XILINX
ALL PROGRAMMABLE™

A Framework for Reduced Precision Neural Networks on FPGAs

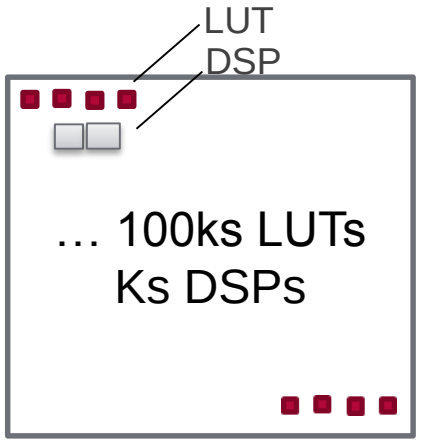
Kees Vissers
Xilinx Research

- Background
- CNNs and their challenges
- Quantized neural networks
- FINN: Framework for fast exploitation of Quantized Neural Networks
- Experimental results
- Conclusion

What are FPGAs?

Programmable devices that contain:

- MAC (DSP48) for floating point, 16 bit integer, 8 bit integer
- Logic Lookup tables (LUTS) for any function at bit-precision, including 2 bit and 1 bit MAC (xnor, popcount), and compression, security, etc.
- Large number of flexible memory blocks, with high internal bandwidth
- High-speed I/O, good external memory interfaces
- ARM cores
- Family of devices



Customizable hardware architectures with fine-grain programmability

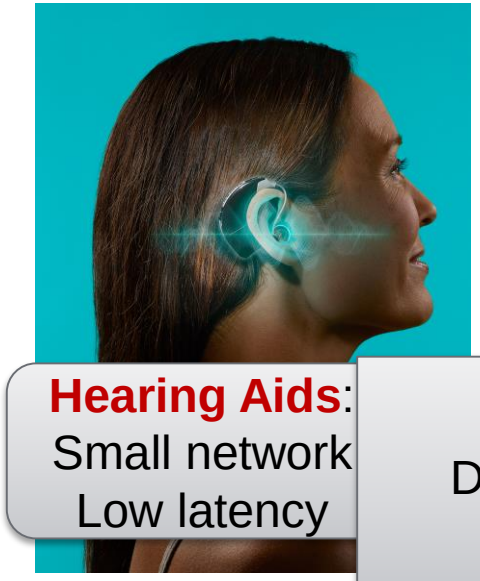
Challenge 1: Diverse Applications with Diverse Design Targets

Translate & AlphaGo:
Huge networks

AlphaGo

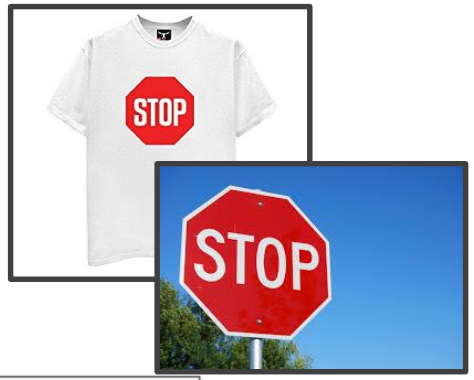
Medical Diagnosis:
Small networks

Robotics:
Real-time



Hearing Aids:
Small network
Low latency

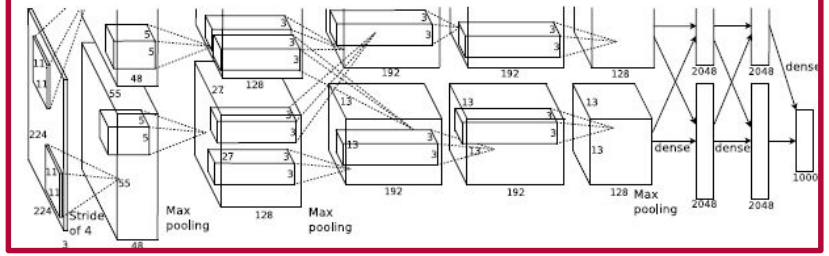
ADAS
High accuracy
Low latency



Challenge 1:
Different use cases require different networks & different figures of merits
(speed, latency, energy, accuracy)

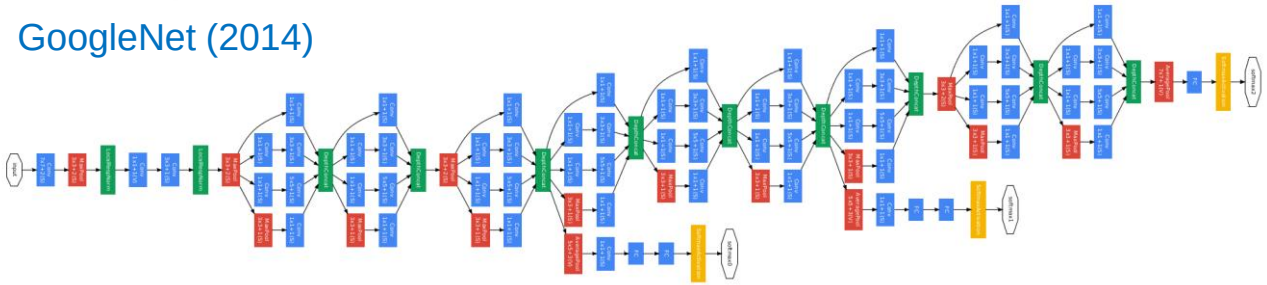
Challenge 2: Neural Networks Will Continue to Change

AlexNet (2012)

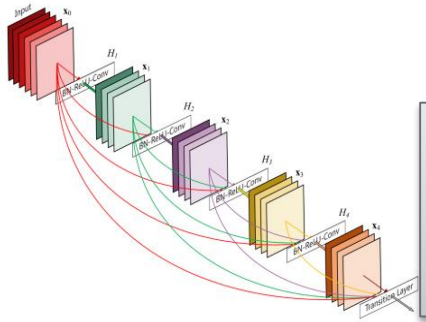


Number and **types** of layers are changing

GoogleNet (2014)



DenseNet (2016)

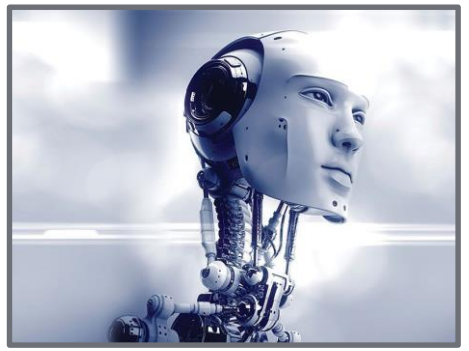


Data representations and **quantization methods** are changing

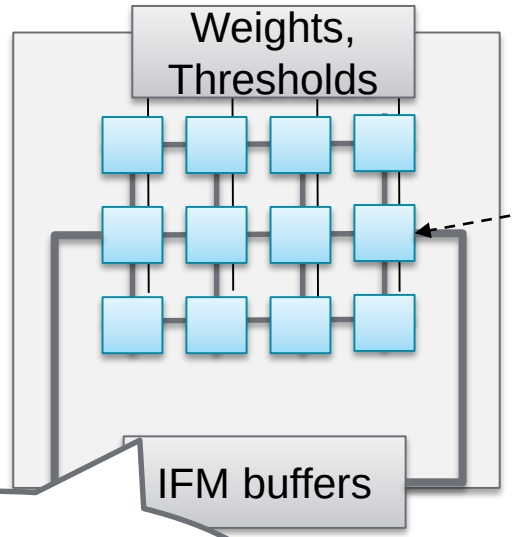
Graph Connectivity is changing

Challenge 2:
Continuous stream of new algorithms

Customized ML Processor Datapath

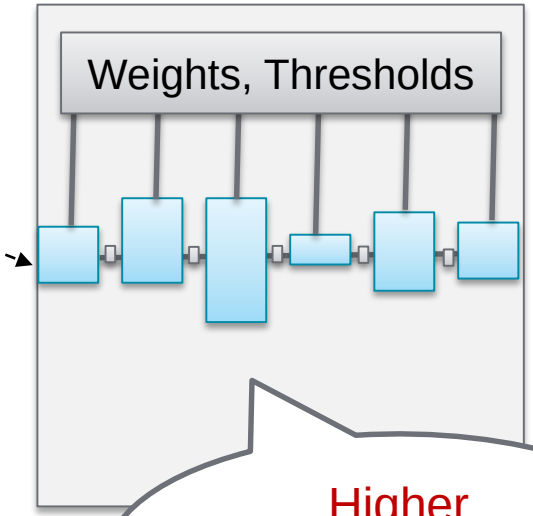


Generality vs Performance
Latency vs Resources



DSPs, LUTs,…

Customizable operations

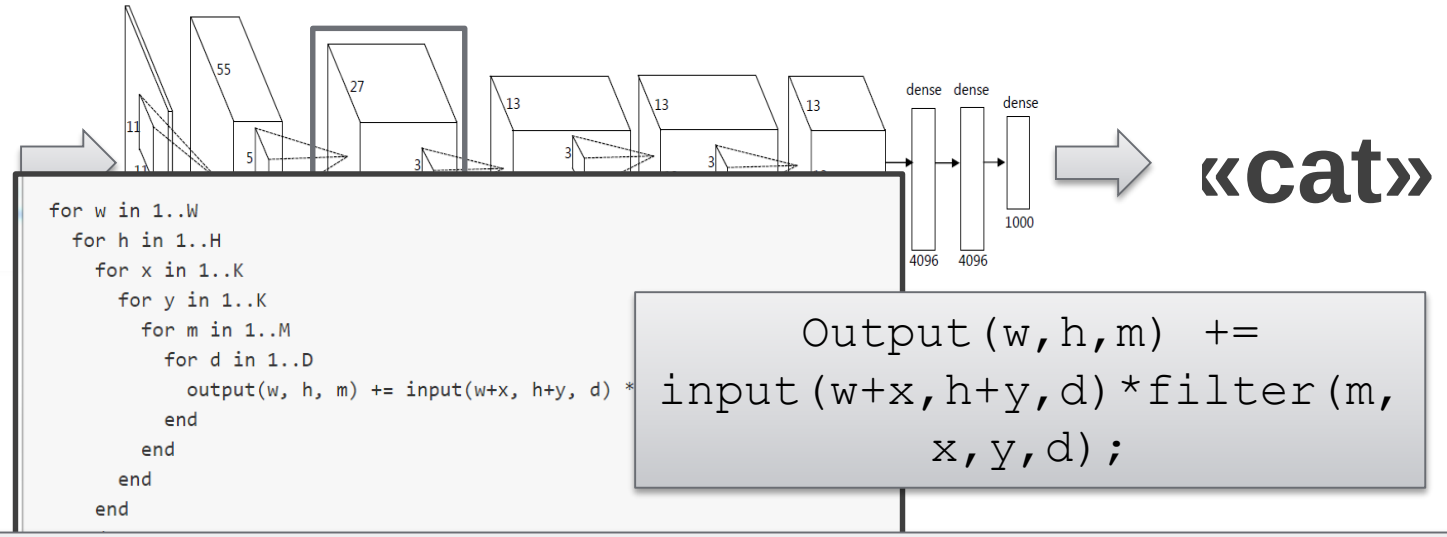


Flexible (all NNs), less resources

Higher performance, lower latency

Challenge 3: Highly Compute and Memory Intensive

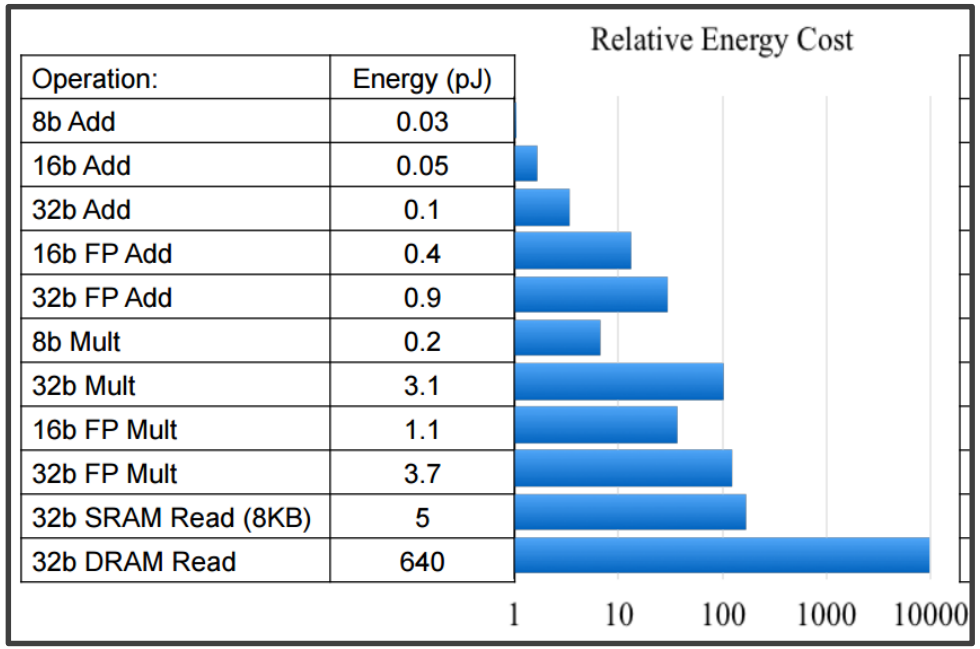
- **The predominant CNN computation is linear algebra**
  - Demands lots of (simple) computation and lots of parameters (memory)
    - AlexNet: 244MB & 1.5GOPS, VGG16: 552MB & 30.8GOPS; GoogleNet: 41.9MB & 3.0GOPS for ImageNet



Challenge 3:
billions of multiply-accumulate ops & tens of megabytes of parameter data

Increasingly Reduced Precision Networks

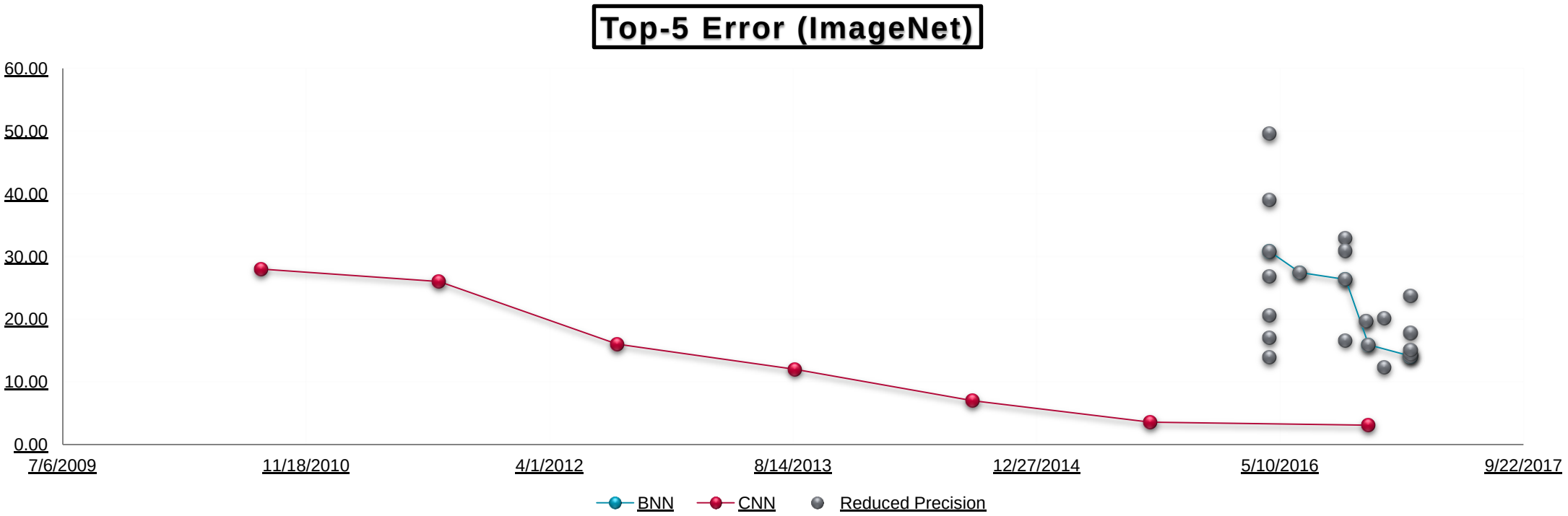
- **Floating point (FP) CNNs contain a lot of redundancy**
- **Reducing precision is shown to work down to 1b with minimal loss of accuracy –**
  - ICLR 2017 with ternary weight networks on par with FP for AlexNet top-1 and top-5, ResNet20,32,44,56
  - Accuracy gap is closing
- **Reducing precision brings numerous advantageous**
  - Power
  - Performance
  - Memory requirements
  - Not just for FPGAs



Source: Bill Dally (Stanford), Cadence ***Embedded Neural Network Summit****, February 1, 2017*

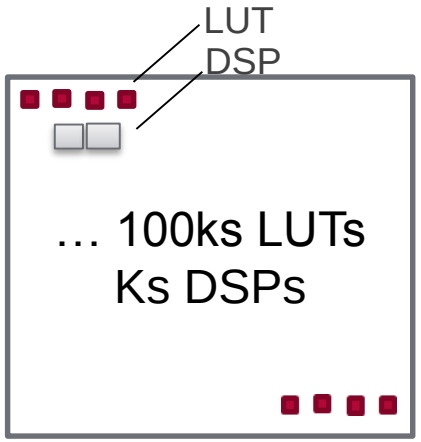
Accuracy of Quantized Neural Networks (QNNs) Improving

Published Results for FP CNNs, QNNs and binarized NNs (BNNs)



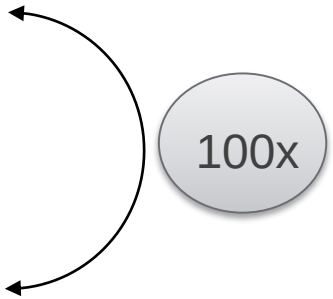
- Accuracy results are improving rapidly through for example new training techniques, topological changes and other methods

Potential of Reduced Precision on FPGAs



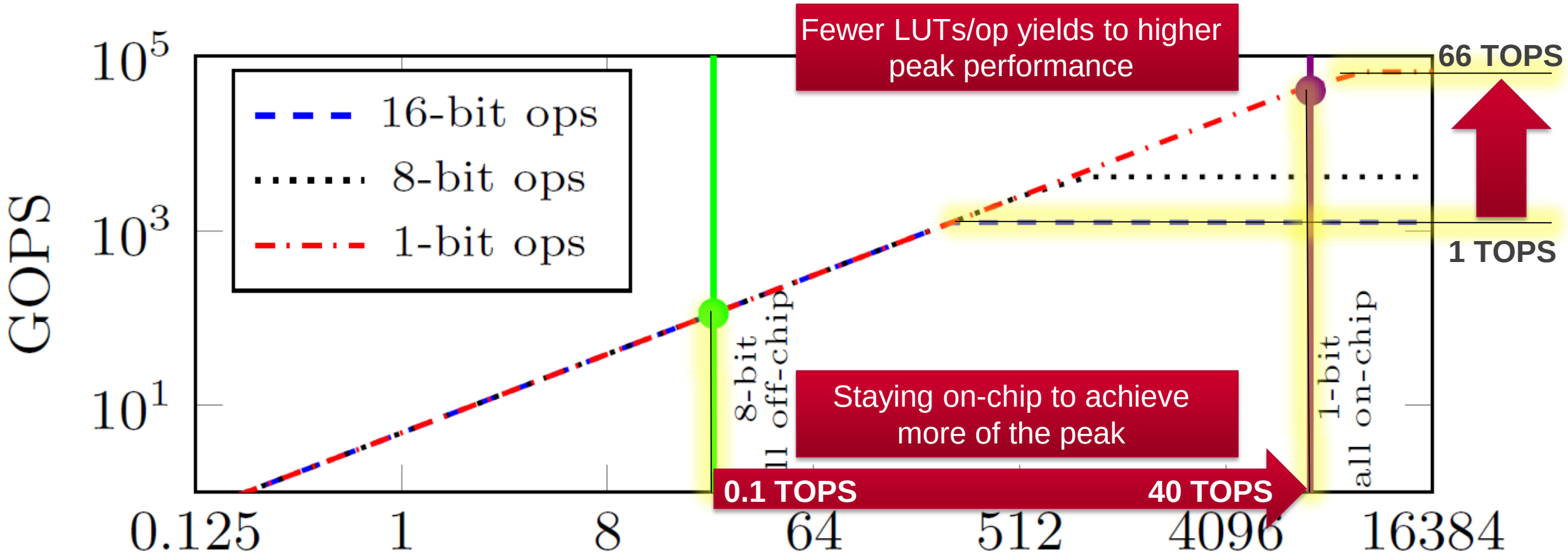
- **Cost per operation is greatly reduced**
  - For example, for BNN: FP multiply accumulate becomes XNOR with bit counts
- **Memory cost is greatly reduced**
  - Large networks can fit entirely into on-chip memory (OCM) (UltraRAM, BRAM)
- **Today's FPGAs have a much higher peak performance for reduced precision operations**
  - FPGA performance is anti-proportional to the cost per operation when applications are sufficiently parallel
  - Lower cost per op & massively parallel = more ops every cycle

Precision	Cost per Op LUT	Cost per Op DSP	MB needed (AlexNet)	TOps/s (KU115)*	TOps/s (VU9P)**	TOps/s (ZU19EG)*
1b	2.5	0	7.6	~46	~100	~66
4b	16	0	30.5	~11	~15	~16
8b	45	0	61	~3	~6	~4
16b	15	0.5	122	~1	~4	~1
32b	178	2	244	~0.5	~1	~0.3



*Assumptions: Application can fill device to 70% (fully parallelizable) 250MHZ
**Assumptions: Application can fill device to 70% (fully parallelizable) 300MHZ

Potential of QNNs on FPGAs (ZU19EG)

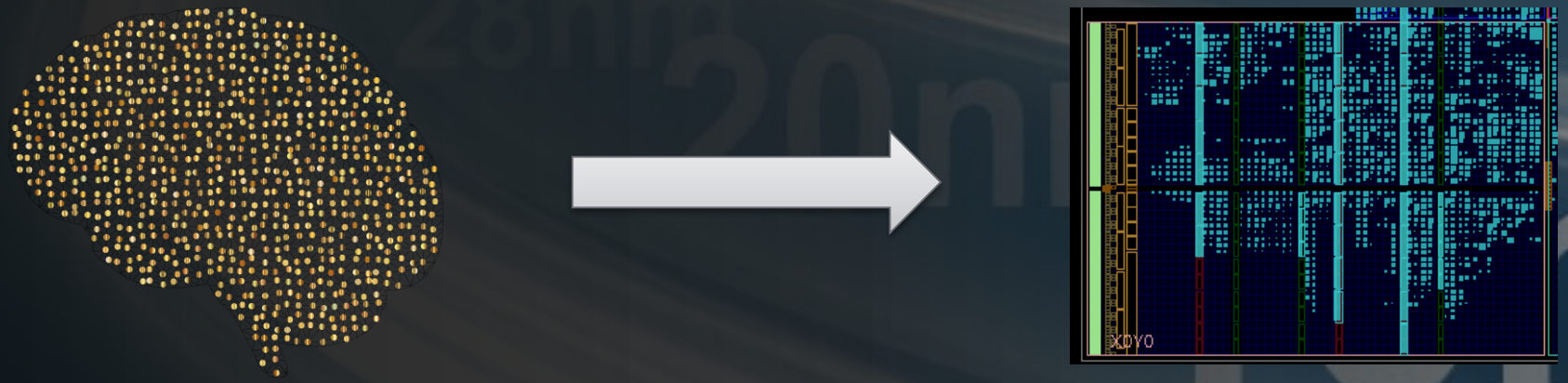


- Reduced Precision allows to scale NN performance on FPGAs to unprecedented levels

Assumption: O

Exploitation of Quantized Neural Networks through

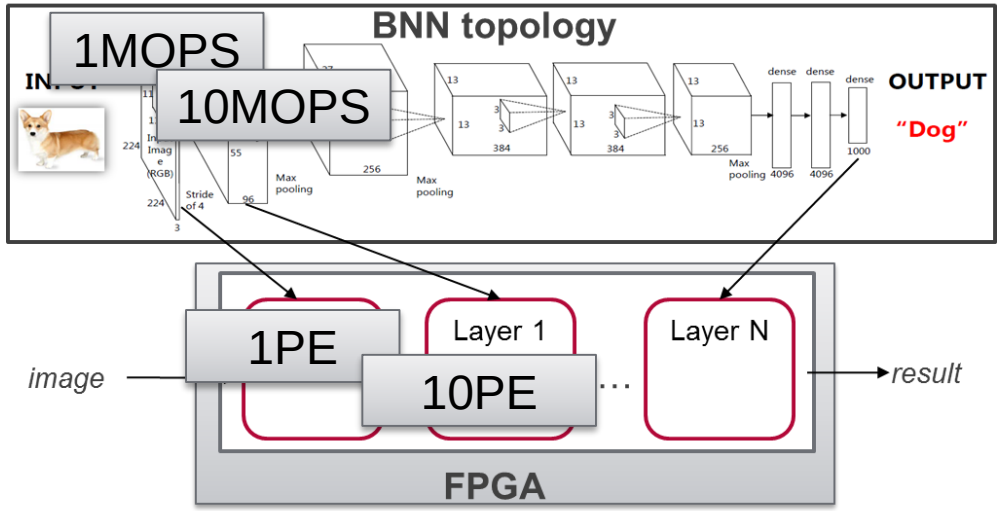
FINN: A Framework for Fast, Scalable Neural Network Inference



https://arxiv.org/abs/1612.07119
http://arxiv.org/abs/1701.03400

FINN Design Principles

- **Custom-tailored hardware**
  - Customized data types
  - Customized dataflow architecture to match network topology
- **Keep all parameters on-chip, if possible**
- **C++ design entry**
  - To support portability, scalability & rapid exploration



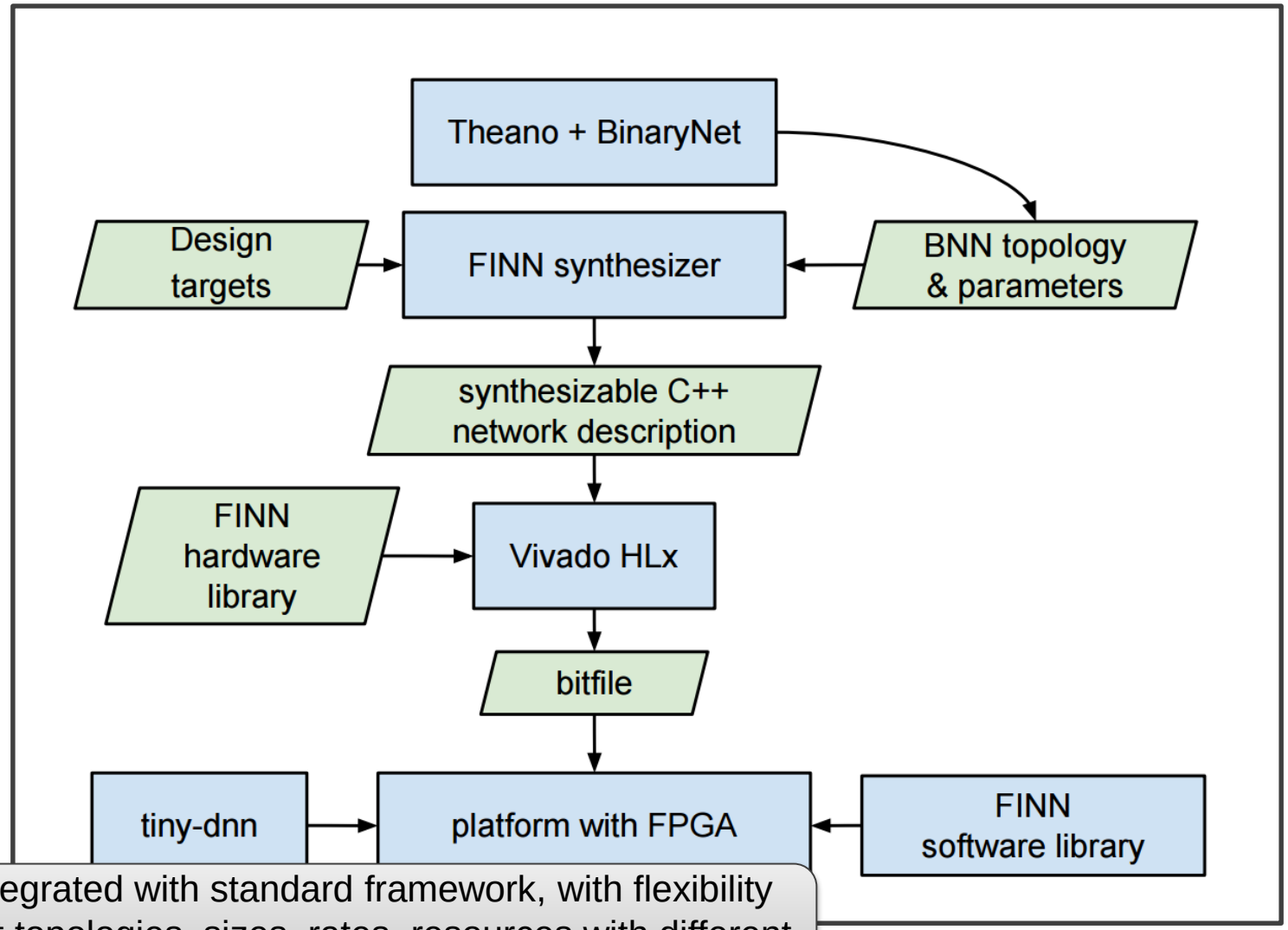
Customized Dataflow Architecture

Work Flow for Exploration of NNs of FPGAs

- Integration with tiny-dnn and Theano, Tensorflow and Caffe

theano TensorFlow™ Caffe

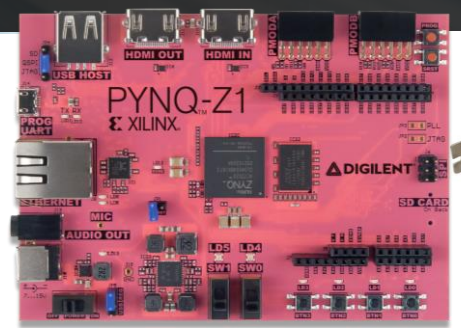
- All code in C/C++
- Can execute on CPU and FPGA
  - No RTL needed



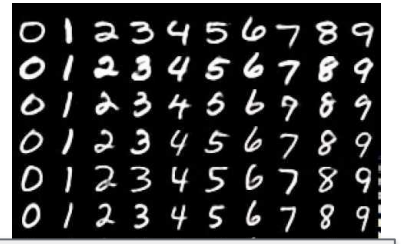
Fast workflow, integrated with standard framework, with flexibility to support different topologies, sizes, rates, resources with different devices (Z7045, KU115, Z7020)

Experimental Results

- Embedded platforms (Zynq Z7045 & 7020): ZC706, PYNQ open source platform
- Server class accelerator: ADM_PCIE_8K5 & TUL Accel. kit in OpenPOWER (& x86 with SDAccel)



Input Data

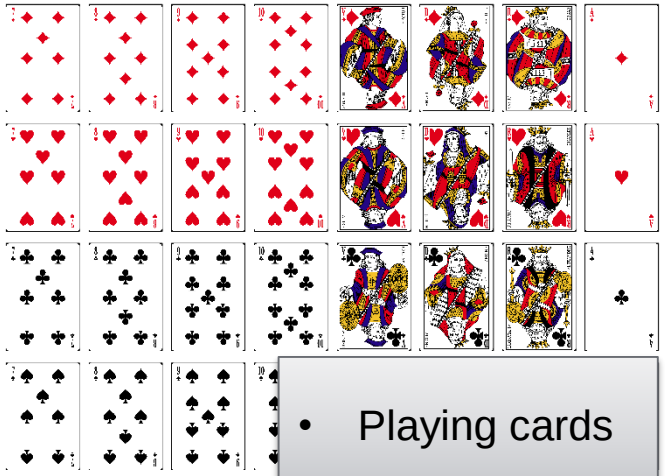


- MNIST handwritten digits

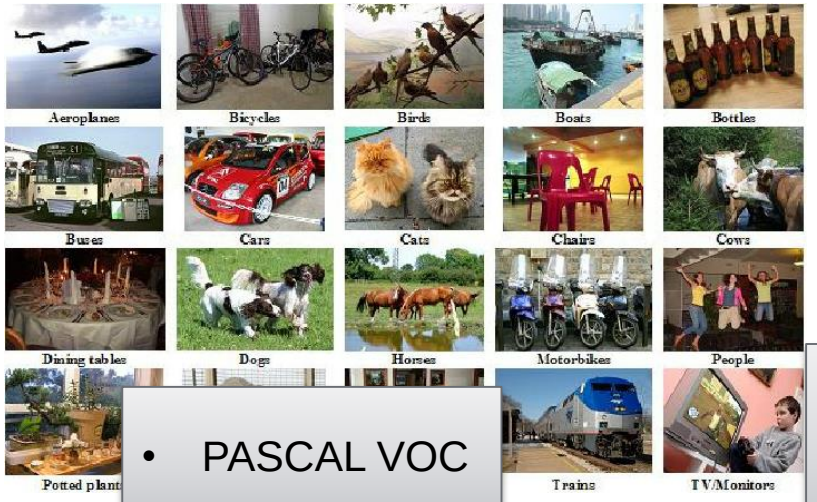
- Streetview house numbers

- German road signs

- Cifar-10: cats, dogs, etc



- Playing cards

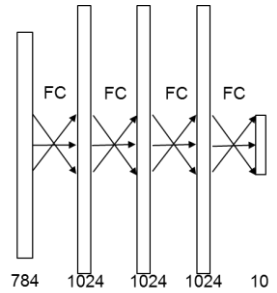


- PASCAL VOC

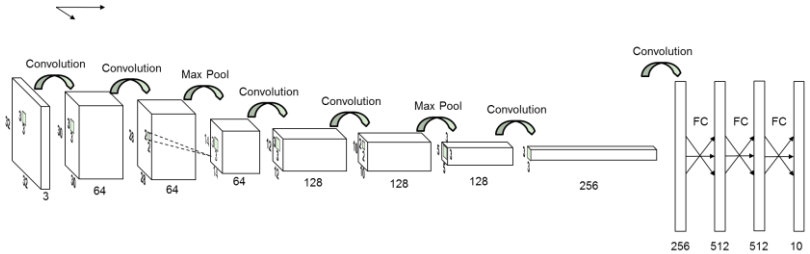
- Imagenet

Test Networks

- **Multilayer Perceptron**
  - Input images: 28x28 pixels, black-white (MNIST)
  - Up to 5.8MOPS/frame



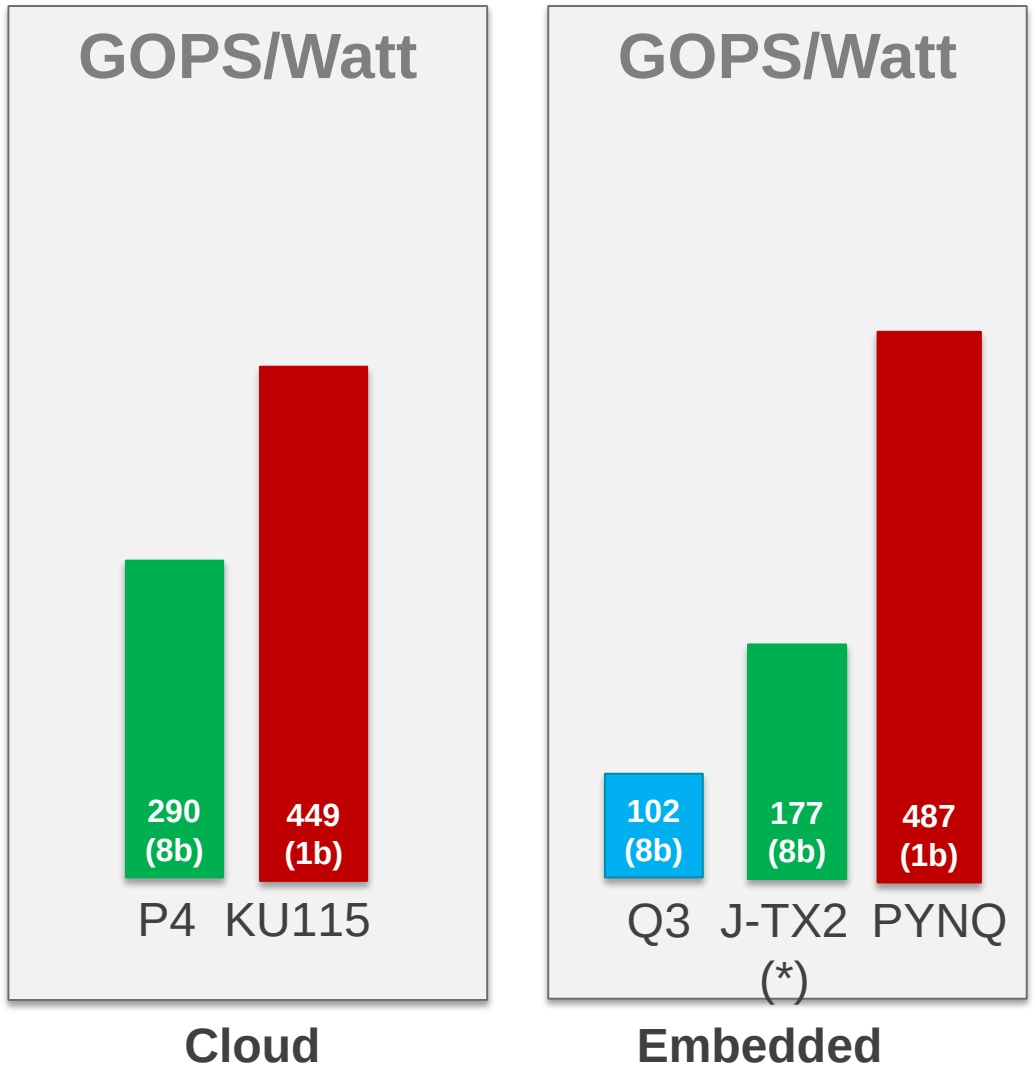
- **VGG-16 derivative**
  - Input images: 32x32 pixels, RGB image (SVHN, CIFAR-10, traffic signs, playing cards)
  - Up to 1.2GOPS/frame

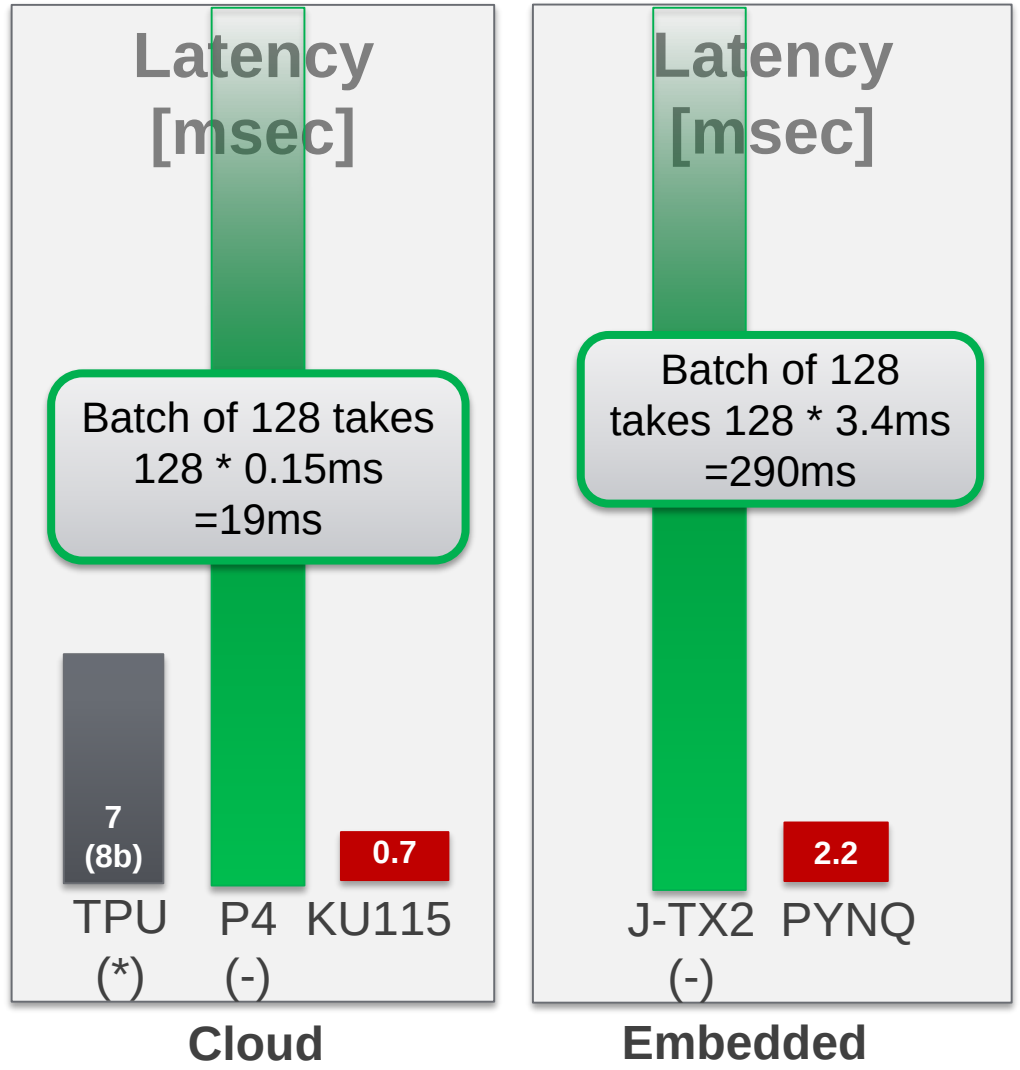


- **DorefaNet**
  - Input images: 226x226 pixels, RGB (ImageNet)
  - Up to

- **YoloV2, TinyYolo**
  - Input images: 448x448, RBG (VOC, COCO)
  - 35 and 7GOPS/frame

QNN Results - Latency, Performance/Power

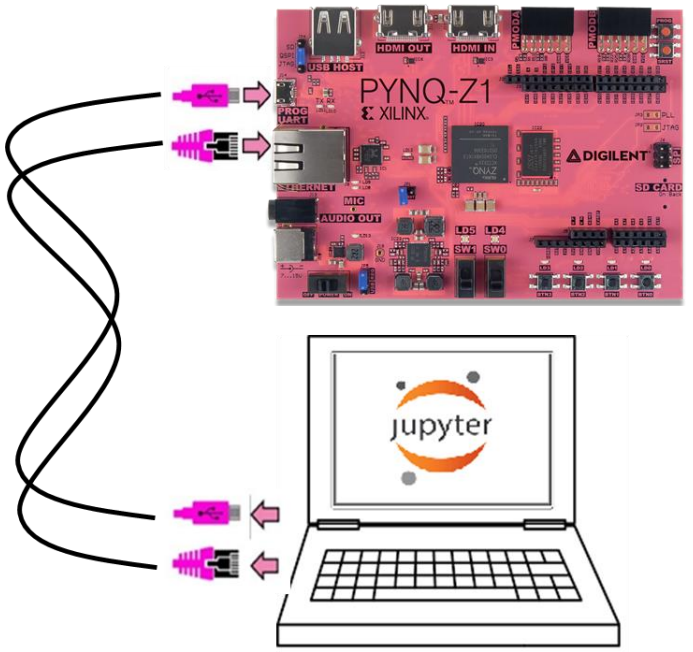




*: claimed
-: estimated

PYNQ FINN Open Source Release

- **FINN is open sourced and available at**
  - https://github.com/Xilinx/BNN-PYNQ
  - New features are continuously rolled out
- **Supported on low cost open source platform Pynq**
  - Visit www.pynq.io
- **Easy to use with precooked overlays & examples**
  - BNNs
- **1000x faster than Raspberry Pi3**



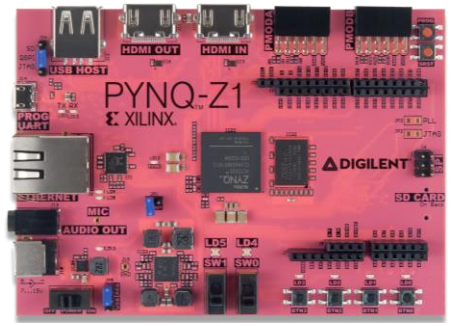
Z7020	FPS (FPGA)	GOPS/s	BRAM	LUT	Latency [us]	Power [W]
LFC	**168k**	974	112 (80%)	30.6K (57.6%)	**102**	<2.5
CNV	3.04k	341	140 (100%)	28.5K (53.5%)	**1580**	<2.5

Z7020 ARM FPS	Raspberry Pi3 FPS
17.3	44.3
1.2	2.3

Summary

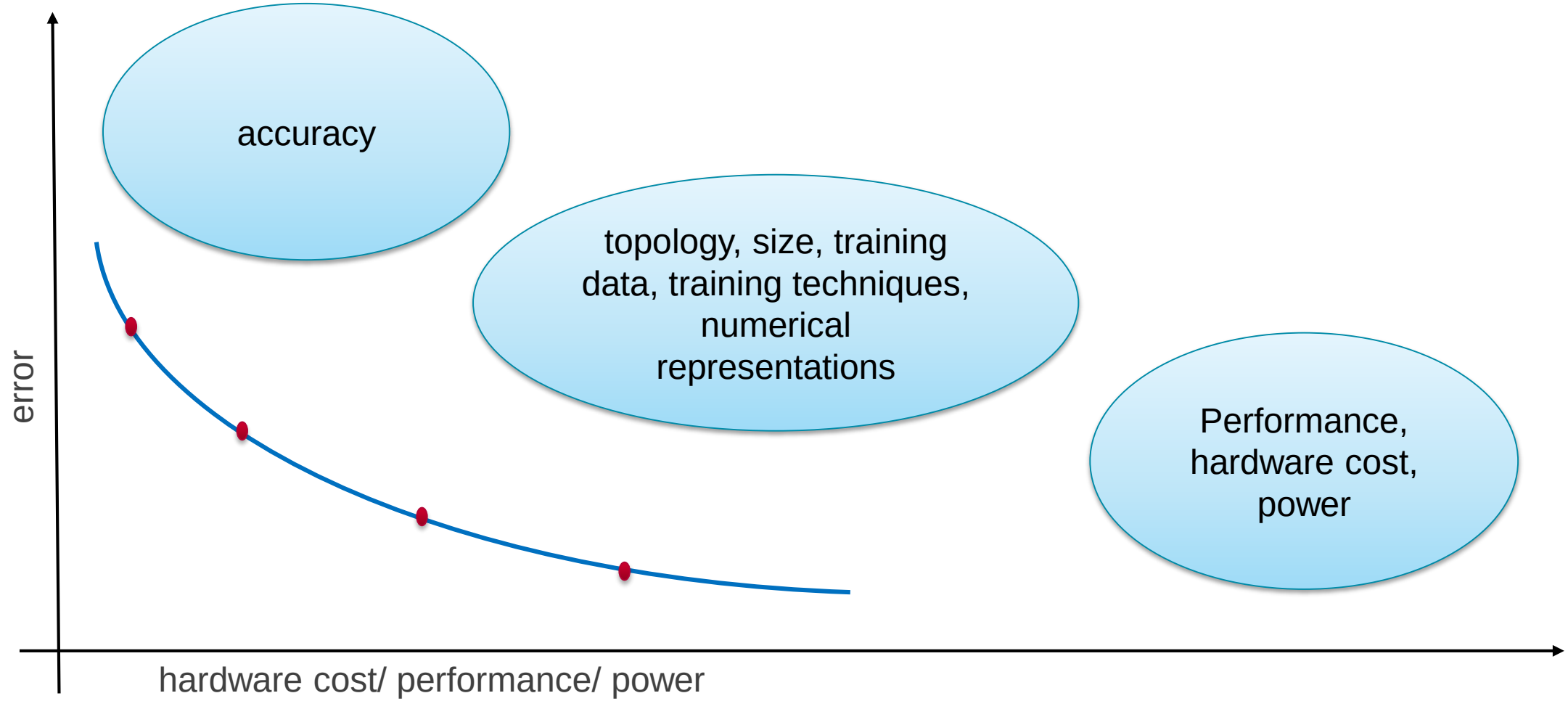
- **Benefits of FPGA implementations:**
  - Extreme performance with reduced precision
  - Low latency through dataflow – no batching needed
  - Flexibility
  - Low power total solution: keep data on chip, compress data, compute at reduced precision (good for memory bandwidth too)

- **Get started with FINN & Pynq**
  - www.pynq.io
  - https://github.com/Xilinx/BNN-PYNQ



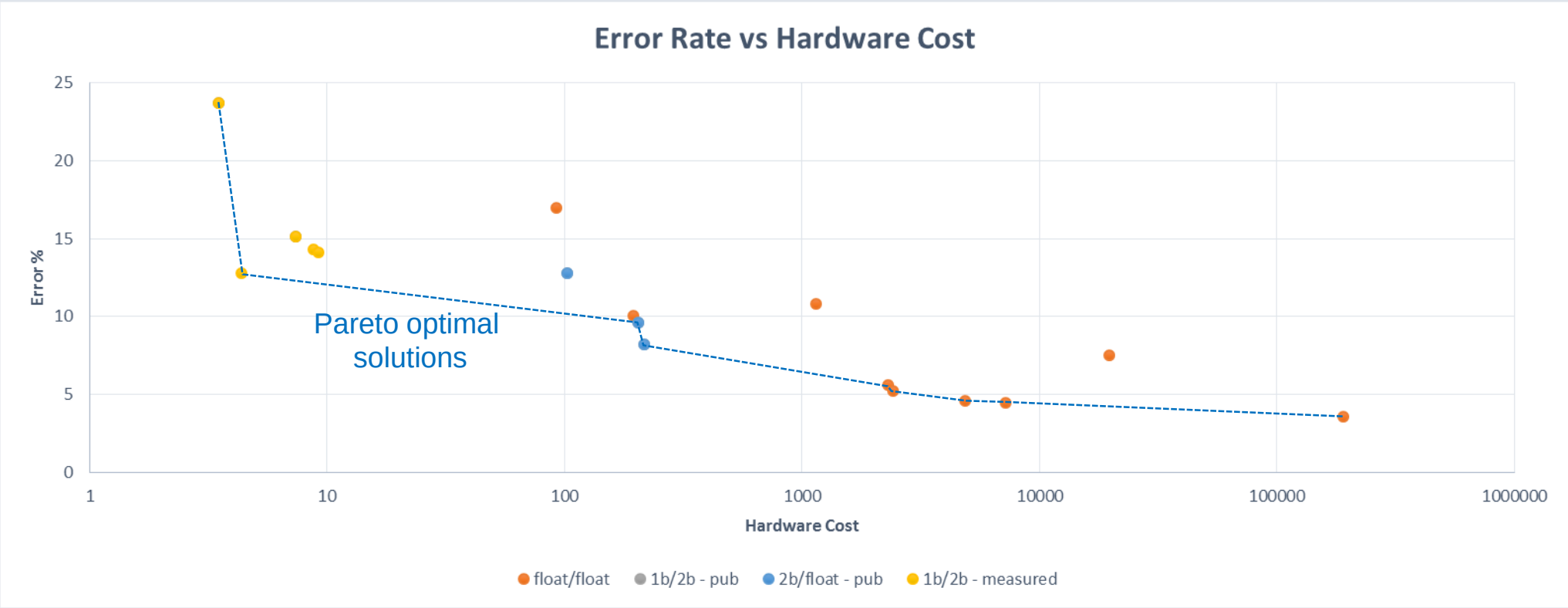
The Name of the Game: Designing Hardware-Optimal CNNs

It's a trade-off…



Example: ImageNet Classification

Published Results & Xilinx Research internal Experiments



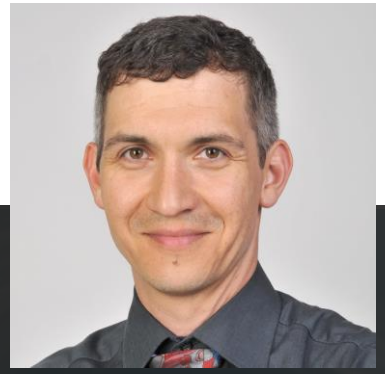
Floating point is too expensive
Below 10% uses ensembles and cost likely prohibitively high
Pareto optimal: 1b – 8b provide good compromises

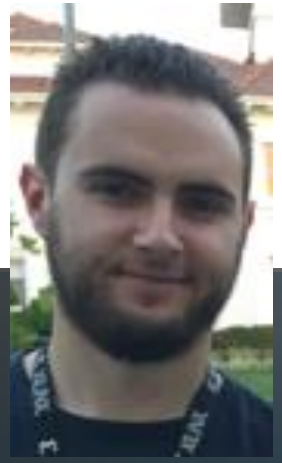
Inference Accelerators – Accuracy vs Hardware Cost for a fixed topology

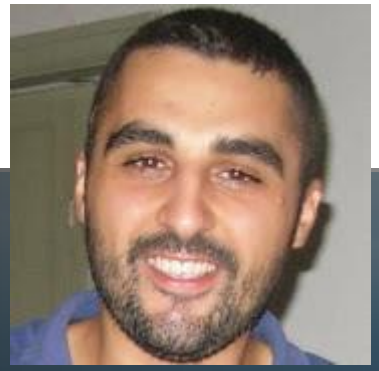
- Just reducing precision, reduce hardware cost & increases error
- Recuperate accuracy by retraining & increasing network size
- 1b, 2b and 4b provide pareto optimal solutions

Conclusions: We're only at the start…

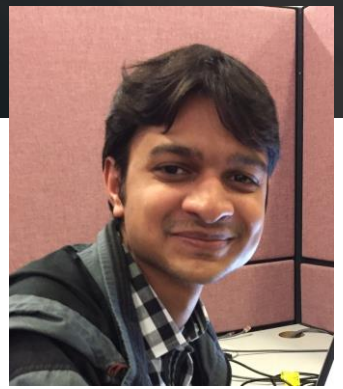
- **We presented a framework for exploring Neural Networks at any precision ranging from 32bit Floating Point to 1bit for weight and activation.**
- **We have shown that for a number of cases optimal networks can use compute and storage below 8bit!**
- **Lots of scope for research in exploring the design space between accuracy, performance, cost, power etc.**
- **Very Exciting times for Neural Networks on heterogeneous platforms.**

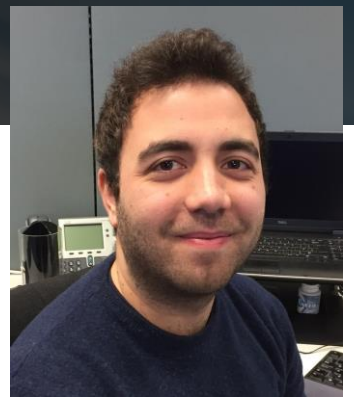


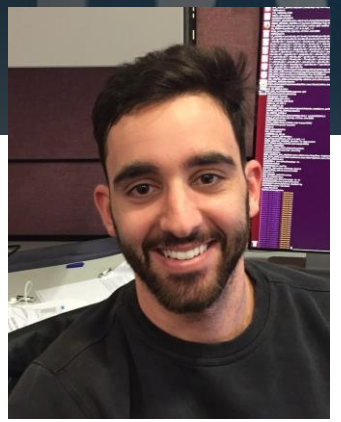


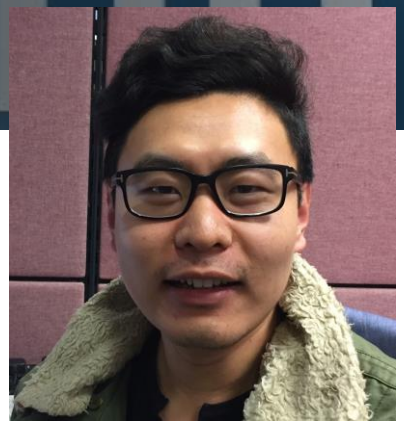


- Thanks to a large team at Xilinx including Xilinx Research Ireland









Technical Details on Finn architectures

Architecture of a Matrix-Vector Threshold Unit (MVTU)

- **Fully connected layers & convolutional layers are mapped on matrix-vector multiply threshold units (MVTUs)**
- **MVTUs support OFM (neuron) and folding over weights (synaptic)**
- **Weight and output stationary (weights and popcounts are retained locally)**
- **Max pool units are optionally placed behind MVTUs**

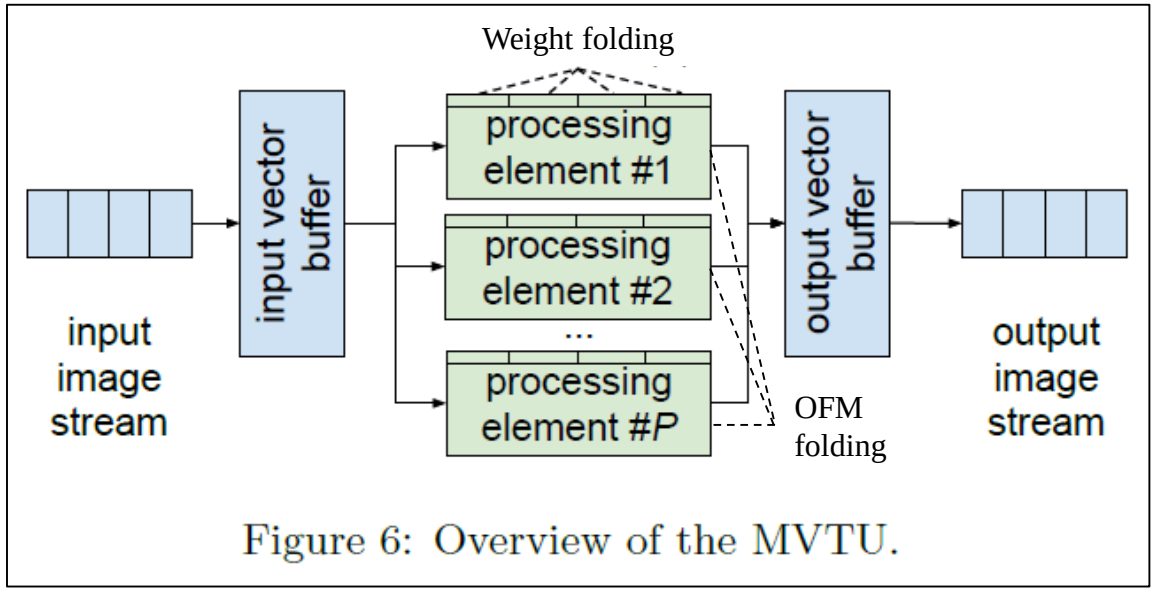


Figure 6: Overview of the MVTU.

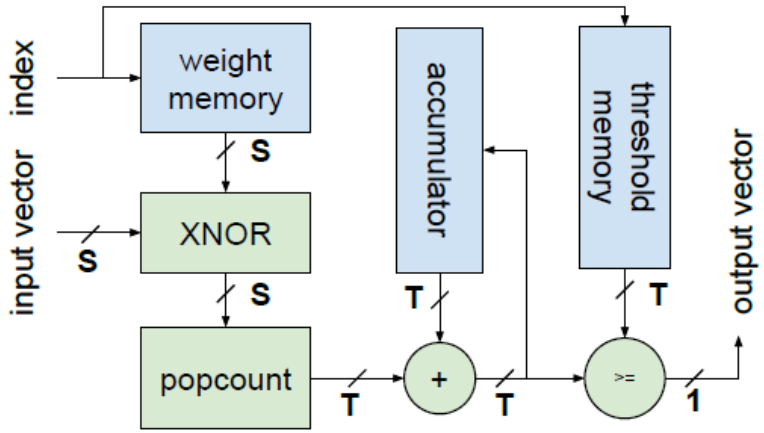


Figure 7: MVTU Processing Element (PE) datapath. **Bold** indicates bitwidth.

Synthesizable C++ Network Description

```
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);
}
```

Stream definitions

Move image in from PS memory

Layer instantiation
connected by streams

Move results to PS memory

MVTU

```
for (unsigned int nm = 0; nm < neuronFold; nm++) {
    for (unsigned int sf = 0; sf < synapseFold; sf++) {
#pragma HLS PIPELINE II=1
          ap_uint<SIMDWidth> inElem;
          if (nm == 0) {
             inElem = in.read();
             inputBuf[sf] = inElem;
          } else {
             inElem = inputBuf[sf];
          }
          for (unsigned int pe = 0; pe < PECount; pe++) {
#pragma HLS UNROLL
             ap_uint<SIMDWidth> weight = weightMem[pe][nm * synapseFold + sf];
             ap_uint<SIMDWidth> masked = ~(weight ^ inElem);
             accPopCount[pe] += NaivePopCount<SIMDWidth, PopCountWidth>(masked);
          }
    }
    ap_uint<PECount> outElem = 0;
    for (unsigned int pe = 0; pe < PECount; pe++) {
#pragma HLS UNROLL
          outElem(pe, pe) = accPopCount[pe] > thresMem[pe][nm] ? 1 : 0;
          accPopCount[pe] = 0;          // clear the accumulator
    }
```

Folding

Reading
Inputs or consume
internal (when folded)

Indexing weight and
threshold memory
binary MAC

Batchnorm
activations

Architecture of Infrastructure on Zynq SOC