Heterogeneous Compute Architectures For Deep Learning In The Cloud

Ken O'Brien, **Nicholas Fraser**, Michaela Blott

27th March 2019

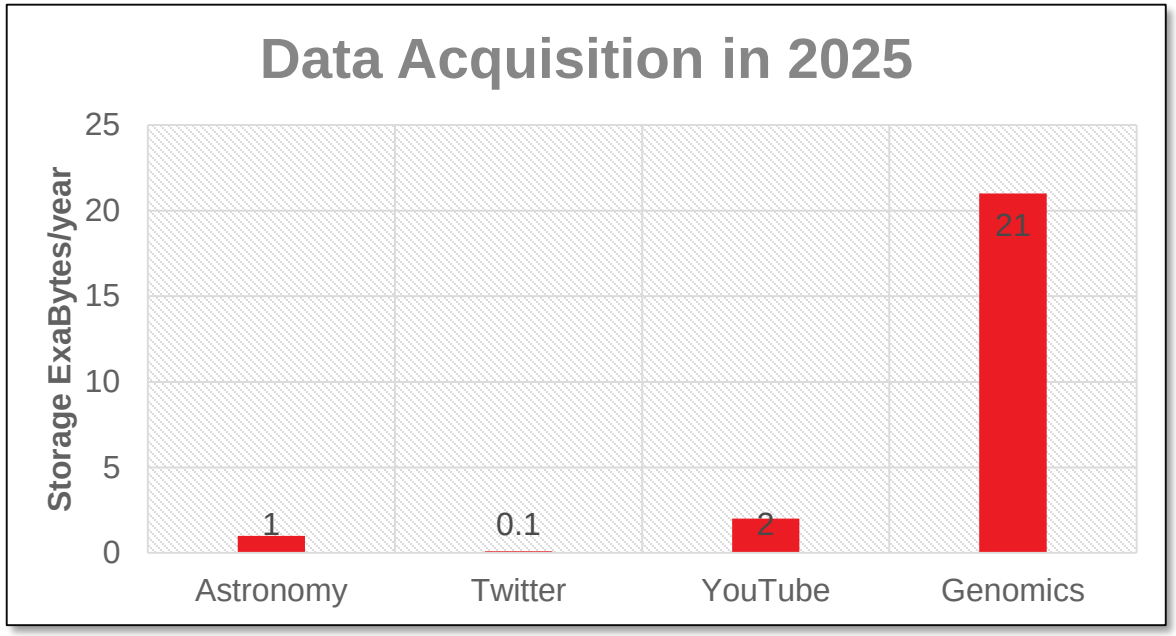
XILINX

Outline

- Why FPGAs?
- Deep Learning: Challenges & Solutions
- FINN
- FPGAs to ACAPs

Mega-Trend:
Explosion of Data

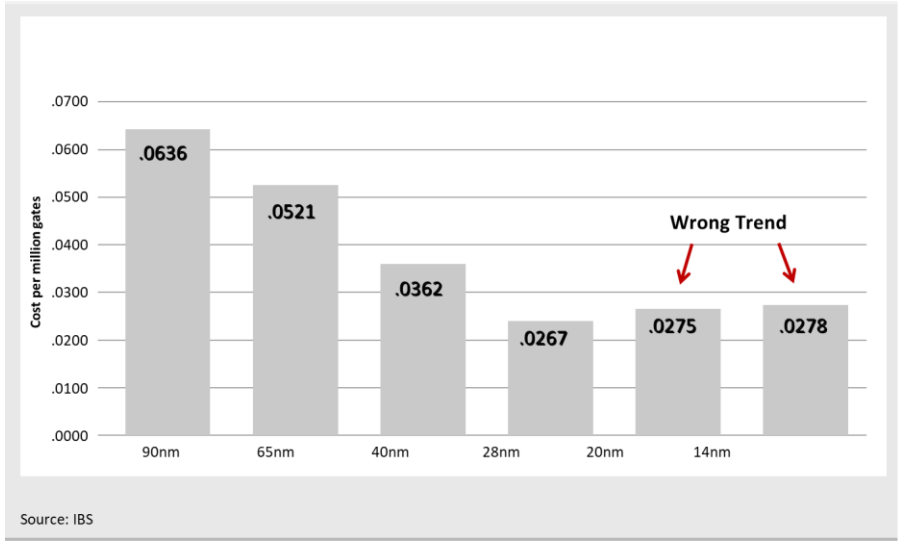
- Astronomically growing amounts of data
  - More sensors
  - More users
  - More use cases: Genomics (DNA) **“Genomical”**



We need significantly more compute resources to process and extract patterns / insights from this data!

Stephens, Zachary D., et al.
"Big data: astronomical or genomical?."

Technology:
End of Moore's Law & Dennard Scaling

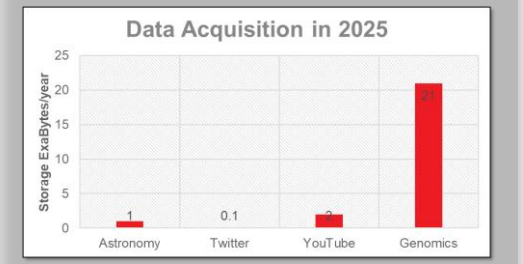


Economics become questionable

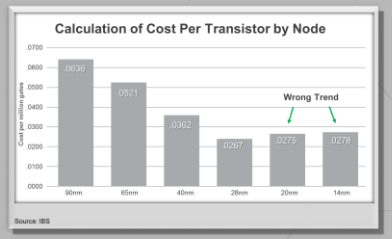
Power dissipation becomes problematic

Era of Heterogeneous Compute using Accelerators

Trends



Technology

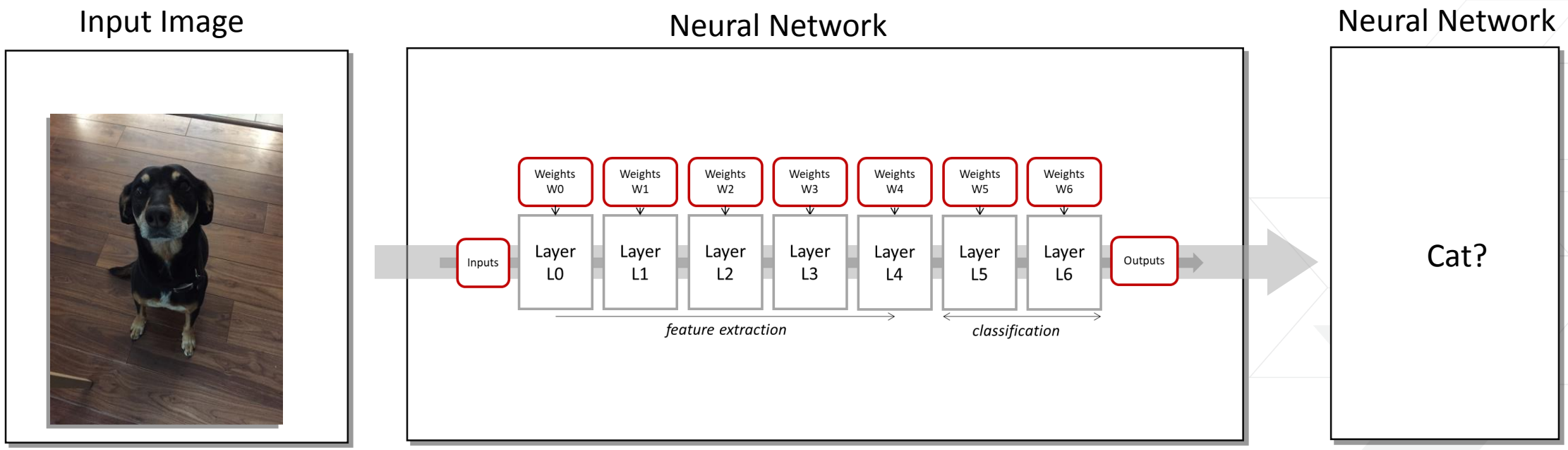


- **Diversification of increasingly heterogenous devices and system**
  - Moving away from standard van Neumann architectures
- **True Architectural innovation & Unconventional Computing Systems**

Deep Learning
- customized precision arithmetic

XILINX

What's the Challenge?
Example: Convolutional Neural Networks
Forward Pass (Inference)



For ResNet50:

70 Layers

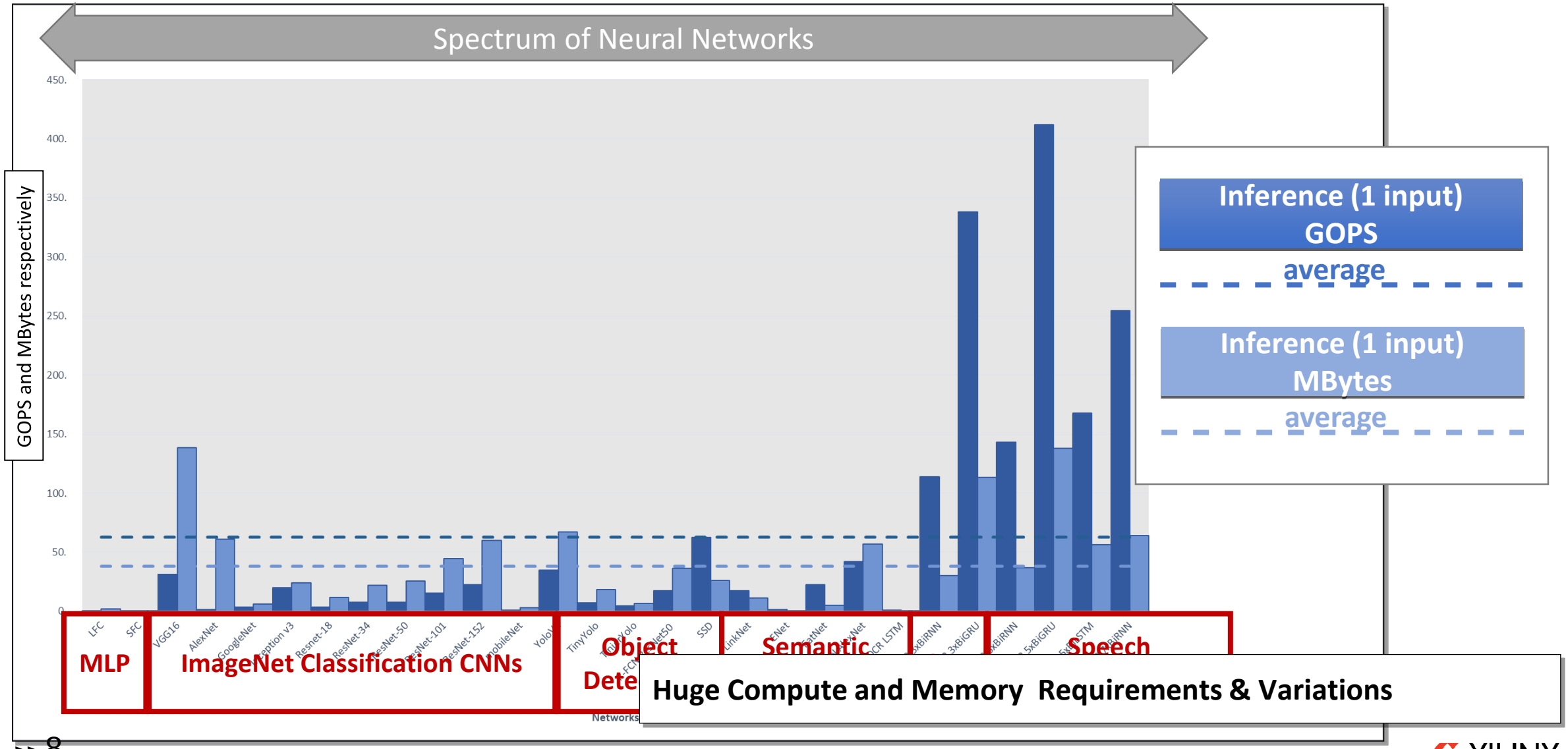
7.7 Billion operations

25.5 millions of weights

Basic arithmetic, incredible parallel but Huge Compute and Memory Requirements

Compute and Memory for Inference

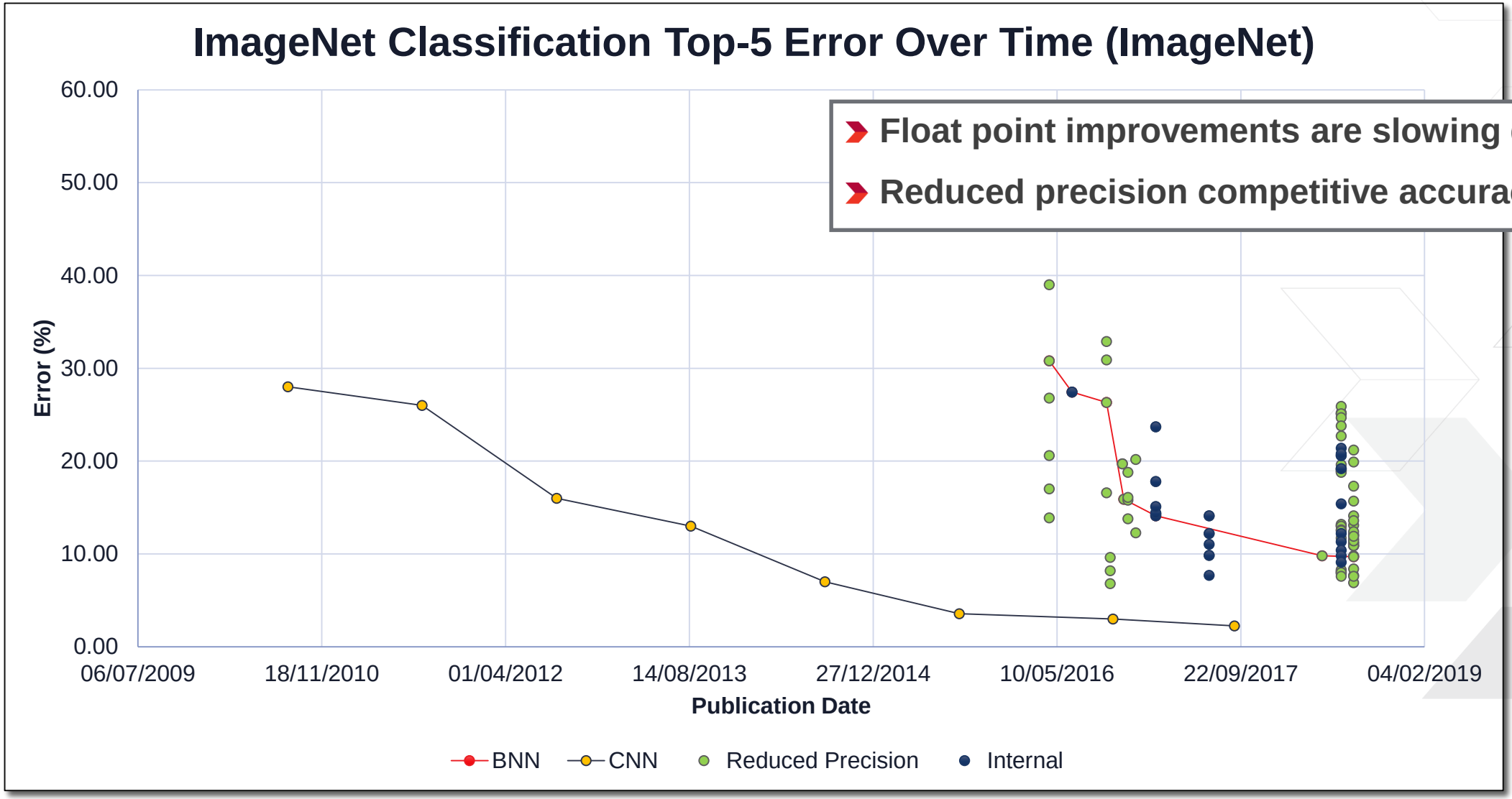
*architecture independent
**1 image forward
*** batch = 1
**** int8



XILINX

Floating Point to Reduced Precision Neural Networks

Deliver Competitive Accuracy



XILINX.

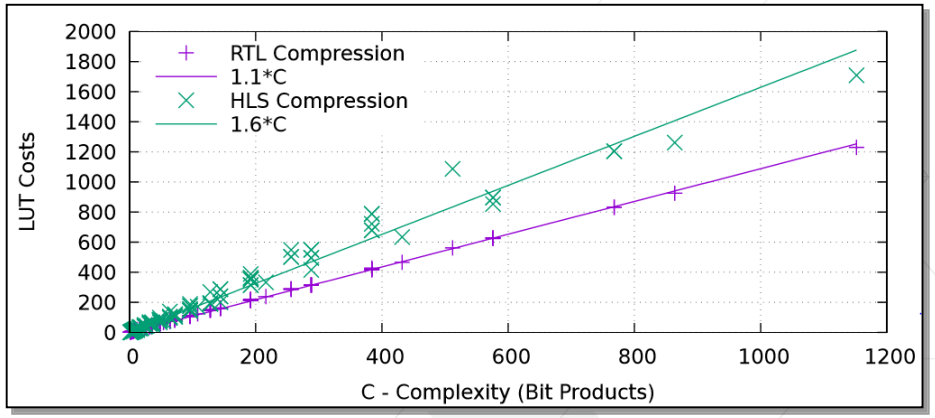
Reducing Precision

Scales Performance & Reduces Memory

- **Reducing precision shrinks LUT cost**
  - Instantiate **100x** more compute within the same fabric

- **Potential to reduce memory footprint**
  - NN model can stay on-chip => no memory bottlenecks

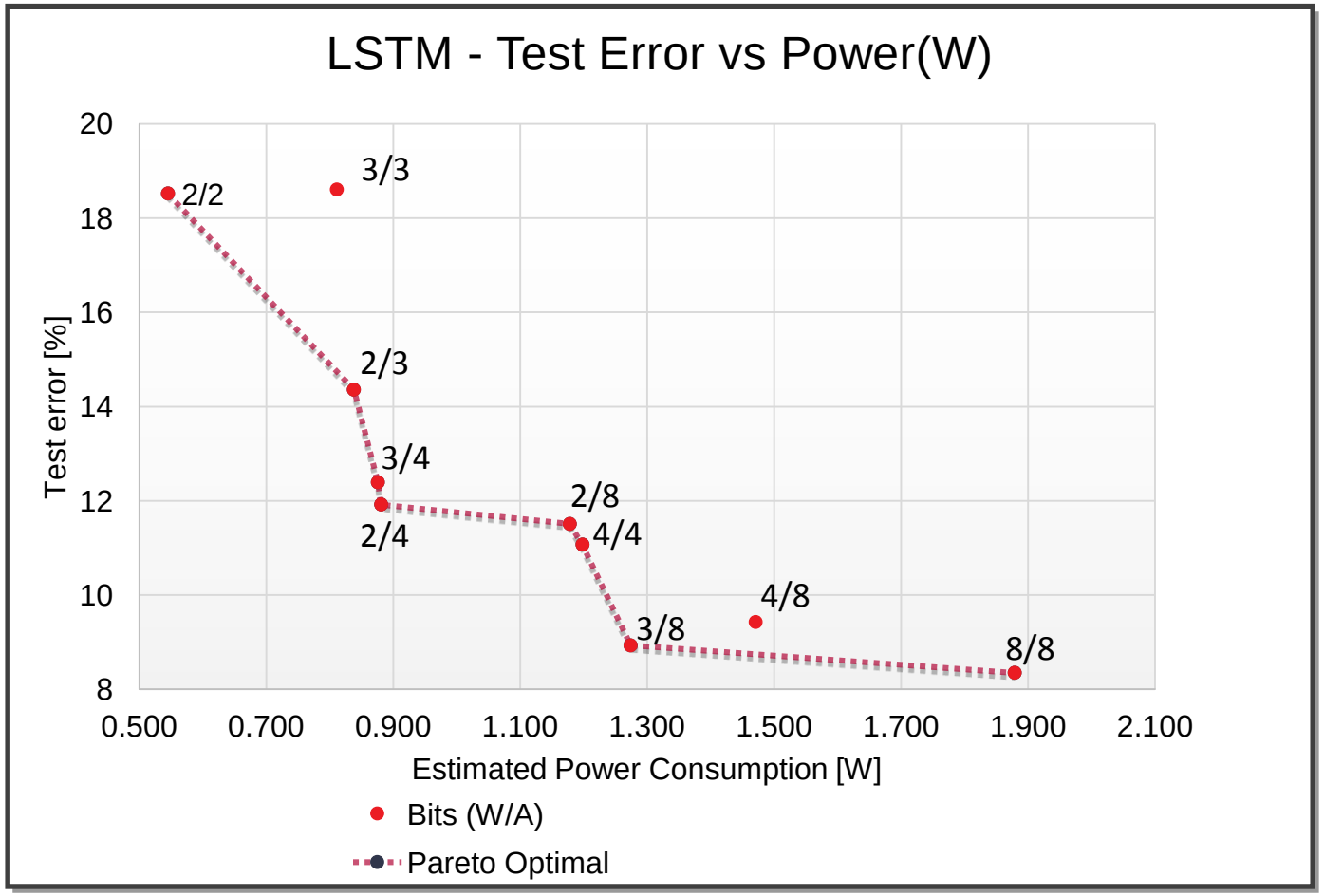
Precision	Modelsize [MB] (ResNet50)
1b	3.2
8b	25.5
32b	102.5



C= size of accumulator *
size of weight *
size of activation

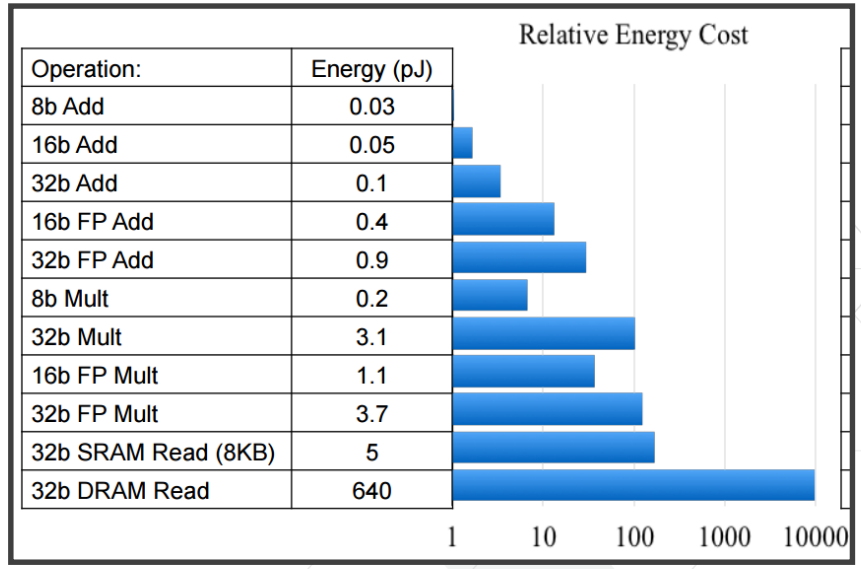
Reducing Precision Inherently Saves Power

FPGA:



Target Device ZU7EV ● Ambient temperature: 25 °C ● 12.5% of toggle rate ● 0.5 of Static Probability ● Power reported for PL accelerated block only

ASIC:



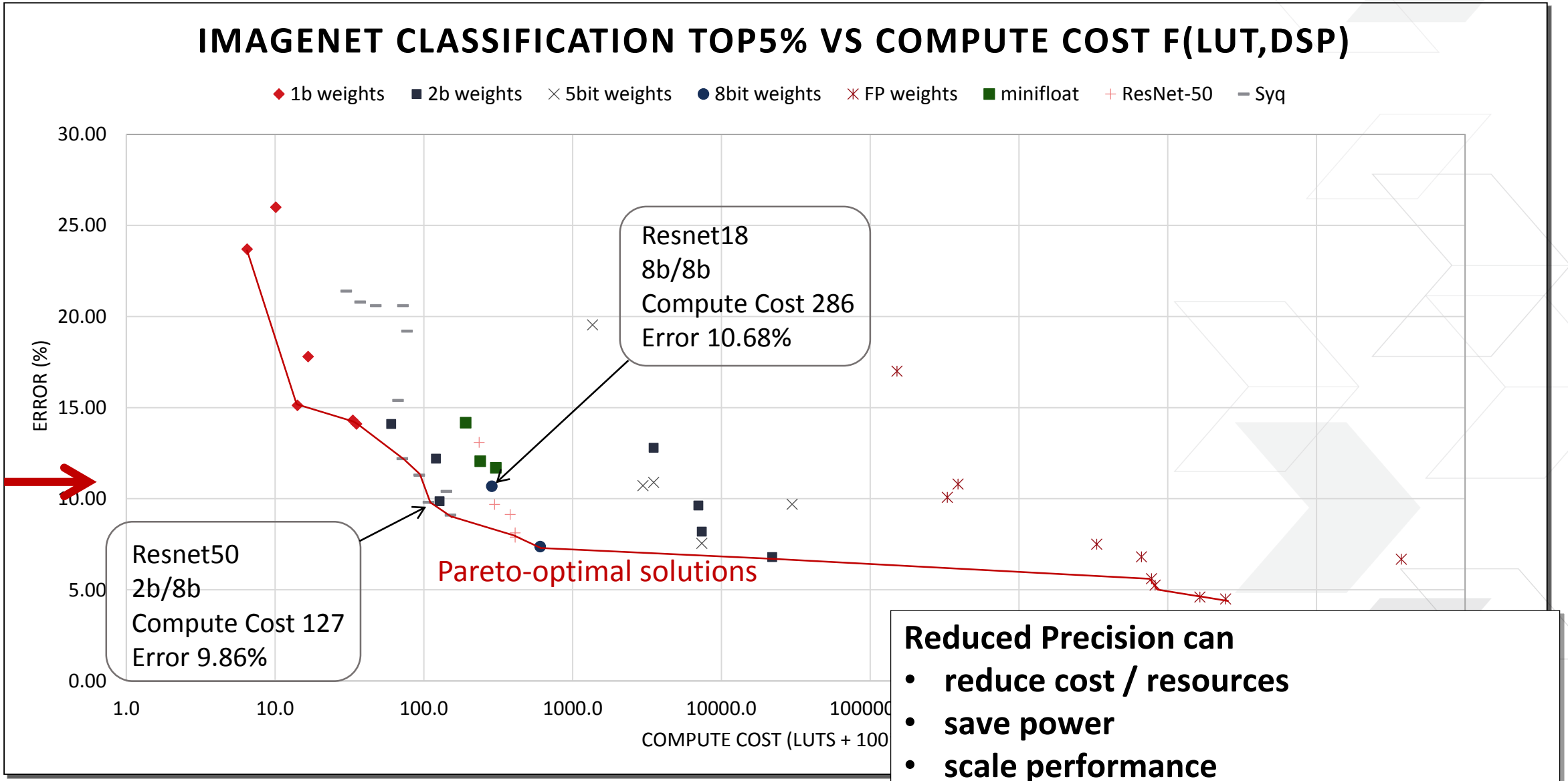
Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

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

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Rybalkin, V., Pappalardo, A., Ghaffar, M.M., Gambardella, G., Wehn, N. and Blott, M. "FINN-L: Library Extensions and Design Trade-off Analysis for Variable Precision LSTM Networks on FPGAs."

Design Space Trade-Offs



Reduced Precision can

- **reduce cost / resources**
- **save power**
- **scale performance**

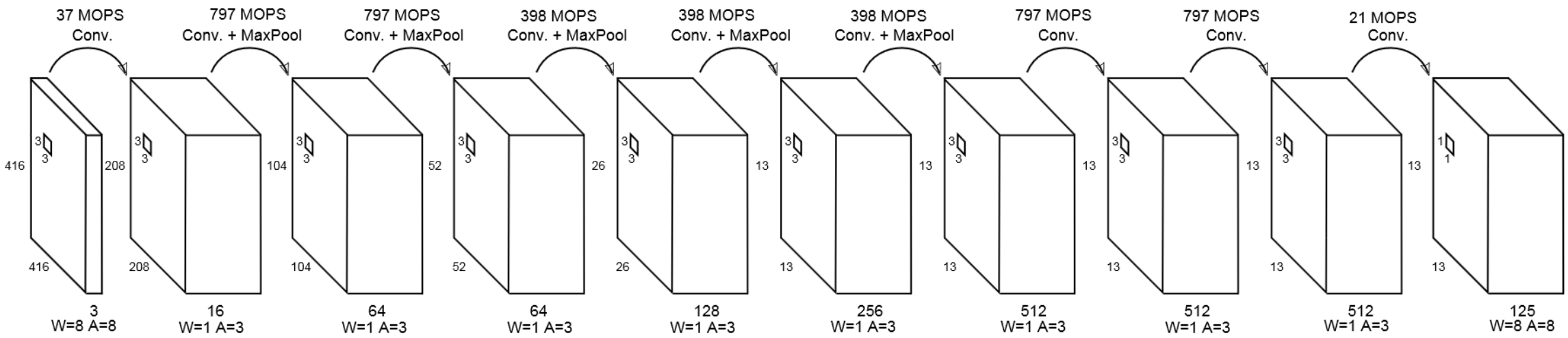
XILINX

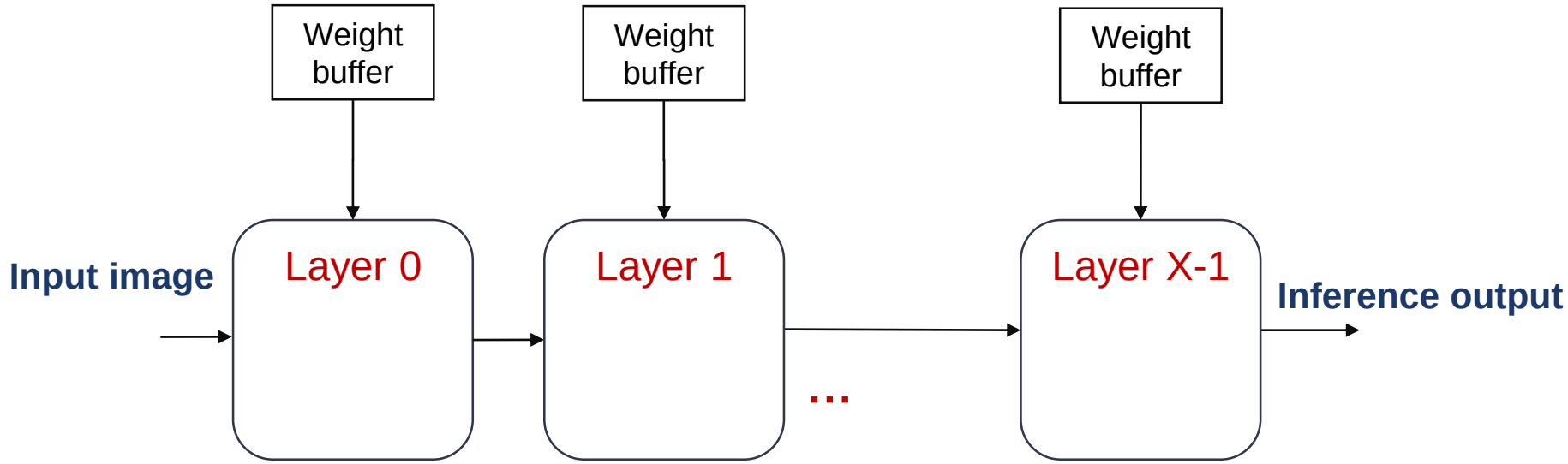
Scaling with FINN

FINN –Tool for Exploration of NNs of FPGAs

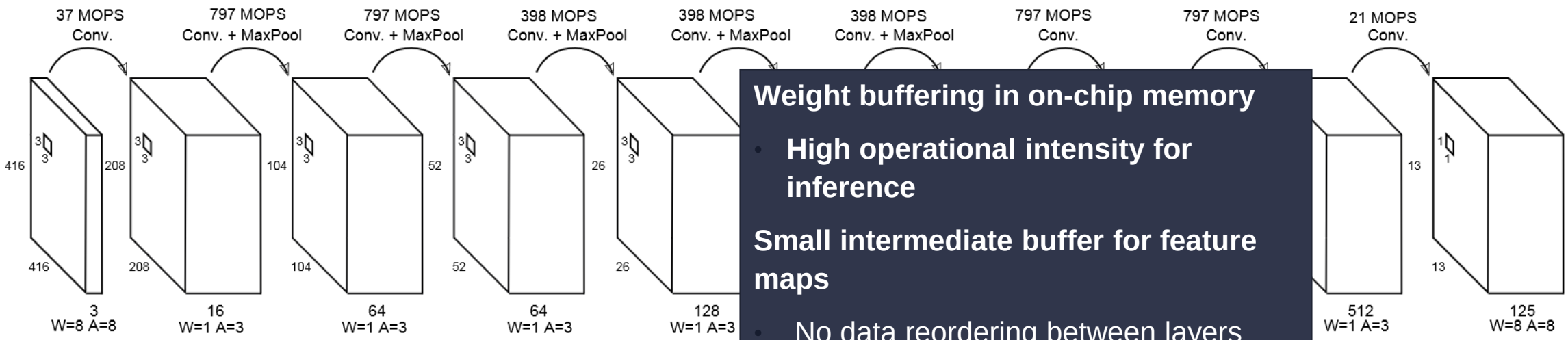
- **Design Flow Tool for Quantized Neural Networks**
  - Rapid access to network structure and compute/memory footprint statistics
  - Performance prediction for target device
  - Automatic architecture scaling and generation for target device
- **Multi-stage tool-flow**
  - Frontend
  - Design Space Exploration
  - Backend
- **Binary Network Release Available**
  - https://github.com/Xilinx/FINN

HW Architecture – Dataflow





HW Architecture – Dataflow

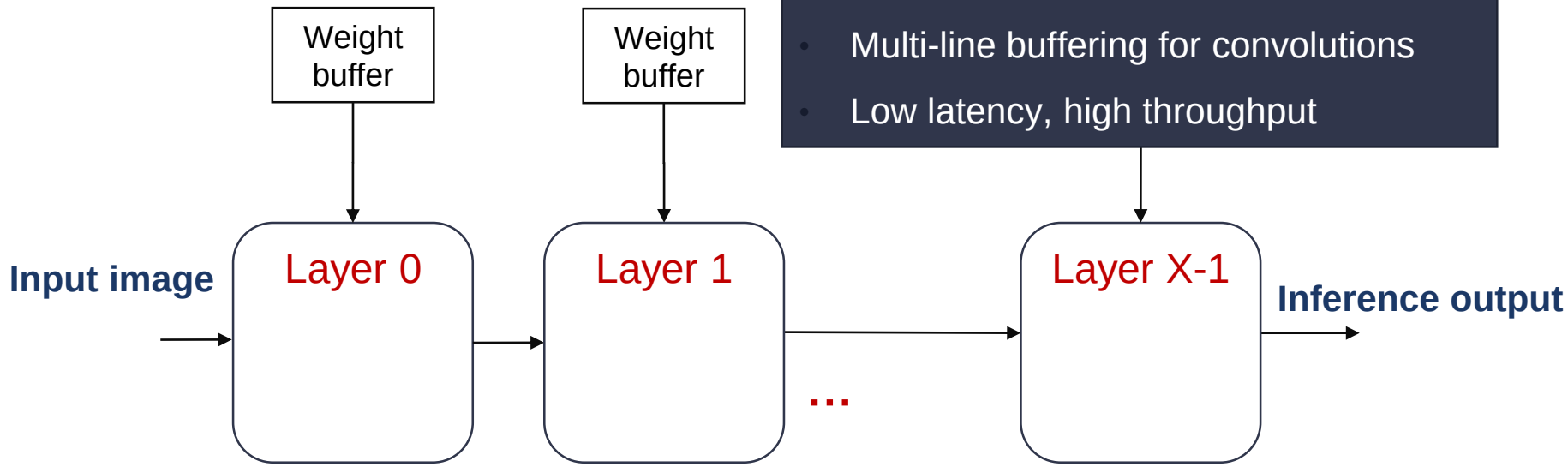


Weight buffering in on-chip memory

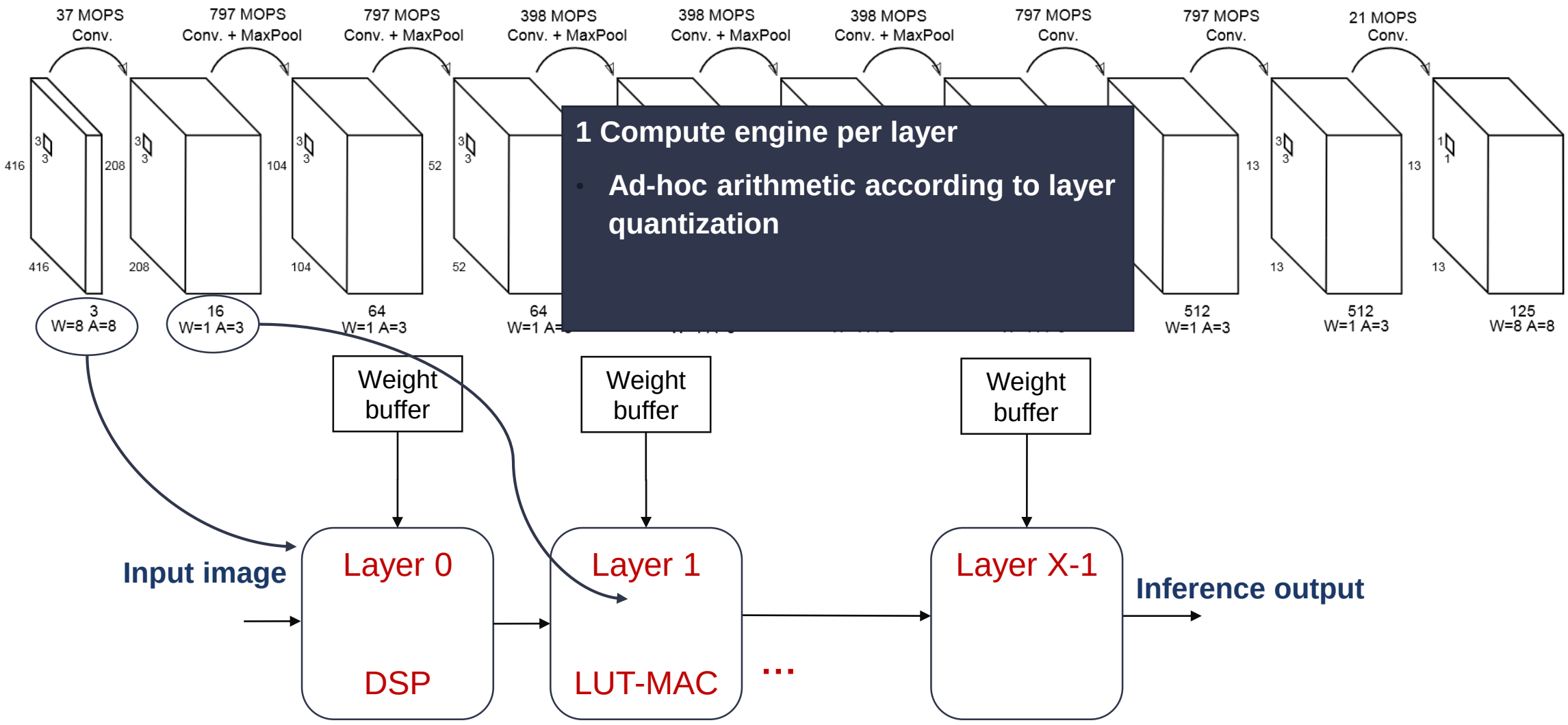
- **High operational intensity for inference**

Small intermediate buffer for feature maps

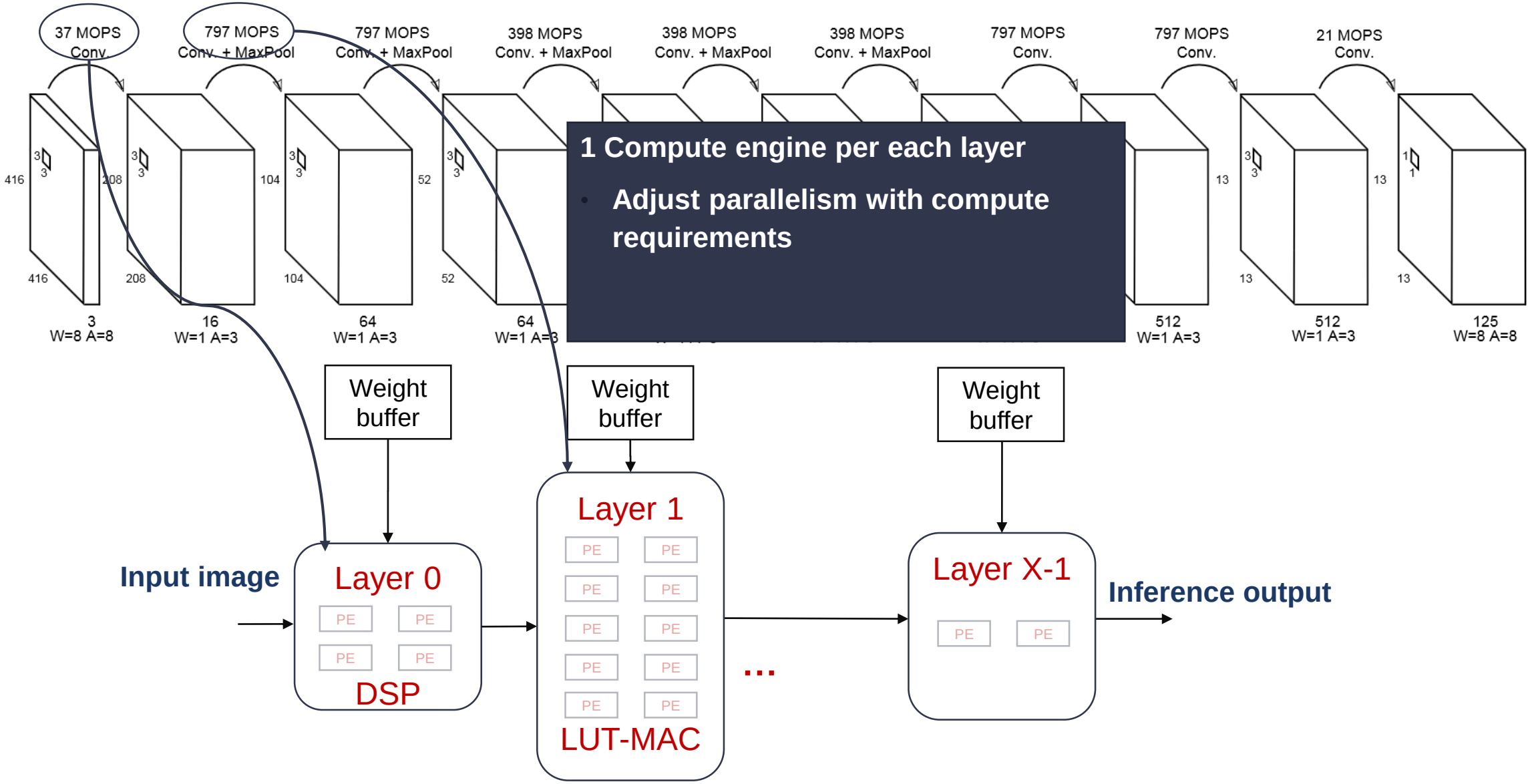
- No data reordering between layers
- Multi-line buffering for convolutions
- Low latency, high throughput



HW Architecture – Dataflow



HW Architecture – Dataflow



Frontend Stage – Import and Network Statistics

Neural Network Description (Prototxt)

```
}
layer {
  name: "qt_inp"
  type: "Quant"
  bottom: "bn_inp"
  top: "qt_inp"
  quant_param {
    forward_func: "hwgq"
    backward_func: "relu"
    centers: 0.538 centers:
    clip_thr: 1.614
  }
}
layer {
  name: "ip1"
  type: "BinaryInnerProduct"
  bottom: "qt_inp"
  top: "ip1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  inner_product_param {
    num_output: 256
```

Per layer operations

ops/layer
171320832
892296000
595745280
135364608
127401984
102760448
33554432
8192000

Topology summary

out_dim	filter_dim	in_chan	out_chan	parallel	in_dim
212.0	12	3	68	1	224
54.0	5	34	90	2	54
26.0	3	180	272	1	26
12.0	3	136	192	2	12
12.0	3	192	128	2	12
5.0	1	12544	4096		
1.0	1	4096	4096		
1.0	1	4096	1000		

Design Space Exploration Stage: Balanced Dataflow

Neural Network Description

```
layer {
  name: "qt_inp"
  type: "Quant"
  bottom: "bn_inp"
  top: "qt_inp"
  quant_param {
    forward_func: "hwgq"
    backward_func: "relu"
    centers: 0.538 centers:
    clip_thr: 1.614
  }
```

Device Specification File

```
{
  "name": "XLNX:KU115",
  "type": "fpga",
  "frequency" : 200,
  "resources": {
    "LUT": 1451000,
    "DSP": 5520,
    "BRAM": 75.9,
    "URAM": 0
  }
}
```

FINN

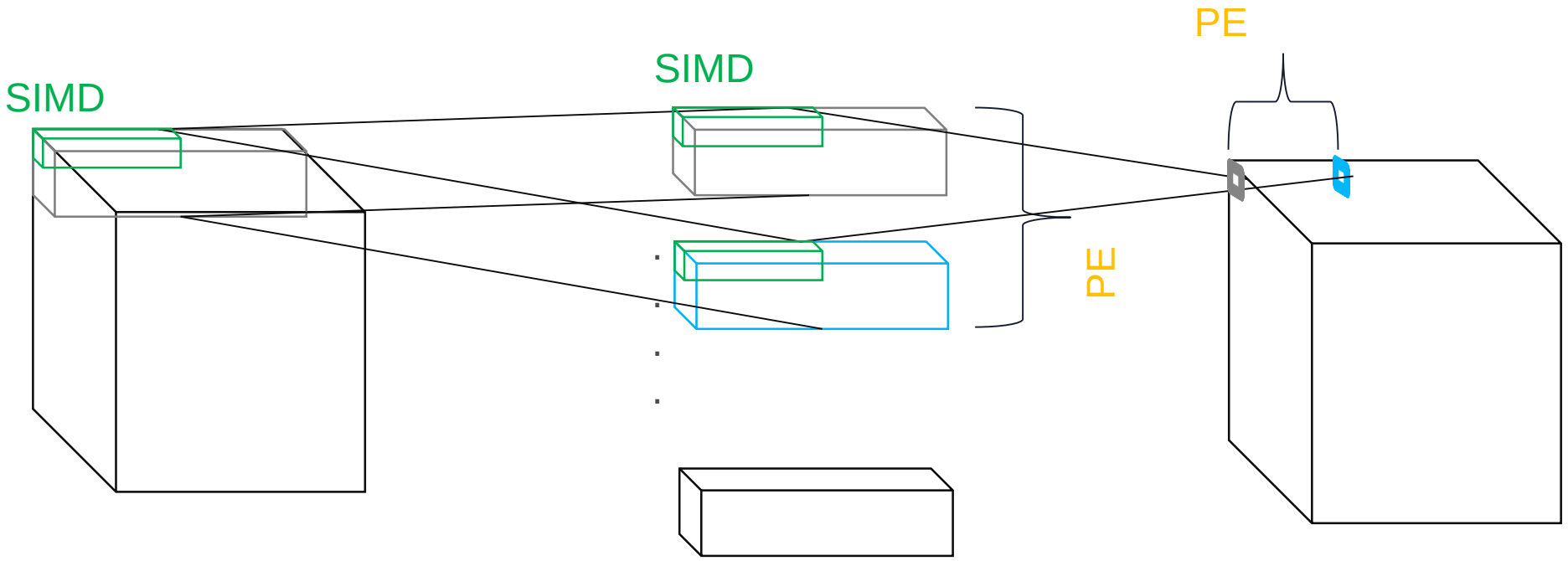
Folding Factor Calculation

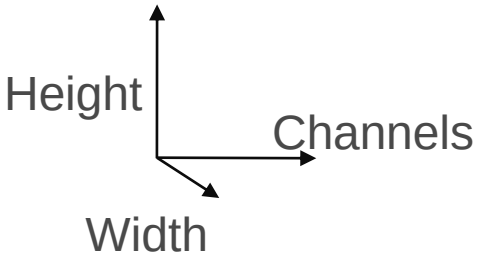
NAME	SIMD	PE	MMV
ConvolutionLayer	3	34	1
ConvolutionLayer	34	9	1
ConvolutionLayer	36	8	1
ConvolutionLayer	34	1	1
ConvolutionLayer	32	1	1
FullyConnectedLayer	32	4	1
FullyConnectedLayer	16	1	1
FullyConnectedLayer	4	1	1

Performance Prediction

```
Achieved FPS: 27400 with 15.83% LUT utilization
and 48.73% BRAM utilization
LUTS: 141004/890829 BRAM: 787/1615
```

Convolutional Layer – Folding





Design Space Exploration Stage: Balanced Dataflow

Neural Network Description

```
layer {
  name: "qt_inp"
  type: "Quant"
  bottom: "bn_inp"
  top: "qt_inp"
  quant_param {
    forward_func: "hwgq"
    backward_func: "relu"
    centers: 0.538 centers:
    clip_thr: 1.614
  }
```

Device Specification File

```
{
  "name": "XLNX:KU115",
  "type": "fpga",
  "frequency" : 200,
  "resources": {
    "LUT": 1451000,
    "DSP": 5520,
    "BRAM": 75.9,
    "URAM": 0
  }
}
```

FINN

1: Given a target FPS, what resources are required?

2: Given total resources, what FPS can be achieved?

Folding Factor Calculation

NAME	SIMD	PE	MMV
ConvolutionLayer	3	34	1
ConvolutionLayer	34	9	1
ConvolutionLayer	36	8	1
ConvolutionLayer	34	1	1
ConvolutionLayer	32	1	1
FullyConnectedLayer	32	4	1
FullyConnectedLayer	16	1	1
FullyConnectedLayer	4	1	1

Performance Prediction

```
Achieved FPS: 27400 with 15.83% LUT utilization
and 48.73% BRAM utilization
LUTS: 141004/890829 BRAM: 787/1615
```

Vivado HLS – QNN Library

```
template<
                // convolution parameters
                unsigned int ConvKernelDim,            // e.g 3 for a 3x3 conv kernel (assumed square)
                unsigned int IFMChannels,              // number of input feature maps
                unsigned int IFMDim,                   // width of input feature map (assumed square)
                unsigned int OFMChannels,              // number of output feature maps
                // unsigned int OFMDim,                // IFMDim-ConvKernelDim+1 or less
                unsigned int Stride,

                // matrix-vector unit parameters
                unsigned int SIMDWidth,                // number of SIMD lanes
                unsigned int PECount,                  // number of PEs
                unsigned int WMemCount,                // entries in each PEs weight memory
                unsigned int TMemCount,                // entries in each PEs threshold memory

                // precision parameters
                unsigned int WeightsPrecision,  // Number of bits in thresholds
                unsigned int ThresholdPrecision,// Number of bits in thresholds
                unsigned int MacPrecision,             // MAC bitwidth
                unsigned int Input_precision,                  // Input data bitwidth
                unsigned int ActivationPrecision,      //Output data bitwidth
                unsigned int ActivationType=0,
                template<int> class type_input  = ap_uint       // For first layer use int value
>
void ConvolutionalLayer_Same_Batch(stream<ap_uint<IFMChannels * Input_precision> > & in,
```

- **Layer-specific configuration values**
  - Support to multiple padding, in this case same
- **Implementation-specific parallelism values**
  - Folding factors
- **Precision configuration values**
  - Independent precision for input/output activations and weights and signed/unsigned math

Backend Stage - Hardware/ Runtime Generation

Neural Network Description

```
layer {
  name: "qt_inp"
  type: "Quant"
  bottom: "bn_inp"
  top: "qt_inp"
  quant_param {
    forward_func: "hwgq"
    backward_func: "relu"
    centers: 0.538 centers:
    clip_thr: 1.614
  }
```

Device Specification File

```
{
  "name": "XLNX:KU115",
  "type": "fpga",
  "frequency" : 200,
  "resources": {
    "LUT": 1451000,
    "DSP": 5520,
    "BRAM": 75.9,
    "URAM": 0
  }
}
```

Optimal Folding Factors

FINN

FINN QNN Library

VIVADO HLx Editions

Hardware Generation

```
// definition for the streaming QNN accelerator
void DoCompute(stream<ap_uint<32> > & inStream, stream<
  // variable declarations for internal streams

  stream<ap_uint<128> > FPGABufferLayer_1;
  #pragma HLS stream depth=2 variable=FPGABufferLayer_1
  stream<ap_uint<128> > FPGABufferLayer_3;
  #pragma HLS stream depth=2 variable=FPGABufferLayer_3
  stream<ap_uint<128> > FPGABufferLayer_5;
  #pragma HLS stream depth=2 variable=FPGABufferLayer_5
  // streaming compute engines for each layer
#pragma HLS DATAFLOW

  MatrixVector_Precision_Batch<16, 64, 1, 64, 784, 256,
ferLayer_1, weights_FPGABipolarMatrixThresholdLayer_0,
  MatrixVector_Precision_Batch<64, 64, 1, 64, 256, 256,
 FPGABufferLayer_3, weights_FPGABipolarMatrixThresholdL
```

Hardware Generation – Network Dataflow Example

- top.cpp
  - Sequence of layers, 1:1 with network topology

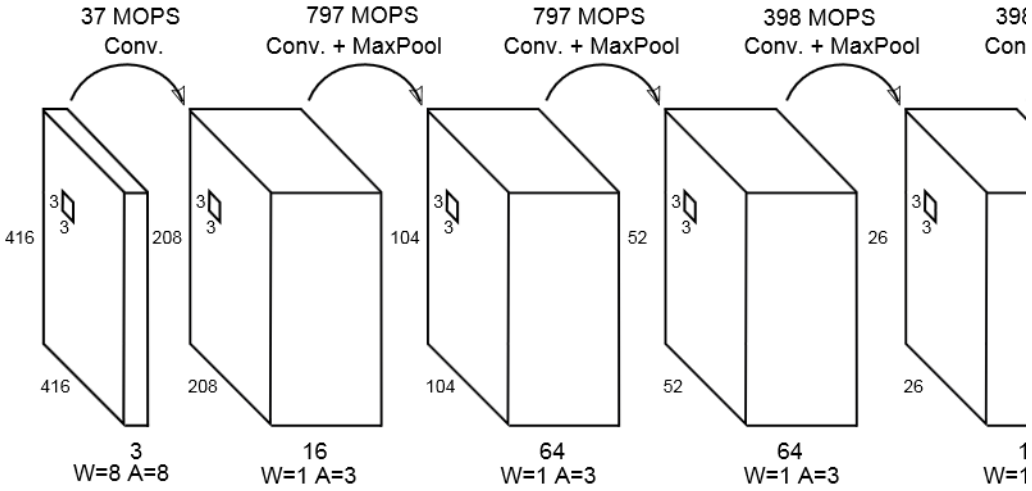
```
StreamingFxdConvLayer_Batch<L0_K, L0_IFM_CH, L0_IFM_DIM, L0_OFM_CH, L0_OFM_DIM, 8, 1, L0_SIMD, L0_PE, 24, 16, L0_WMEM,  L0_TMEM>(i
StreamingConvLayer_Batch<L1_K, L1_IFM_CH, L1_IFM_DIM, L1_OFM_CH, L1_OFM_DIM, L1_SIMD, L1_PE, 16, L1_WMEM, L1_TMEM>(inter1, inter2,
StreamingMaxPool_Batch<L1_OFM_DIM, 2, L1_OFM_CH>(inter2, inter3, numReps);
StreamingConvLayer_Batch<L2_K, L2_IFM_CH, L2_IFM_DIM, L2_OFM_CH, L2_OFM_DIM, L2_SIMD, L2_PE, 16, L2_WMEM, L2_TMEM>(inter3, inter4,
StreamingConvLayer_Batch<L3_K, L3_IFM_CH, L3_IFM_DIM, L3_OFM_CH, L3_OFM_DIM, L3_SIMD, L3_PE, 16, L3_WMEM, L3_TMEM>(inter4, inter5,
StreamingMaxPool_Batch<L3_OFM_DIM, 2, L3_OFM_CH>(inter5, inter6, numReps);
StreamingConvLayer_Batch<L4_K, L4_IFM_CH, L4_IFM_DIM, L4_OFM_CH, L4_OFM_DIM, L4_SIMD, L4_PE, 16, L4_WMEM, L4_TMEM>(inter6, inter7,
StreamingConvLayer_Batch<L5_K, L5_IFM_CH, L5_IFM_DIM, L5_OFM_CH, L5_OFM_DIM, L5_SIMD, L5_PE, 16, L5_WMEM, L5_TMEM>(inter7, inter8,
// fully connected layers
StreamingFCLayer_Batch<256, 64, L6_SIMD, L6_PE, 16, L6_MW, L6_MH, L6_WMEM, L6_TMEM>(inter8, inter9, weightMem6, thresMem6, numReps);
StreamingFCLayer_Batch<64, 64, L7_SIMD, L7_PE, 16, L7_MW, L7_MH, L7_WMEM, L7_TMEM>(inter9, inter10, weightMem7, thresMem7, numReps);

StreamingFCLayer_NoActivation_Batch<64, 64, L8_SIMD, L8_PE, 16, L8_MW, L8_MH, L8_WMEM>(inter10, memOutStrm, weightMem8, numReps);
```

- config.h
  - Finn-generated configuration, with network configuration values + parallelism-specific values
- (possible) params.h
  - Finn-generated weights values to be hardcoded in the bitstream

```
// layer 0 (conv)
// layer config
/*
Using peCount = 16 simdCount = 3 for engine 0
Extracting conv-BN complex, OFM=64 IFM=3 k=3
Layer 0: 64 x 27
WMem = 36 TMem = 4
*/
#define L0_K  3
#define L0_IFM_CH  3
#define L0_IFM_DIM  32
#define L0_OFM_CH 64
#define L0_OFM_DIM  30
// hardware config
#define L0_SIMD 3
#define L0_PE   16
#define L0_WMEM 36
#define L0_TMEM 4
// layer 1 (conv)
// layer config
/*
Using peCount = 32 simdCount = 32 for engine 1
Extracting conv-BN complex, OFM=64 IFM=64 k=3
Layer 1: 64 x 576
WMem = 36 TMem = 2
*/
#define L1_K  3
#define L1_IFM_CH  64
#define L1_IFM_DIM  30
#define L1_OFM_CH 64
#define L1_OFM_DIM 28
// hardware config
#define L1_SIMD 32
#define L1_PE   32
#define L1_WMEM 36
#define L1_TMEM 2
```

Scaling Parallelism



- **For each layer, set all SIMD, PE to 1**
  - Single MAC
- **Until hardware no longer fits on device or FPS target reached**
  - Find slowest layer
    - Increase SIMD to next factor of IFM_CHANS **or**
    - Increase PE to next factor of OFM_CHANS

Goal: Calculate folding factors such that layers produce balanced dataflow

FINN
Performance Results

Network	Platform	Precision (W/A)	Performance (TOPS)
MLP	AWS-F1	1/1	50.8
CNV	AWS-F1	1/1	12.1
Tincy-YOLO	AWS-F1	1/3	5.3
DoReFa-Net/PF	AWS-F1	1/2	11.4

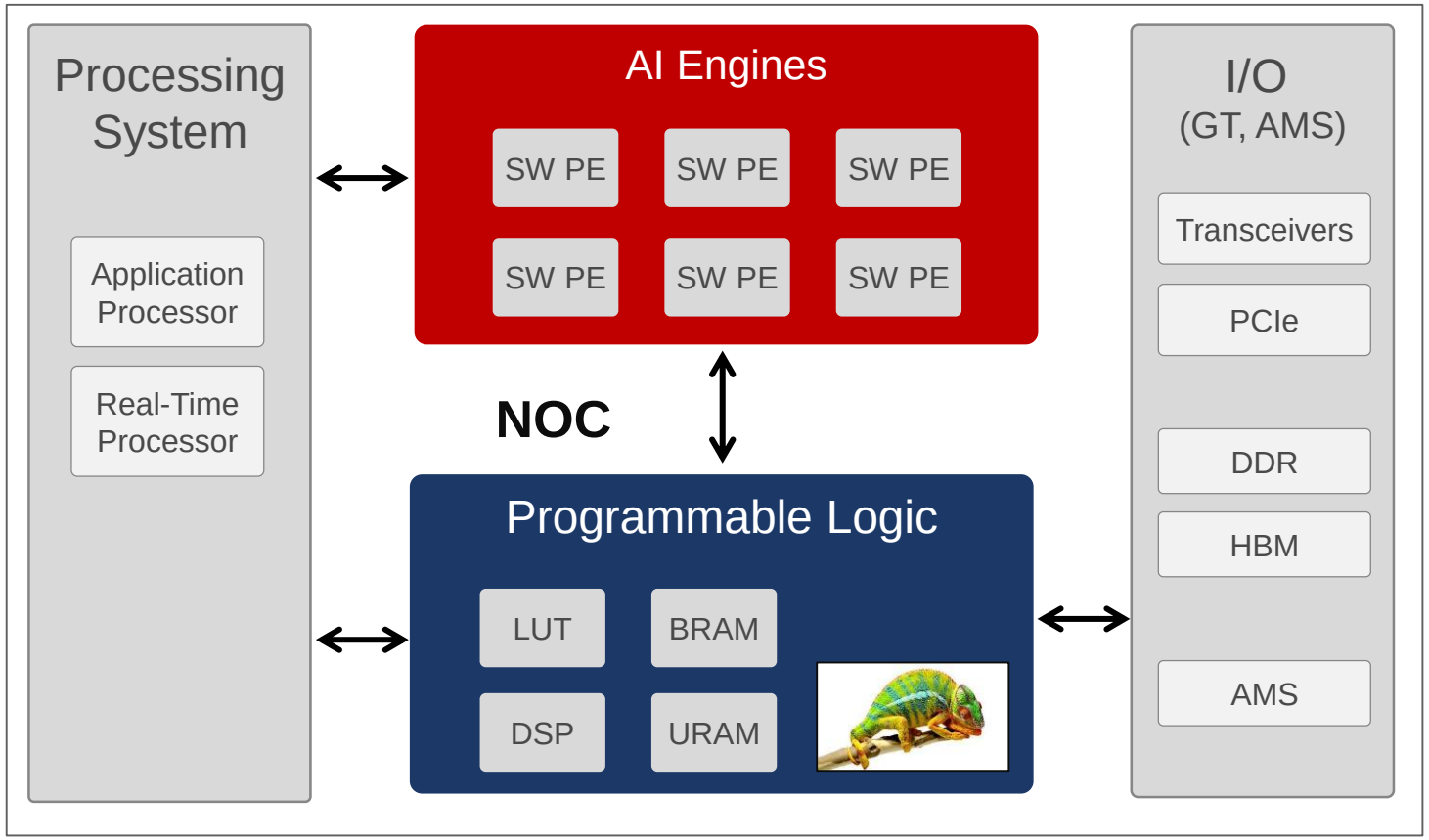
- **Up to 50TOPS measured performance for BNNs**
- **Multiple precision types supported**
  - 8-bit in DSPs, reduced precision in LUTs

Blott, M., et al. "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks"

From FPGAs to ACAPs

XILINX

New Heterogeneous Devices



Up to ~147 TOPS of Int8 performance!

> **From the Xilinx World: Evolution of FPGAs to ACAPs**

Conclusions

- As Moore’s law has ended, heterogeneous accelerated systems have emerged
- High computational demand of machine learning applications is driving hardware development
- Customized dataflow architectures and memory subsystems, custom precisions
  - Dramatic performance scaling and energy efficiency benefits
    - Target Datacenter or Embedded devices
  - Enabling new exciting trade-offs within the design space
- New ACAP devices with AI engines

Thanks!

Adaptable.
Intelligent.

XILINX