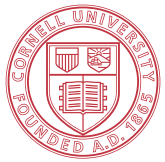
Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs

Ritchie Zhao[1], Weinan Song[2], Wentao Zhang[2], Tianwei Xing[3], Jeng-Hau Lin[4], Mani Srivastava[3], Rajesh Gupta[4], Zhiru Zhang[1]

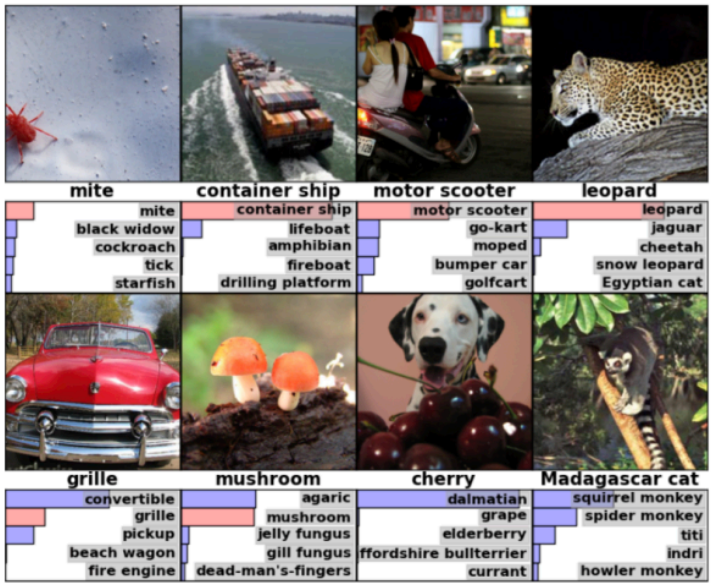
[1] Electrical and Computer Engineering, Cornell University
[2] Electronics Engineering and Computer Science, Peking University
[3] Electrical Engineering, University of California Los Angeles
[4] Computer Science and Engineering, University of California San Diego

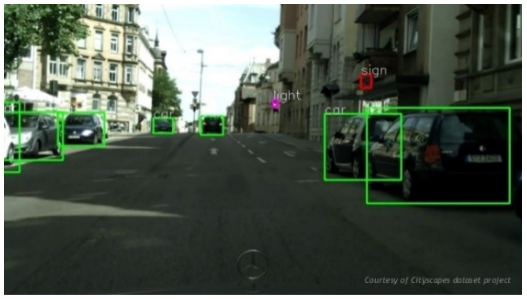


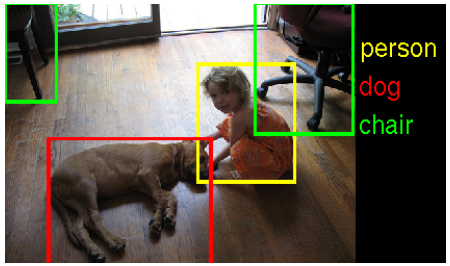
Cornell University

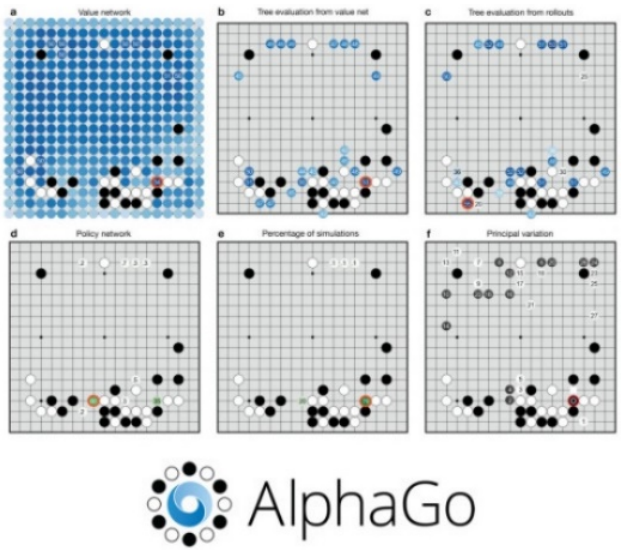
CSL

Rise of the Machines



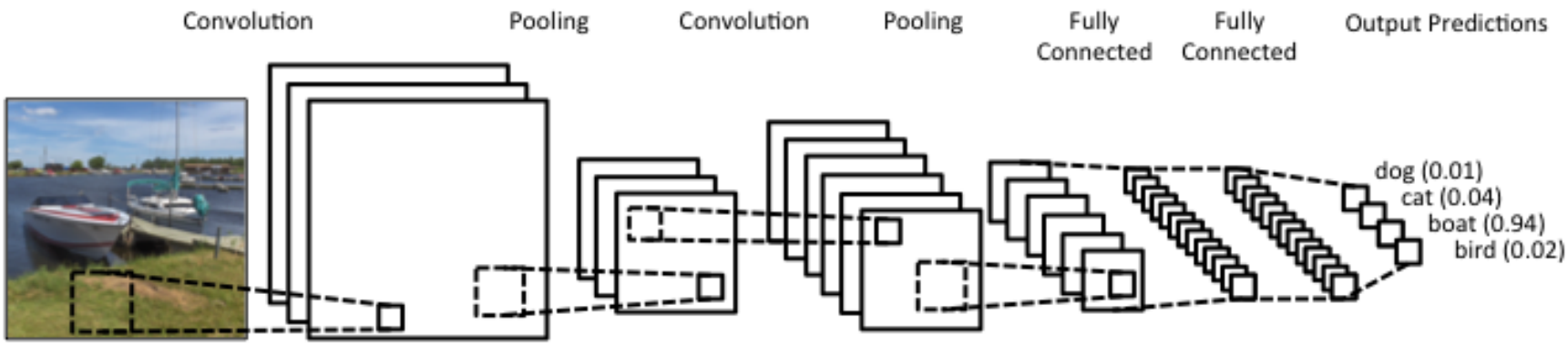






- Deep learning has revolutionized AI research and the world
  - Self-driving vehicles
  - Machine transcription/translation
  - AlphaGo
- Key algorithm is the **convolutional neural network (CNN)**

CNN Architecture



- Basics:
  - **Convolutional** (conv) layers in the front
  - **Fully connected** (dense) layers in the back
  - **Pooling** layers reduce the size of **feature maps** (fmaps)

- Enormous computational and memory requirements
  - VGG-19 Network**:** 140 million floating-point parameters and 15 billion floating-point operations per image [1]

[1] K. Simonyan and A. Zisserman. **Very Deep Convolutional Networks for Large-Scale Image Recognition**. *arXiv:1409.15568*, Apr 2015.

CNNs on FPGA?

- **Challenges**
  - CNN weights and fmaps don't fit in on-chip memory
  - Difficult for FPGAs to compete on floating-point throughput
  - Deep learning frameworks for CPU/GPU (e.g. Theano, Caffe, TensorFlow)

- **Opportunities**
  - **Energy efficiency** → deep learning on embedded platforms
  - **Networked FPGA cloud** → overcome performance limitations through scale (MSR Catapult)
  - **Hardware CNN optimizations**
    - Data reordering and tiling (Zhang FPGA'15)
    - Dynamic fixed-point quantization (Qui FPGA'16)
    - Sparse model compression (Han ISCA'16, FPGA'17)

Very Low Precision CNNs

- **ML Research Papers:**

	– BinaryConnect	[NIPS]	Dec 2015
This Model	– BNN	[arXiv]	Mar 2016
	– Ternary-Net	[arXiv]	May 2016
	– XNOR-Net	[ECCV]	Oct 2016
	– Local Binary CNNs	[arXiv]	Aug 2016

Near state-of-the-art on MNIST, CIFAR-10 and SVHN

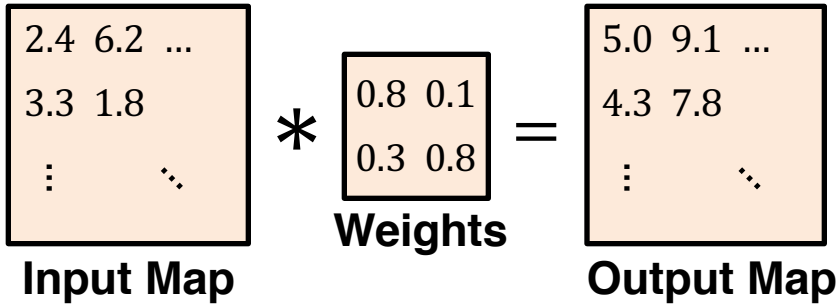
Within 3% of state-of-the-art on ImageNet

- **Our Paper:**
  - FPGA implementation of BNN[2] for CIFAR-10 inference
  - New model optimizations and hardware structures
  - Open-source HLS code available

[2] M. Courbariaux et al. **Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1**. *arXiv:1602.02830*, Feb 2016.

Binarized Neural Networks (BNN)

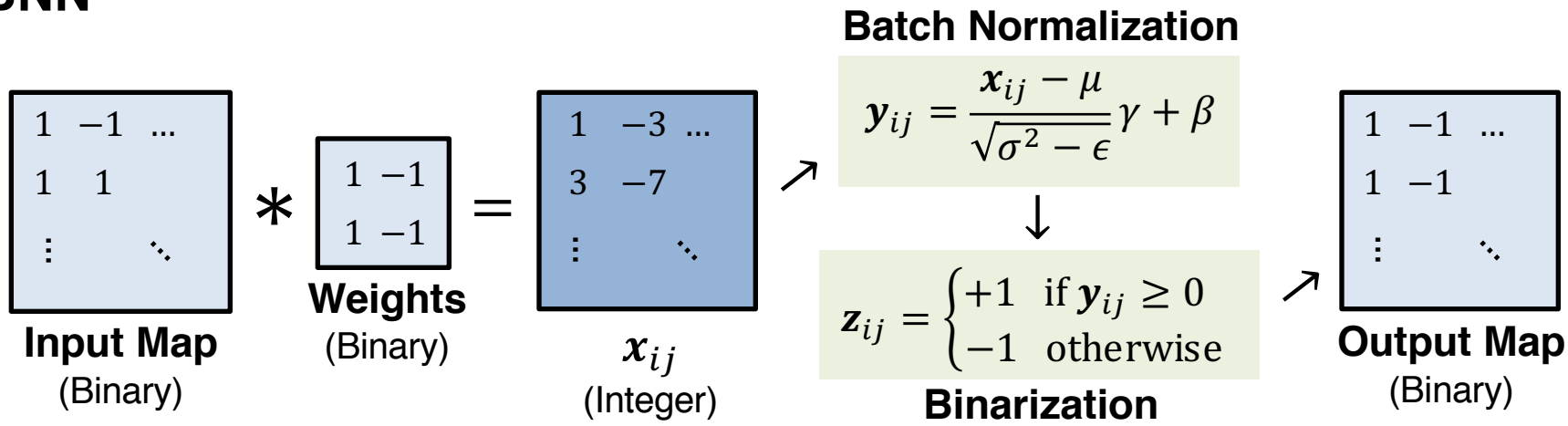
CNN



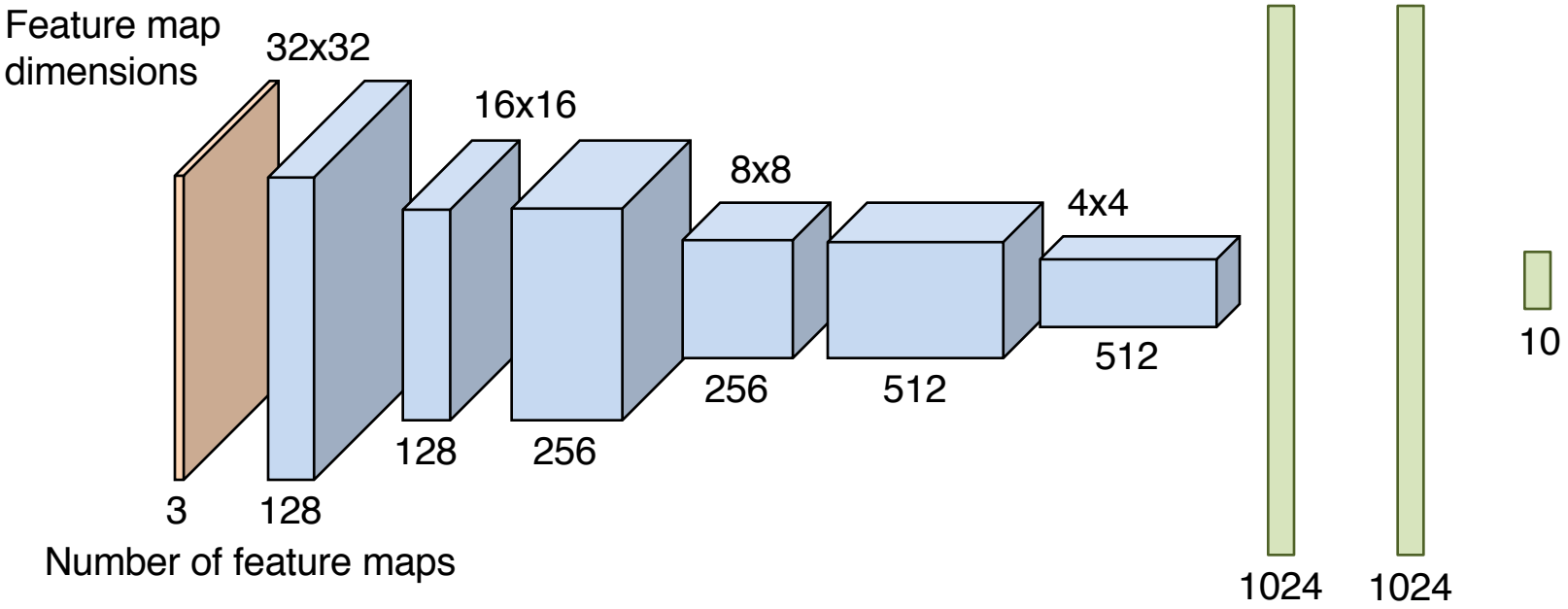
Key Differences

1. Inputs are binarized (–1 or +1)
2. Weights are binarized (–1 or +1)
3. Results are binarized after **batch normalization**

BNN



BNN CIFAR-10 Architecture [2]



- 6 conv layers, 3 dense layers, 3 max pooling layers
- All conv filters are 3x3
- First conv layer takes in floating-point input
- **13.4 Mbits total model size** (after hardware optimizations)

[2] M. Courbariaux et al. **Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1**. *arXiv:1602.02830*, Feb 2016.

Advantages of BNN

1. Floating point ops replaced with binary logic ops

b_1	b_2	$b_1 \times b_2$
+1	+1	+1
+1	−1	−1
−1	+1	−1
−1	−1	+1

b_1	b_2	b_1 XOR b_2
0	0	0
0	1	1
1	0	1
1	1	0

- Encode {+1,–1} as {0,1} → multiplies become XORs
- Conv/dense layers do dot products → XOR and popcount
- Operations can map to LUT fabric as opposed to DSPs

2. Binarized weights may reduce total model size

- Fewer bits per weight may be offset by having more weights

BNN vs CNN Parameter Efficiency

Architecture	Depth	Param Bits (Float)	Param Bits (Fixed-Point)	Error Rate (%)
ResNet [3] (CIFAR-10)	164	51.9M	13.0M*	11.26
BNN [2]	9	-	13.4M	11.40

* Assuming each float param can be quantized to 8-bit fixed-point

- **Comparison:**
  - Conservative assumption: ResNet can use 8-bit weights
  - BNN is based on VGG (less advanced architecture)
  - BNN seems to hold promise!

[2] M. Courbariaux et al. **Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1**. *arXiv:1602.02830*, Feb 2016.
[3] K. He, X. Zhang, S. Ren, and J. Sun. **Identity Mappings in Deep Residual Networks.** *ECCV 2016.*

BNN Hardware Optimizations

Optimizations

1. Quantized the input image and batch norm parameters
2. Removed additive biases (no effect on accuracy)
3. Simplified batch norm and pooling computation

Accuracy Impact

BNN Model	Test Error
Claimed in paper [2]	11.40%
Python out-of-the-box [2]	11.58%
C++ optimized model	**11.19%**
Accelerator	**11.19%**

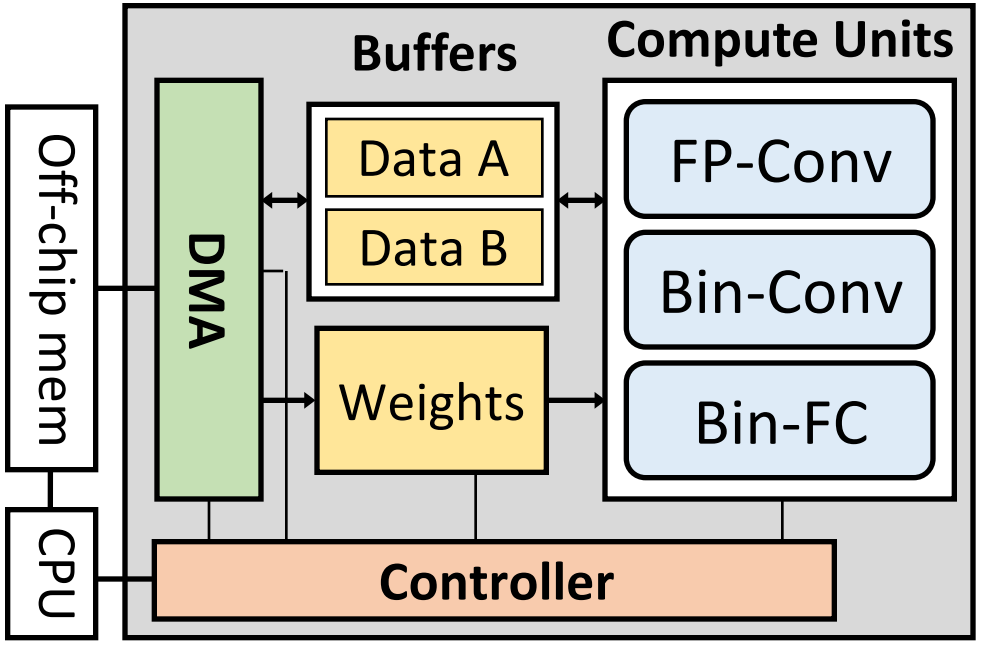
[2] M. Courbariaux et al. **Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1**. *arXiv:1602.02830*, Feb 2016.

Accelerator Design Goals

- **Target low-power embedded FPSoC**
  - Design must be resource efficient to fit the FPGA
  - Leverage resource-sharing across layers (execute layers sequentially on a single module)
- **Store all feature maps on-chip**
  - Binarization makes feature maps much smaller
- **Use Xilinx SDSoC to synthesize RTL from C++ source**

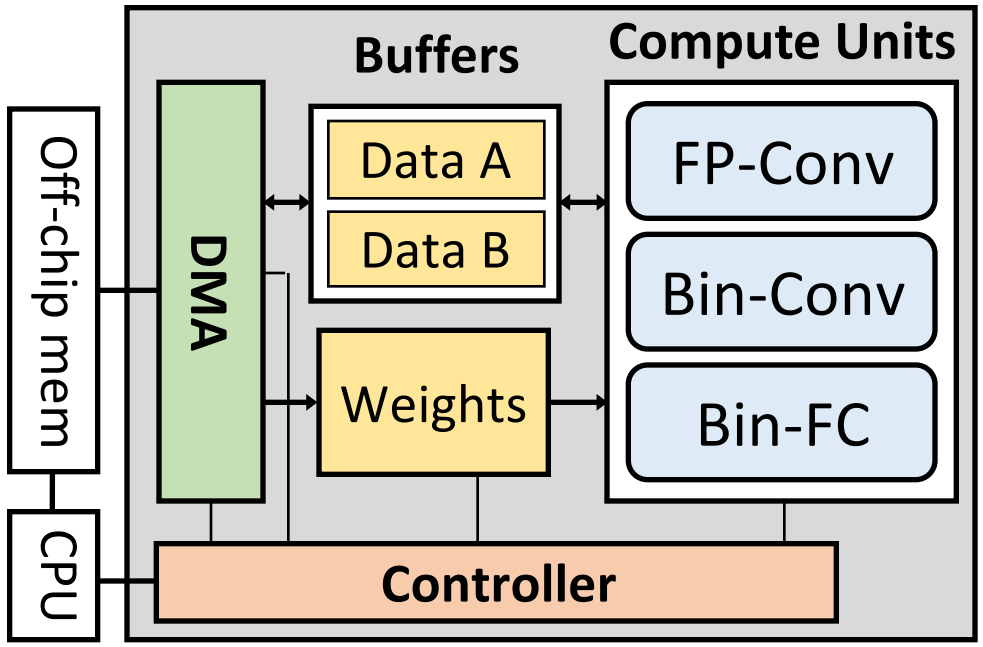
Accelerator Architecture

- **Data buffers (A and B)**
  - Stores feature maps
  - Alternately read from one and write to the other
- **Weight buffer**
  - Store weights and batch norm parameters
  - Reads from off-chip memory

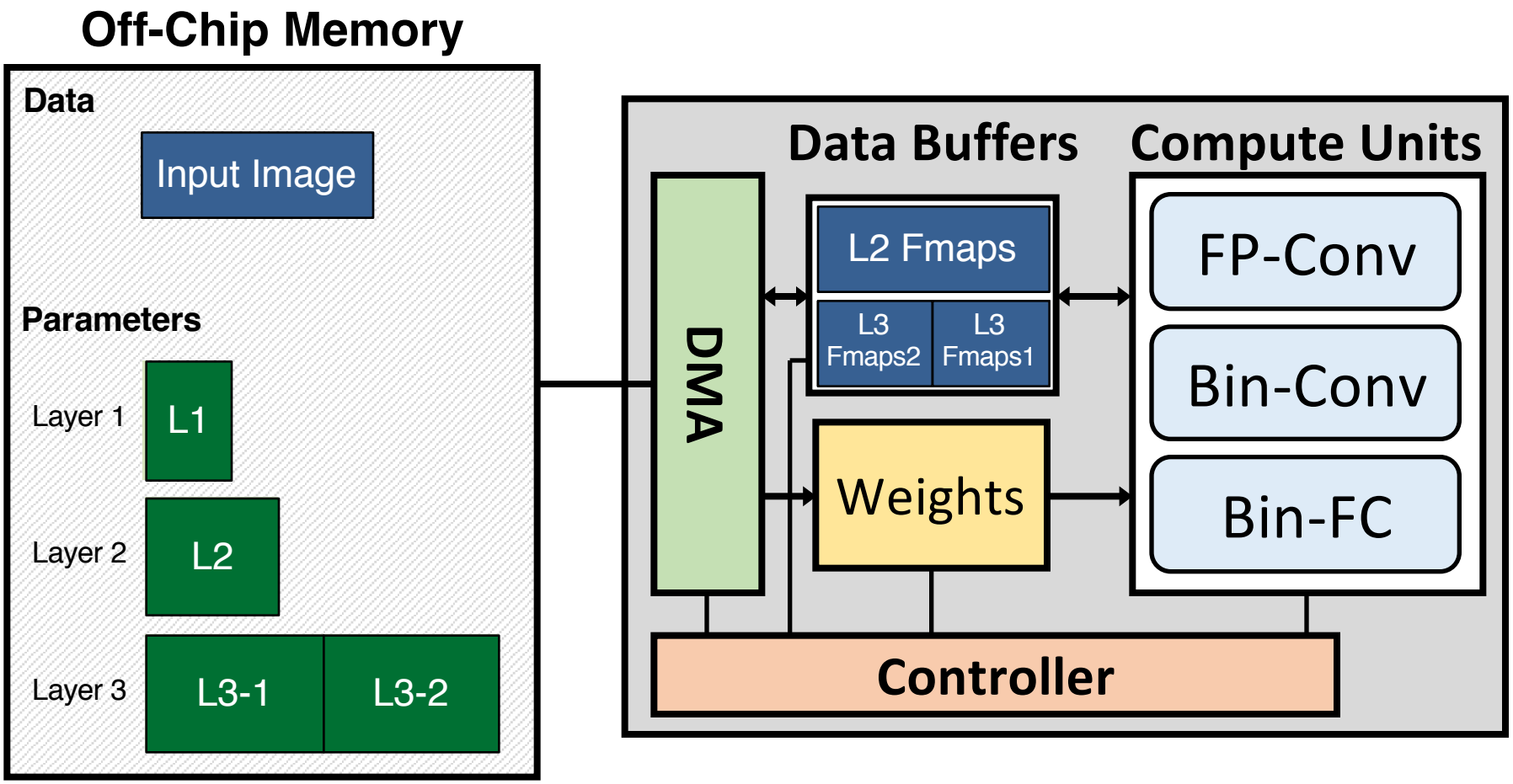


Accelerator Architecture

- **3 compute units**
  - FP-conv → input conv
  - Bin-conv → binary conv
  - Bin-FC → binary dense
- **DMA and controller**
  - Automatically generated by SDSoC



Accelerator Execution Example

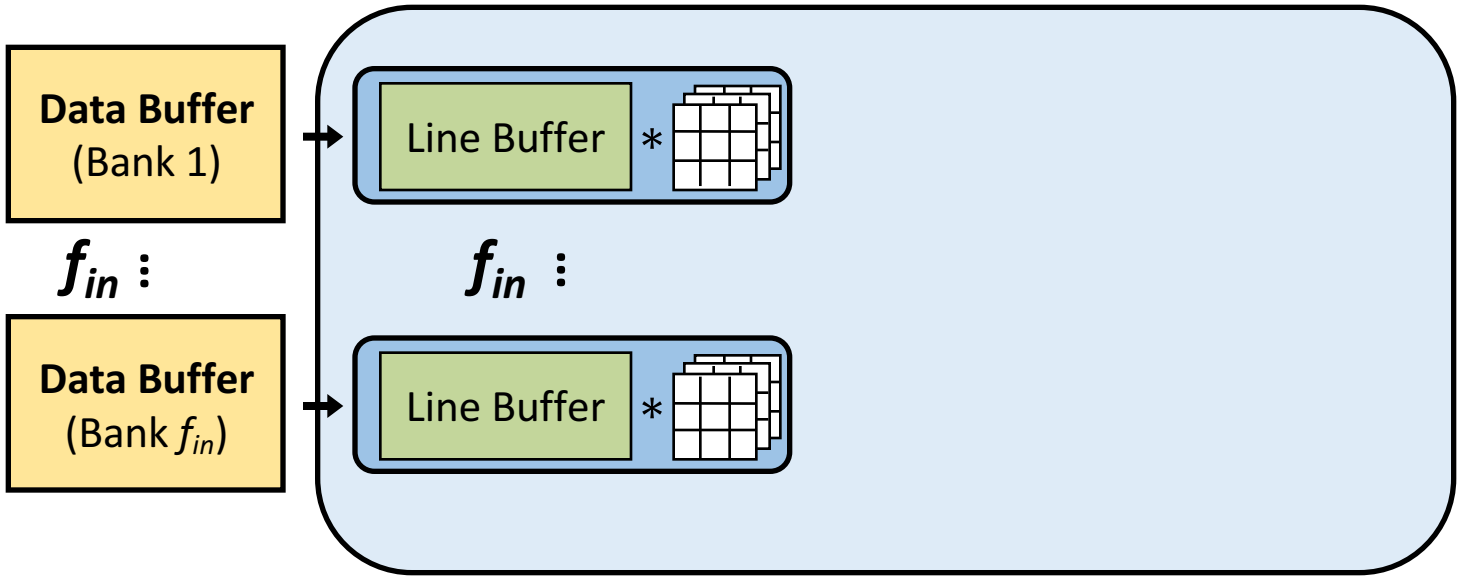


Bin-Conv Unit

- Responsible for the binary conv layers, which take up most of the runtime

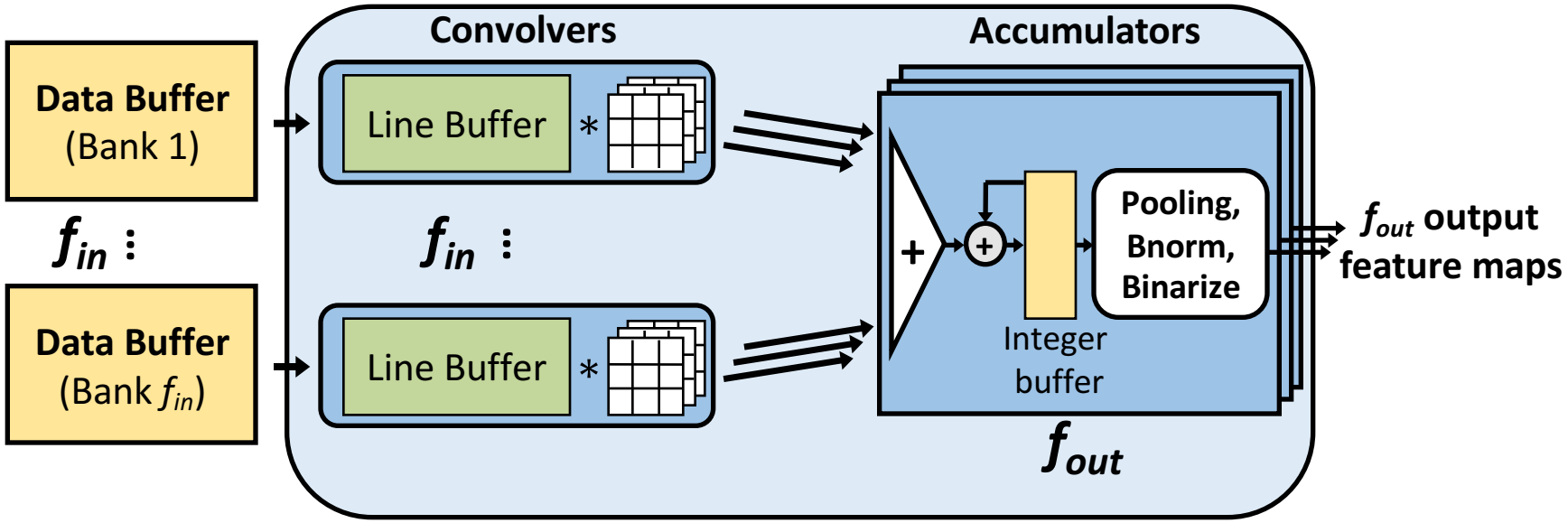
- **Design Goals:**
  - Configurable for different feature map widths and different numbers of feature maps in a layer
  - Scalable performance
  - Exploits parallelism across input/output feature maps and within the pixels of each feature map

Bin-Conv Unit – Input Parallelization



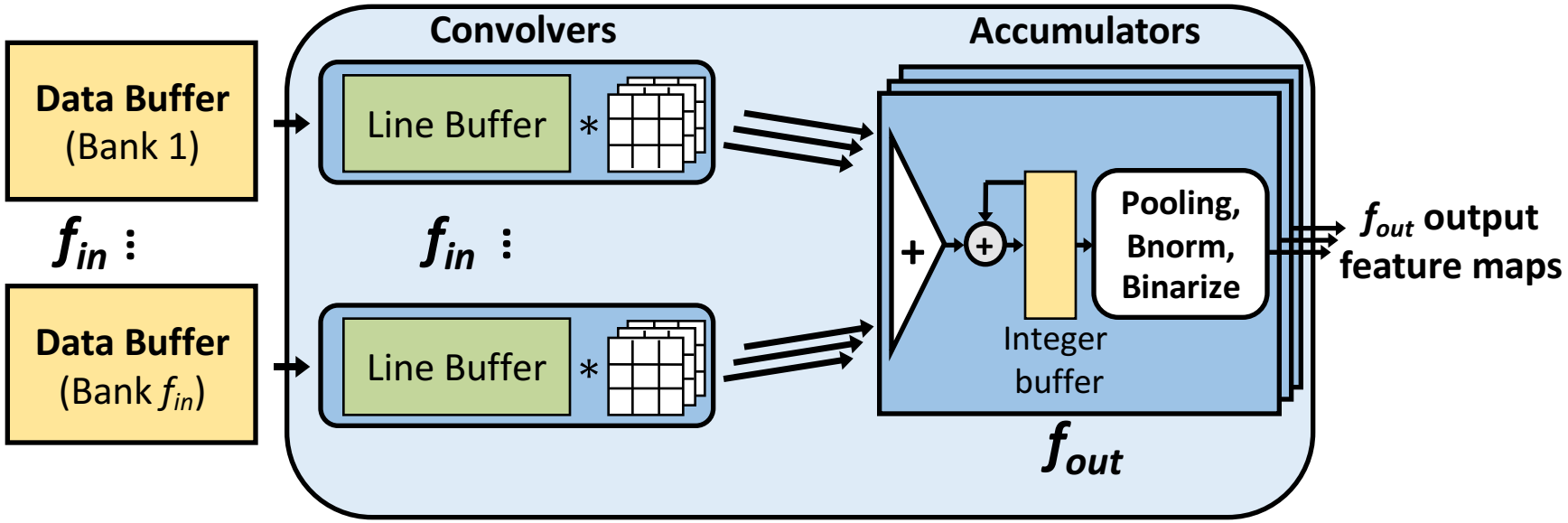
- **f_{in} is the input parallelization factor**
  - Read f_{in} words from the data buffer each cycle
  - The words go to f_{in} **convolvers**
  - Each convolver contains a line buffer and convolution logic

Bin-Conv Unit – Output Parallelization



- **f_{out} is the output parallelization factor**
  - The convolvers generate partial conv sums for f_{out} new maps
  - Accumulate f_{out} output feature maps in parallel
  - Completed feature maps are pushed through pooling, batch norm, and binarization logic

Bin-Conv Unit – Pixel Parallelization



- **The word size is the pixel parallelization factor**
  - Pixels (bits) in a word are processed in parallel
  - **Variable-width line buffer** can be configured for different feature map widths

$f_{in} - f_{out}$ tradeoff

- f_{in}/f_{out} control how many input/output feature maps we process in parallel
  - f_{in} increases the number of line buffers
  - f_{out} increases the number of integer feature map buffers, pooling units, and batch norm units
  - We expect increasing f_{in} to be more area efficient

- In our implementation we fix f_{out}= 1 and allow f_{in} to vary

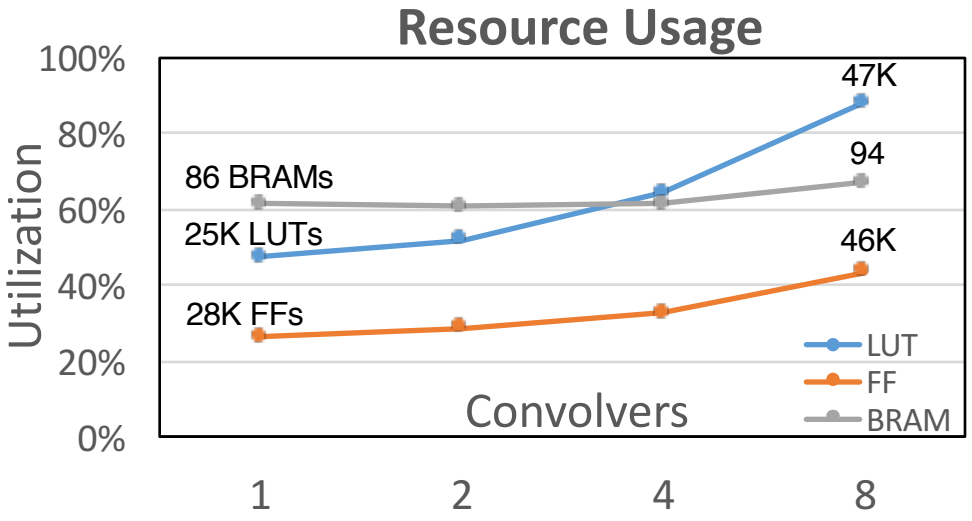
Experiment Setup

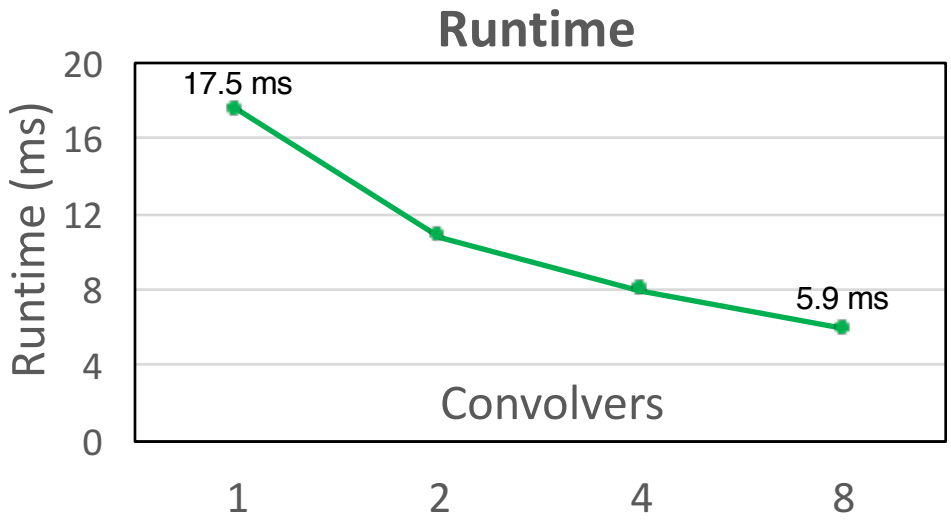
- **Target platform:** Zedboard with XC7Z020 FPGA
  - 53K LUTs, 106K FFs
  - 40 BRAMs, 220 DSPs

- **Measurement**
  - Power: Physical meter
  - Resource: Post-route report

- **Platform Comparisons:**
  - **CPU:** Intel Xeon 8-core 2.6GHz
  - **GPU:** NVIDIA Tesla K40
  - **mGPU:** NVIDIA Jetson TK1 embedded board

Resource Usage and Scalability

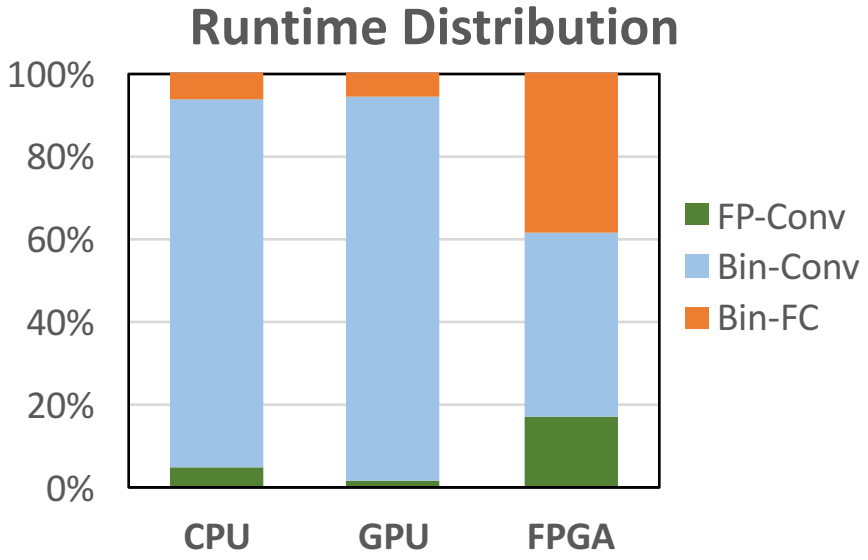




- **Resource:**
  - LUT and FF usage scaled with # of convolvers
  - BRAM and DSP were mostly the same
- **Runtime:**
  - Scales with # of convolvers, some overhead
- **Final Design:**
  - 88% LUT utilization
  - 5.9ms per image
  - 143MHz

Performance Comparison

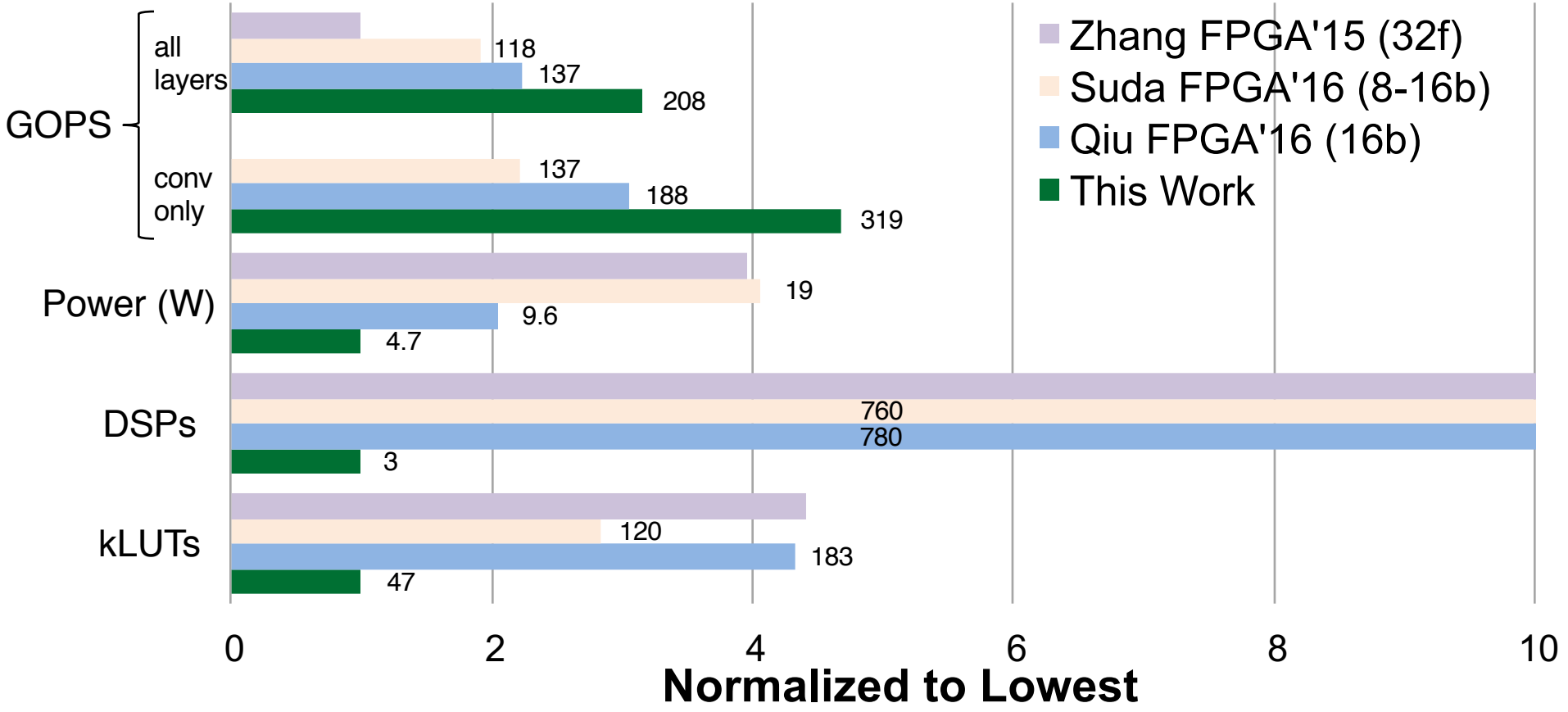
	mGPU	CPU	GPU	FPGA
Runtime per Image (ms)	90	14.8	0.73	5.94
Speedup	1x	6x	120x	15x
Power (W)	3.6	95	235	4.7
Energy Efficiency	3.1	0.71	5.8	36



- **vs. mGPU and CPU**: significant improvement in performance and energy efficiency
- **vs. GPU**: 8x slower but 6x more energy efficient

- Binary conv layers see the most speedup on FPGA
  - First (floating point) conv layer is not heavily parallelized
  - Dense layers are bound by memory throughput

Comparison with CNNs on FPGA



- Comparing CNN and BNN GOPS is not apples-to-apples…
  - But it shows the binary ops we can squeeze out of an FPGA board
  - Our device is considerably smaller than the other works' here

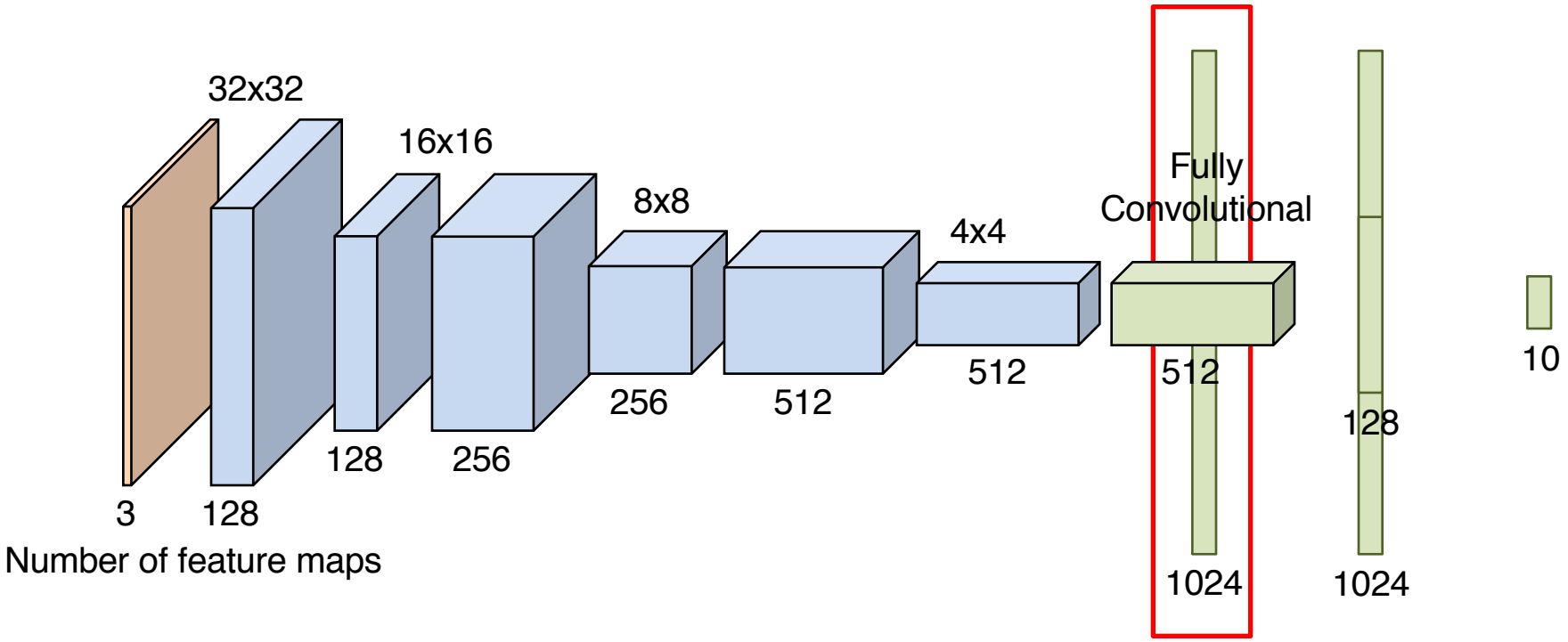
Conclusion

- **Takeaways:**
  - Low precision CNNs on FPGA show great promise
  - There is room for algorithmic improvement in binarized CNNs

- **Code:** https://github.com/cornell-zhang/bnn-fpga
  - '**master**' branch is the debug build (larger area)
  - '**optimized**' branch is the paper build

- **Further Improvements:**
  - '**conv1x1**' branch is a modified BNN with 60% model size reduction and negligible accuracy loss

Improved BNN Architecture



- **13.4 Mbits** total model size (after hardware optimizations)
- **8.0 Mbits** in the first dense layer alone!
- Convert first dense layer into **fully convolutional layer**
  - Shrink second dense layer
- **Model Size:** 13.4M → 5.6M
- **Test Error:** 11.19% → 11.69%