpy array out
  - Internally use PYNQ DMA driver to wr/rd NumPy arrays into I/O streams

GitHub

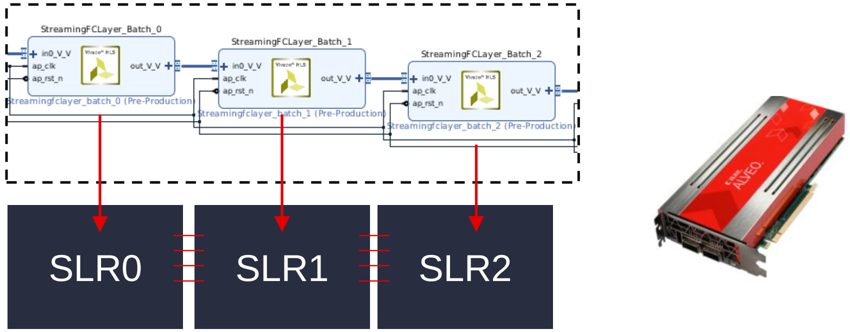
https://github.com/Xilinx/PYNQ
https://github.com/Xilinx/Alveo-PYNQ

XILINX

Upcoming FINN Features

Distributed (Multi-FPGA) Dataflow

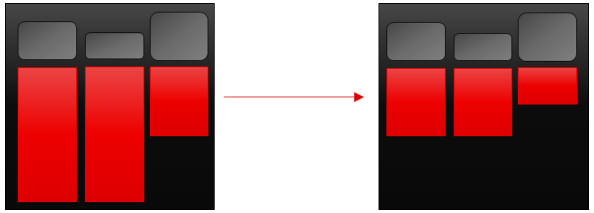
Scale-out performance



Automated Floorplanning for Multi-SLR FPGAs

Extract more performance from Alveo

100k LUT
10M FPS



10k LUT
1M FPS

Automated Folding

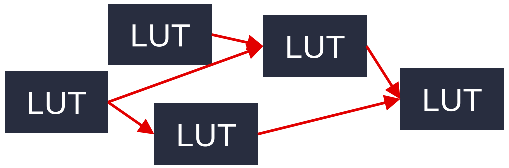
Quickly scale performance & resources without synthesis

Video Tutorials

How to train QNNs and deploy them with FINN

LogicNets

How Efficient Does Your DNN Need To Be?
A Spectrum of FPGA Inference Alternatives



less efficient
generic
broad scope

DPU, overlays
(10k+ FPS)

FINN
(10M+ FPS)

LogicNets
(100M+ FPS)

more efficient
co-designed
specialized

Layer-by-layer compute
(Matrix of Processing Engines)

Optimizing compiler/scheduler

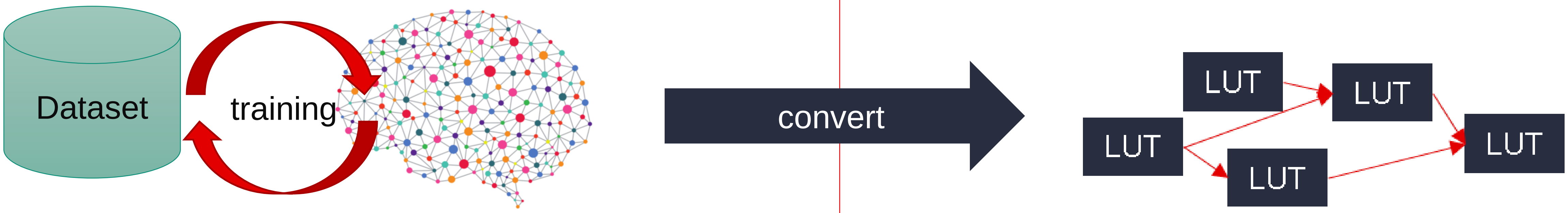
Generated heterogeneous streaming architecture

Custom topologies, arithmetic and hardware

The DNN ***is*** the circuit

Fully unfolded, pipelined, feedforward datapaths

LogicNets at a Glance



Specialized DNN Topology

(*with* ***high sparsity***
+ ***activation quantization)***

Fully-Spatial
Circuit Implementation

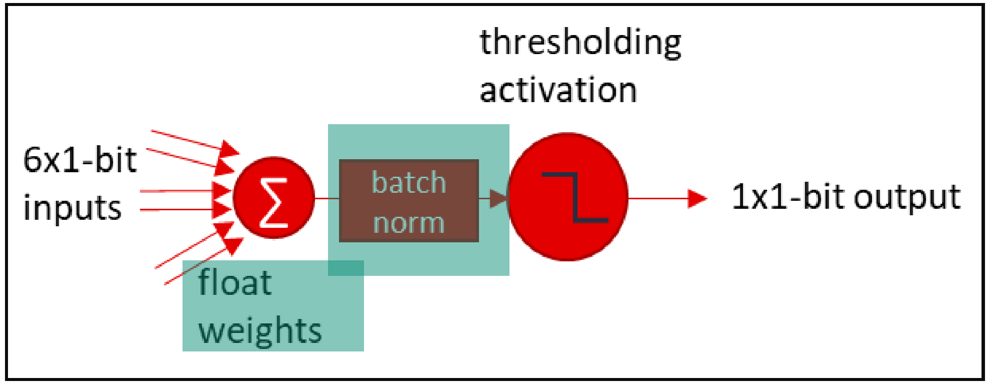
one full sample every clock
low logic depth, high F_{clk}
100M's of samples per second

PyTorch | **FPGA**

Key idea: Quantized Neurons as Truth Tables

Neuron Equivalent (NEQ)

Hardware Building Block (HBB)



convert
(enumerate inputs)

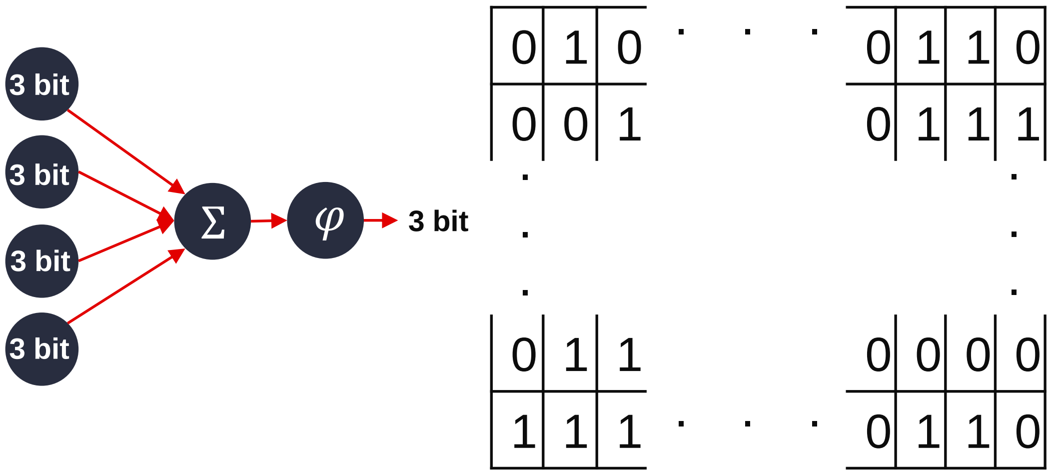
Total input: 6 bits
Total output: 1 bit

Total input: 6 bits
Total output: 1 bit

Hardware cost: 1 x LUT6

PyTorch | **FPGA**

Prohibitive Cost of Implementing Large Truth Tables



Total input: 12 bits
Total output: 3 bit

Truth Table Size: $4096 \times 15 = 2^{3*4} \times 3 * 5$

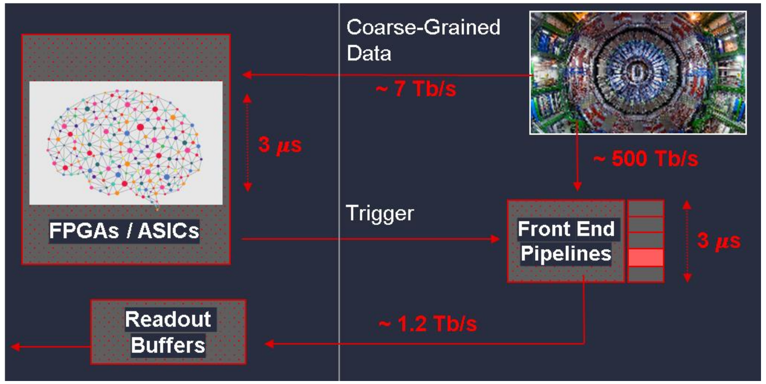
Co-design DNN topology to avoid intractably large LUTs: high sparsity + few-bit activations

LUT6 cost of the neuron: ~4095 LUT6s

XILINX

LogicNets Key Results

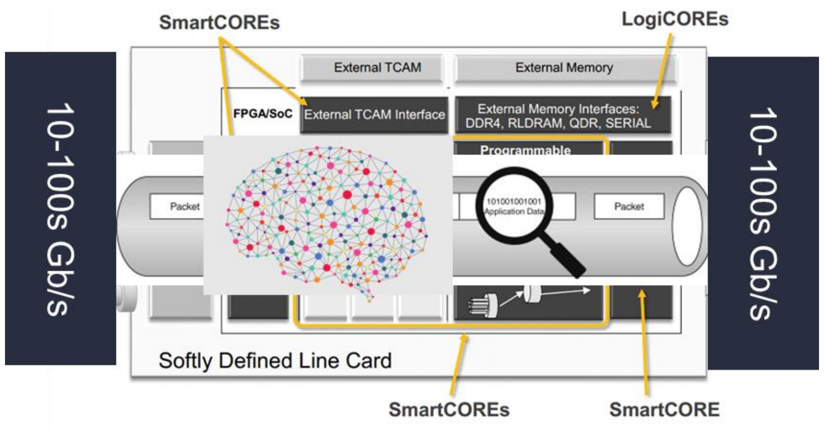
hls4ml JSC dataset [Duarte et al.]



Jet Tagging (CERN LHC)

~**72%** accuracy
using ~**38k** LUTs
at **427 M** samples / second
with **13 ns** latency

UNSW-NB15 Network Intrusion Detection dataset [Moustafa et al.]



Network Intrusion Detection

~**91%** accuracy
using ~**16k** LUTs
at **471 M** samples / second
with **9 ns** latency

[Umuroglu et al., FPL'20]
Preprint: https://arxiv.org/pdf/2004.03021
Video: https://youtu.be/qCyK5v84jpI

Conclusion

- FINN
  - QNN solution stack from training to custom dataflow architecture
  - Full co-design environment with growing library examples
  - Flexible, customizable open-source compiler framework

- LogicNets
  - Sparse + quantized topology converts directly to LUT circuit
  - Many exciting future research directions
  - To be open-sourced as part of FINN ecosystem (~Q1 2021)

Join our Growing Open-Source Community!

GitHub

https://xilinx.github.io/finn

Stanford University

UNC CHARLOTTE

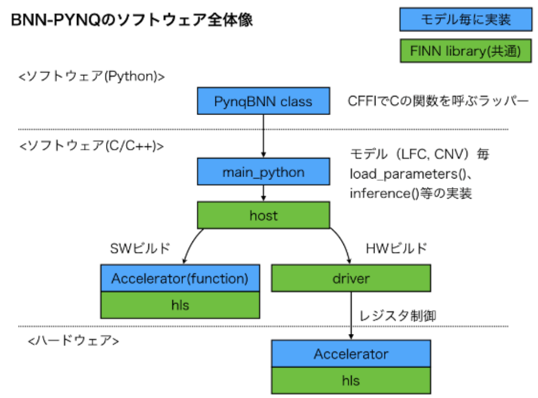
hackster.io

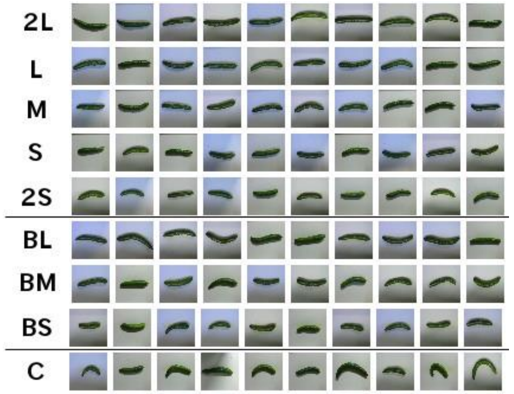
AN AVNET COMMUNITY

NTNU

Knowledge for a better world

University courses, student/hobbyist projects





Japanese documentation effort + «cucumber sorting»

Sketch Recognition (Xilinx Edinburgh)

XILINX

Thank You