TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Final Year Project

Deployment Of Deep Neural Networks on FPGAs

Syed Tihaam Ahmad, Abdullah Ashfaq

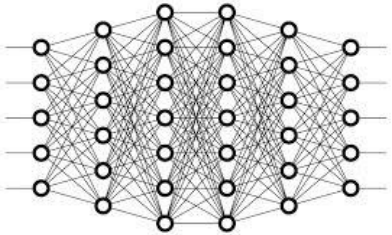
Advisor: Dr. Faisal Shafait
Co Advisor: Dr. Muhammad Shahzad

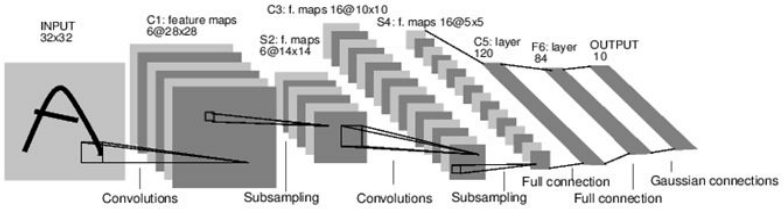
Formalities

	Status
FYP report reviewed by the Advisor	Yes
FYP Report uploaded on PMS/ LMS	Yes
FYP Demo reviewed by the advisor	Yes
FYP Demo uploaded on PMS/LMS	Yes
Course feedback of all courses submitted on CMS	Yes

Motivation

- Deep Neural Networks (e.g. CNNs) are the state-of-the-art algorithms for
  - Image Classification
  - Object Detection
  - Image Segmentation





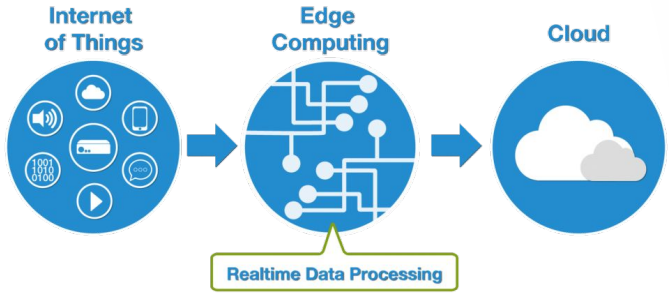
A Full Convolutional Neural Network (LeNet)

Motivation

- DNNs have a very high computational cost
- Require power hungry GPUs (250W)
- Billions of MAC (multiply-accumulate) operations for single inference
- Not feasible for battery dependent applications

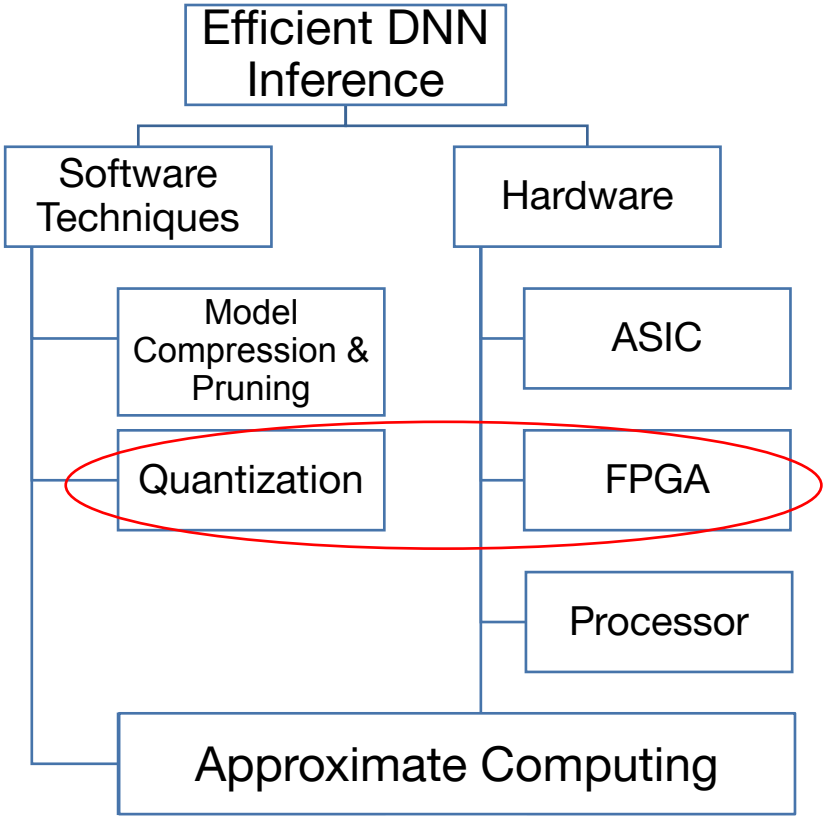
Motivation

- Recent trend towards **Edge** Computing
  - High data volumes and low bandwidth
  - Cloud insufficient on its own



- Machine learning at edge
  - Research towards efficient inference techniques

Overview of Techniques in Literature



Our Choice

- Using FPGAs
  - Flexible
  - Power efficient (< 5W)
  - High throughput
  - Portable

- Xilinx ZC706
  - System on chip: Microprocessor plus FPGA
  - Supports Xilinx HLS Toolchain

Shortcomings

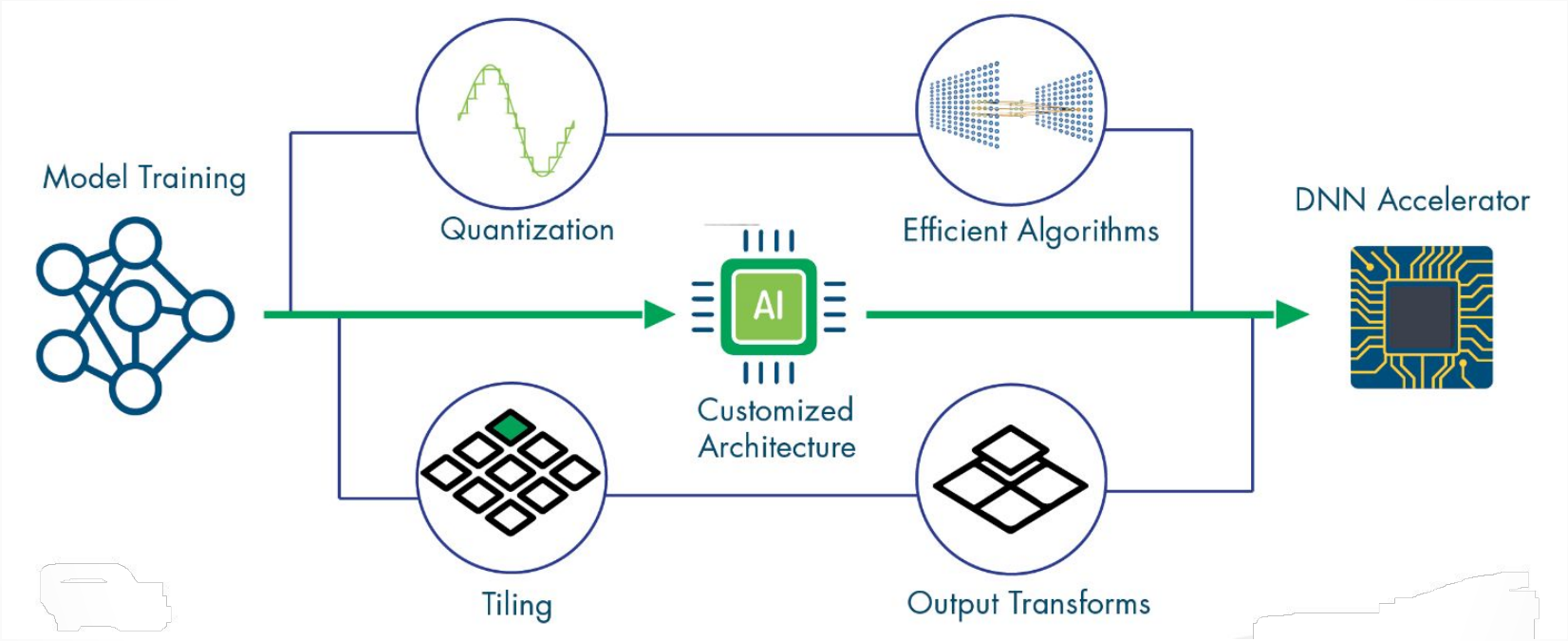
We explored Xilinx's BNN and QNN and deduced the following shortcomings :

- Time-consuming training
- Failure of BNN when deployed for larger models
- Loss of accuracy
- QNN decreases FPS

Our Aim

- To provide generic and versatile building blocks in HLS
  - For variety of networks
  - Complete workflow from training to inference
  - Target Xilinx Zynq FPGAs

Methodology

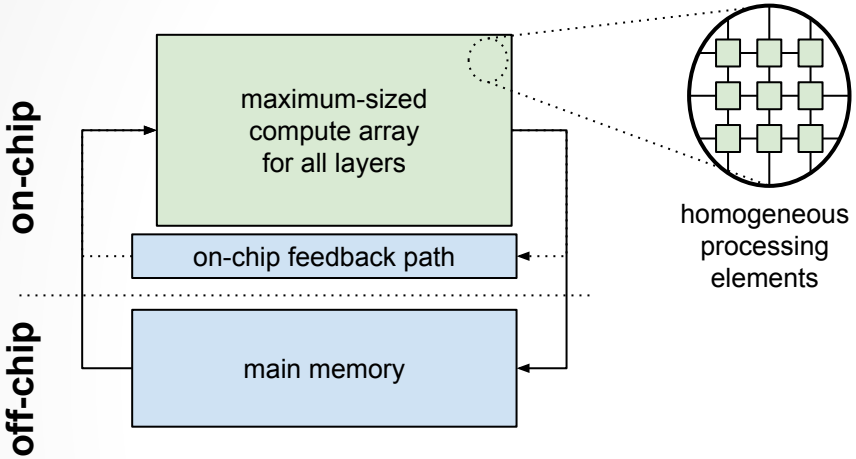


ZYNQ VIVADO

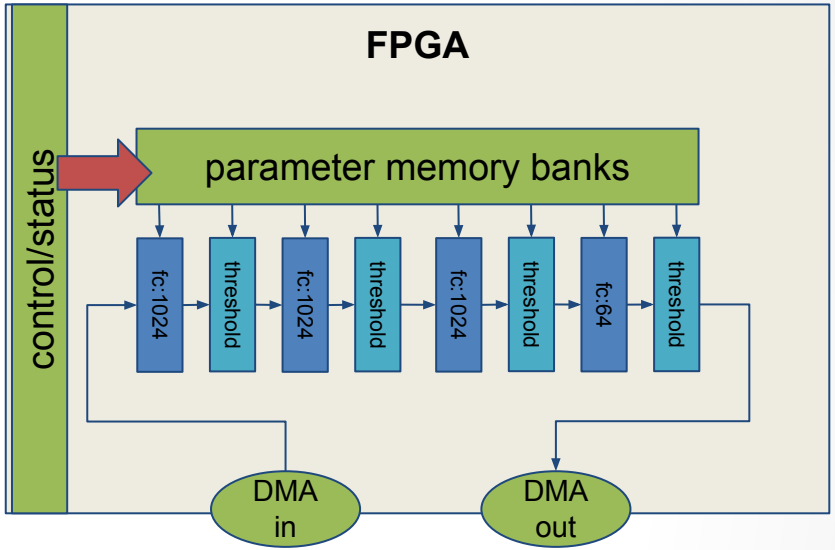
Algorithm

Hardware Architectures

Multi-layer offload



Dataflow architecture



Our Preference: <u>Dataflow</u>

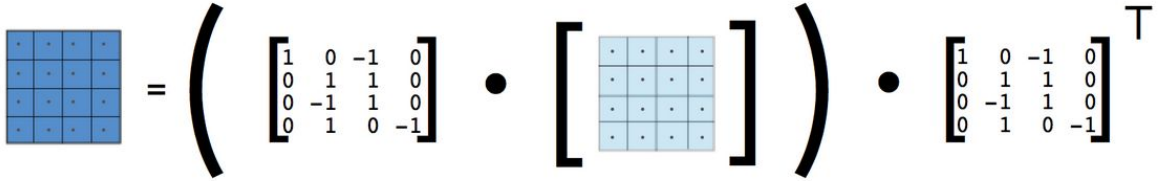
- Avoid “one-size-fits-all” penalties
- **Streaming:** Maximize throughput, minimize latency

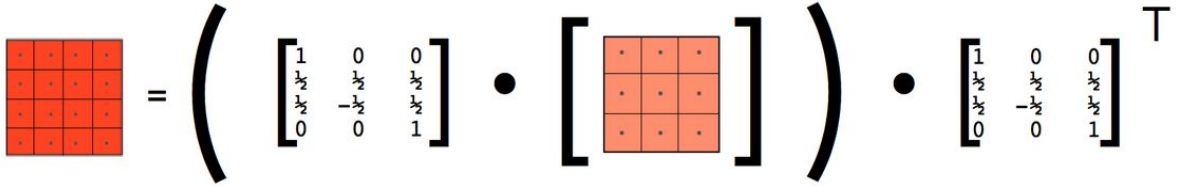
Novelty

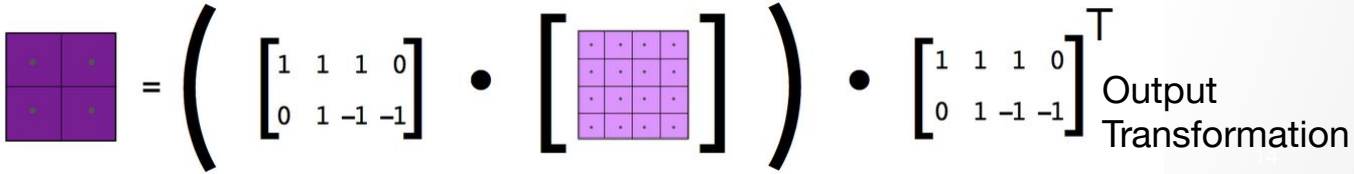
- Quantized 8 bits
  - Accuracy & Throughput trade-off
- Winograd Convolution
  - 2.25x speed up
  - 16 macs instead of 36 macs (4x4 I,3x3 K)

Winograd Convolution

- 2.25x speedup
- Less no of macs used
- Pre-trained weights are transformed



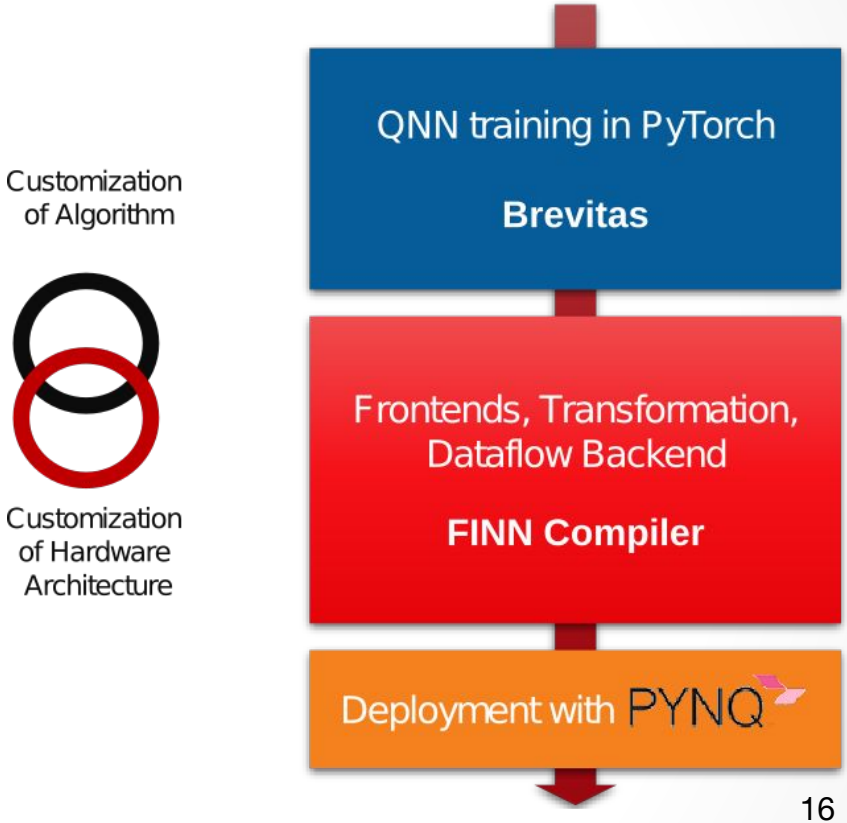




Software Implementation

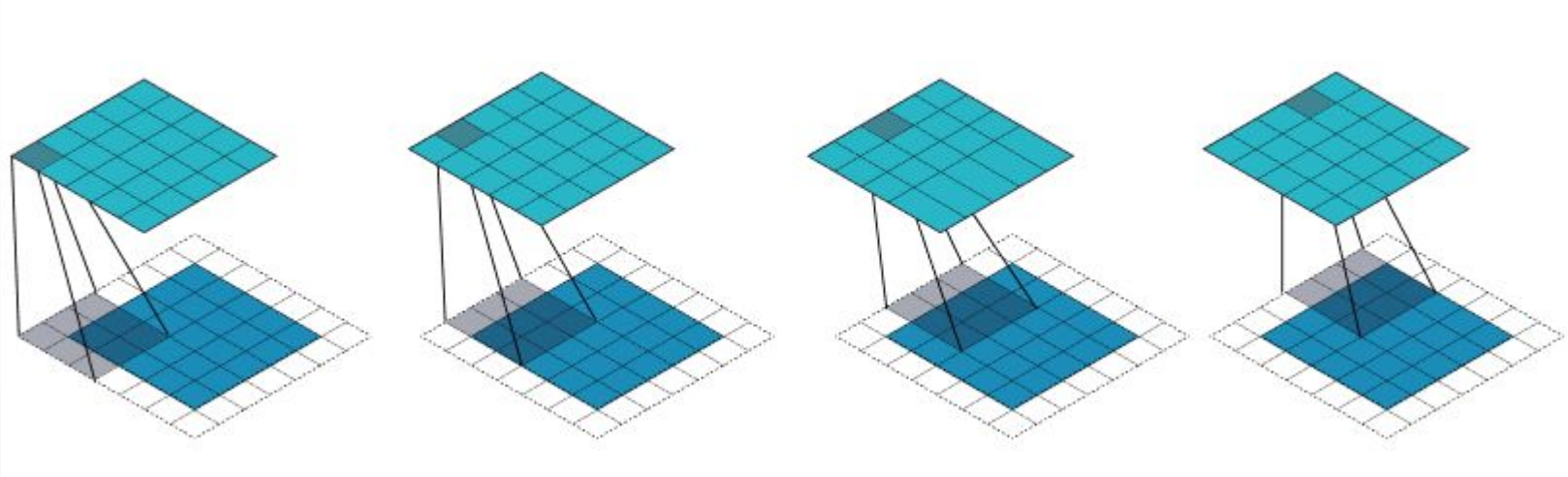
Quantization-Aware Training

- Use of Brevitas framework
- Pytorch based
- Allows flexible no. of bits
- 8-bit training



Pytorch Implementation

- Cross checking
- Layer by Layer confirmation
- Understand ‘quantization’ in brevitas
- Makes easier to write HLS C code
- Uses Conventional conv2d



Intel Nervana Neon Testing

- Python based framework
- Final Software Testing of Brevitas weights
- Inference
- Uses Winograd Convolution

Hardware Implementation

Stage 1: C++ design with Float Ops

```
//CONVOLUTION LAYER
convolution_layer<data,data,data,C1_IFM_ch, C1_IFM_dim, C1_OFM_dim, K_dim, C1_num_filters> C1 (conv1_w_wino);
C1.forward_multi(input,1); //conv forward run

//RELU
Relu<data,C1_OFM_dim*C1_OFM_dim*C1_num_filters> (C1.act,C1.act);

//MAX_POOL
pooling_layer<data,data,28,2,6> m1 ;
m1.maxpool_multi(C1.act, max_output);

//ZERO_PAD
zero_pad_stream<data, 14,14,1,6> pd2;
pd2.zero_pad_multi(max_output, paddedinput2);

//FULLY CONNECTED LAYER
fullyconnected<data,data,data,784,120>(max_output,fc1_w,fc1_b,fc1output);
Relu<data,120> (fc1output,fc1output);
```

Stage 2: Non-Dataflow Design with Fixed Ops

```
//CONVOLUTION LAYER
convolution_layer<data,data,data,C1_IFM_ch, C1_IFM_dim, C1_OFM_dim, K_dim, C1_num_filters> C1 (conv1_w_wino);
C1.forward_multi(input,1); //conv forward run

//RELU
Relu<data,C1_OFM_dim*C1_OFM_dim*C1_num_filters> (C1.act,C1.act);

//MAX_POOL
pooling_layer<data,data,28,2,6> m1 ;
m1.maxpool_multi(C1.act, max_output);

//ZERO_PAD
zero_pad_stream<data, 14,14,1,6> pd2;
pd2.zero_pad_multi(max_output, paddedinput2);

//FULLY CONNECTED LAYER
fullyconnected<data,data,data,784,120>(max_output,fc1_w,fc1_b,fc1output);
Relu<data,120> (fc1output,fc1output);
```

Pros

- Added input, output and bias shifts.
- Got same result as floating-point.

Cons

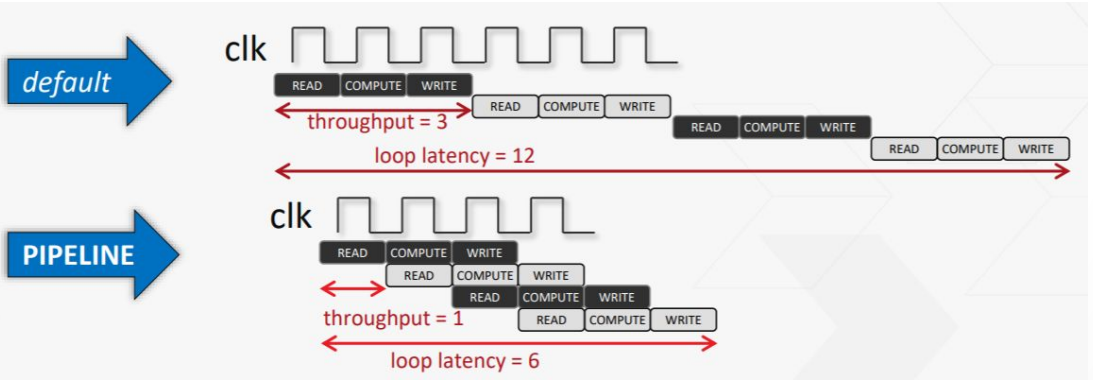
- Non-parameterizable engine.
- No task level parallelism.

Stage 3: Dataflow Design with Int8 Ops

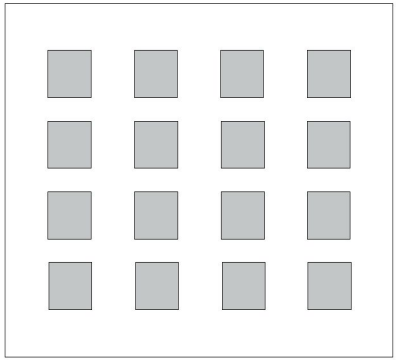
Features

- Custom DMA
- Task Level Parallelism
- Instruction Level Parallelism
- HLS Streaming Interface

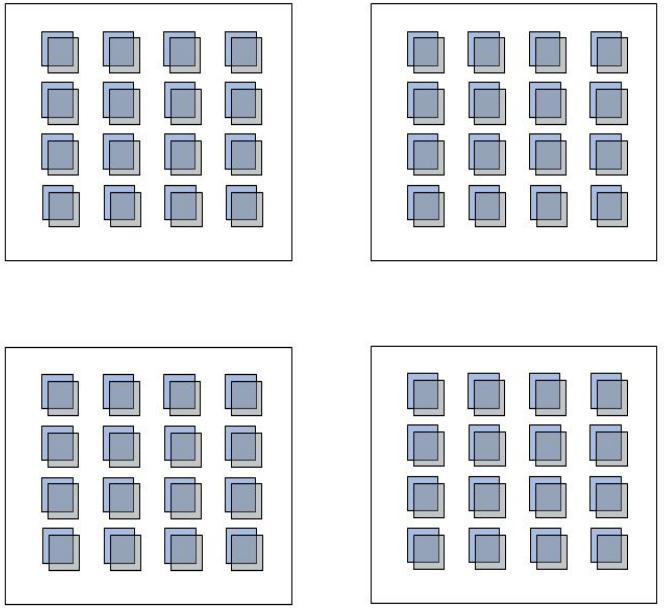
```
void F (...)  {
...
add: for (i=0;i<=3;i++) {
# PRAGMA HLS PIPELINE
      op_READ;
      op_COMPUTE;
      op_WRITE;
}
...
```



Parameterizable Compute Engine

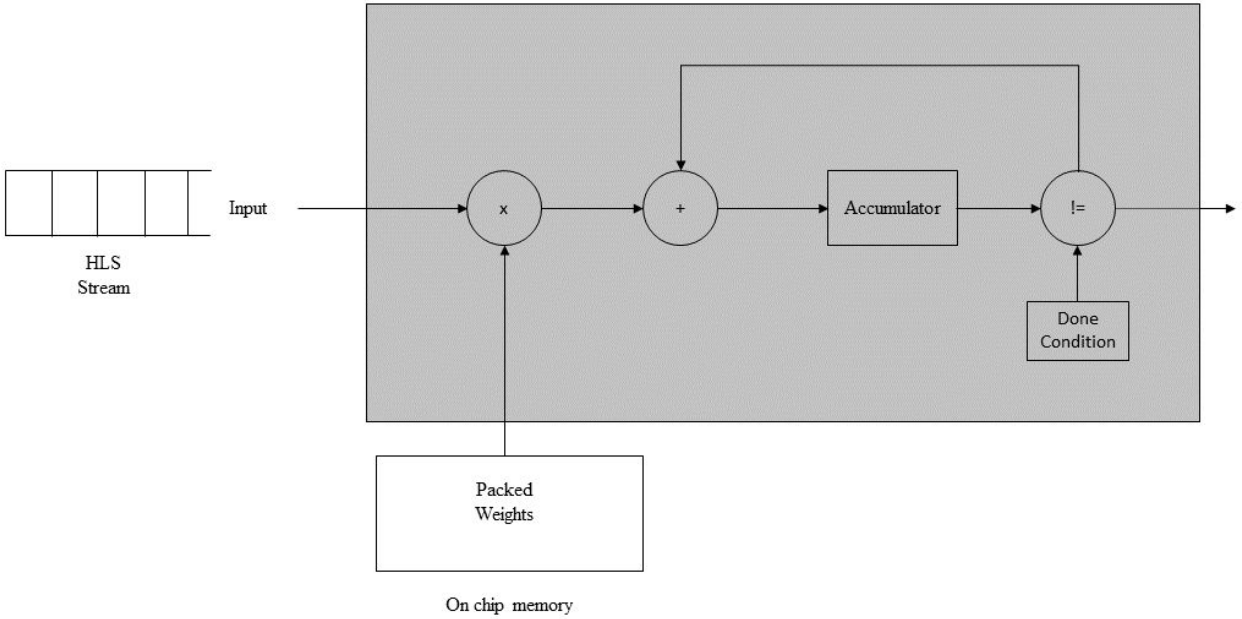


Basic Compute Engine



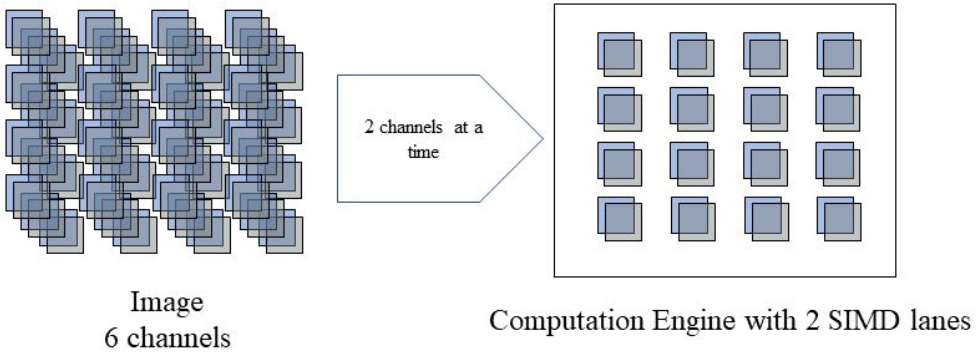
With 2 SIMD Lanes and Computes 4 Outputs in Parallel

Parameterizable Compute Engine: Going Deeper

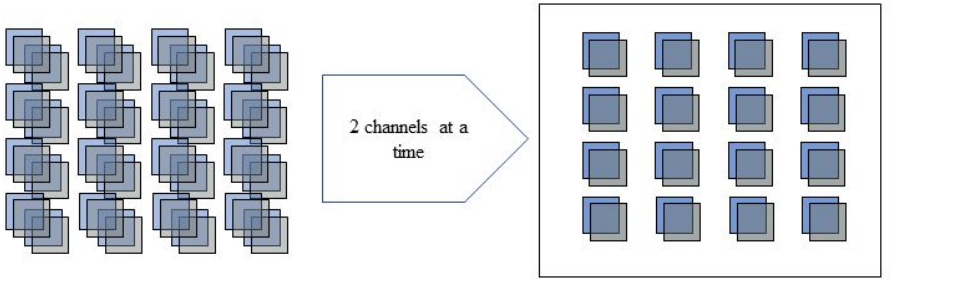


Basic PE Design

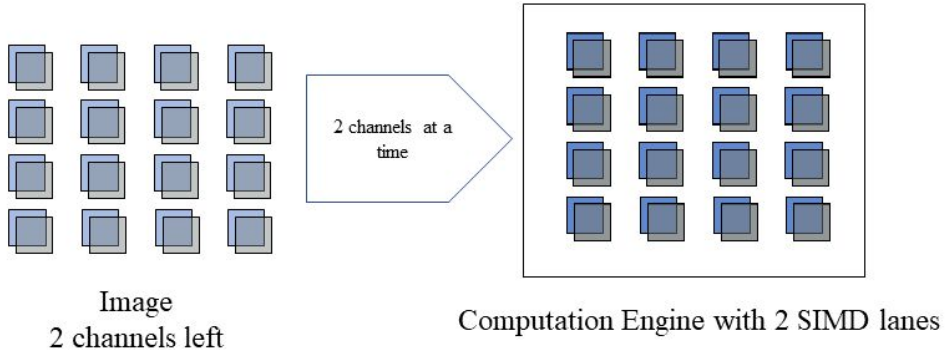
Parameterizable Compute Engine: Dataflow pt.1



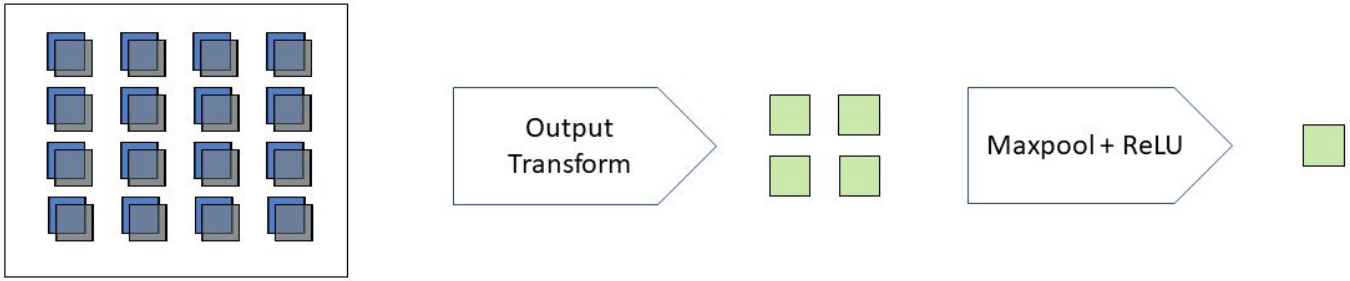
Parameterizable Compute Engine: Dataflow pt.2



Parameterizable Compute Engine: Dataflow pt.3



Parameterizable Compute Engine: Dataflow pt.4



Computation Engine with 2 SIMD lanes

Results: At Mid-Defense

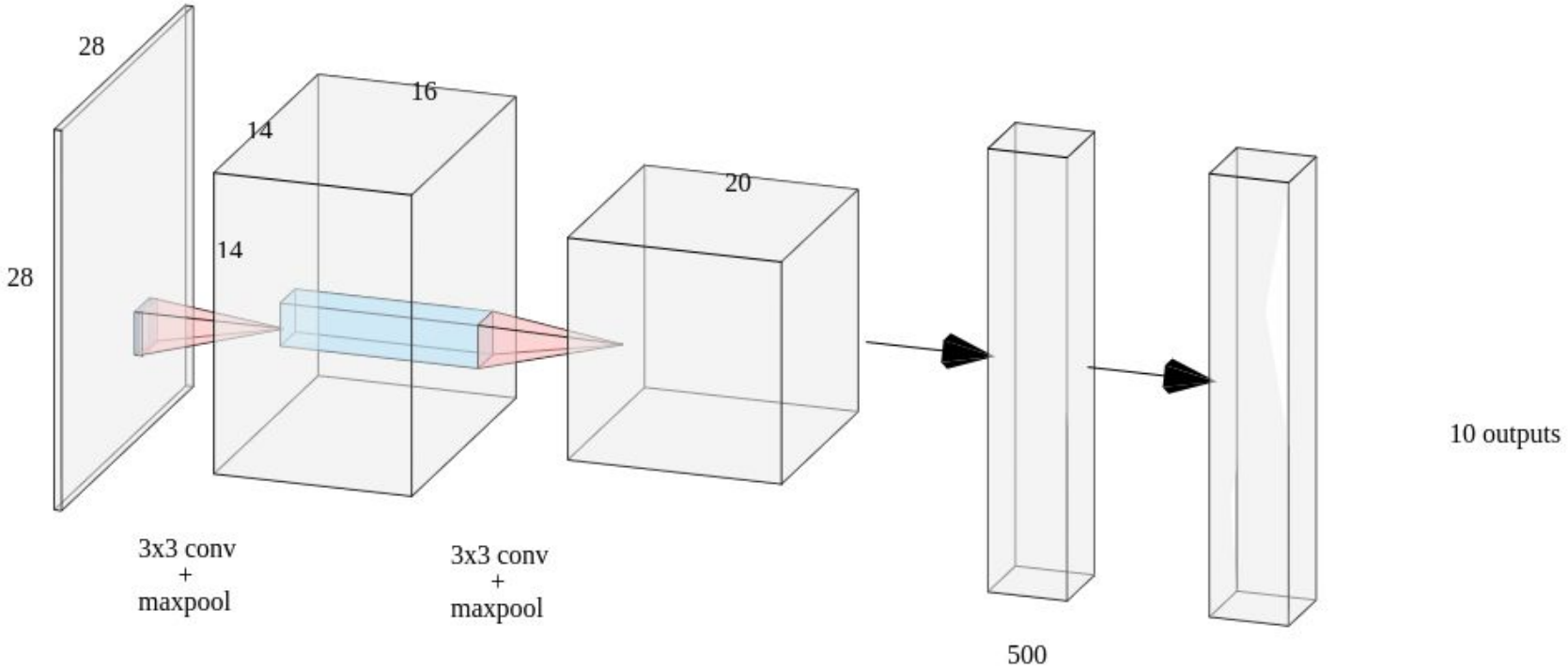
- CHaiDNN: Loopback architecture

Network	**Dataset**	**Accuracy** (Top 1%)	**FPS** (on ZC706)
Googlenet-8bit	Imagenet	67.09 %	8.973396

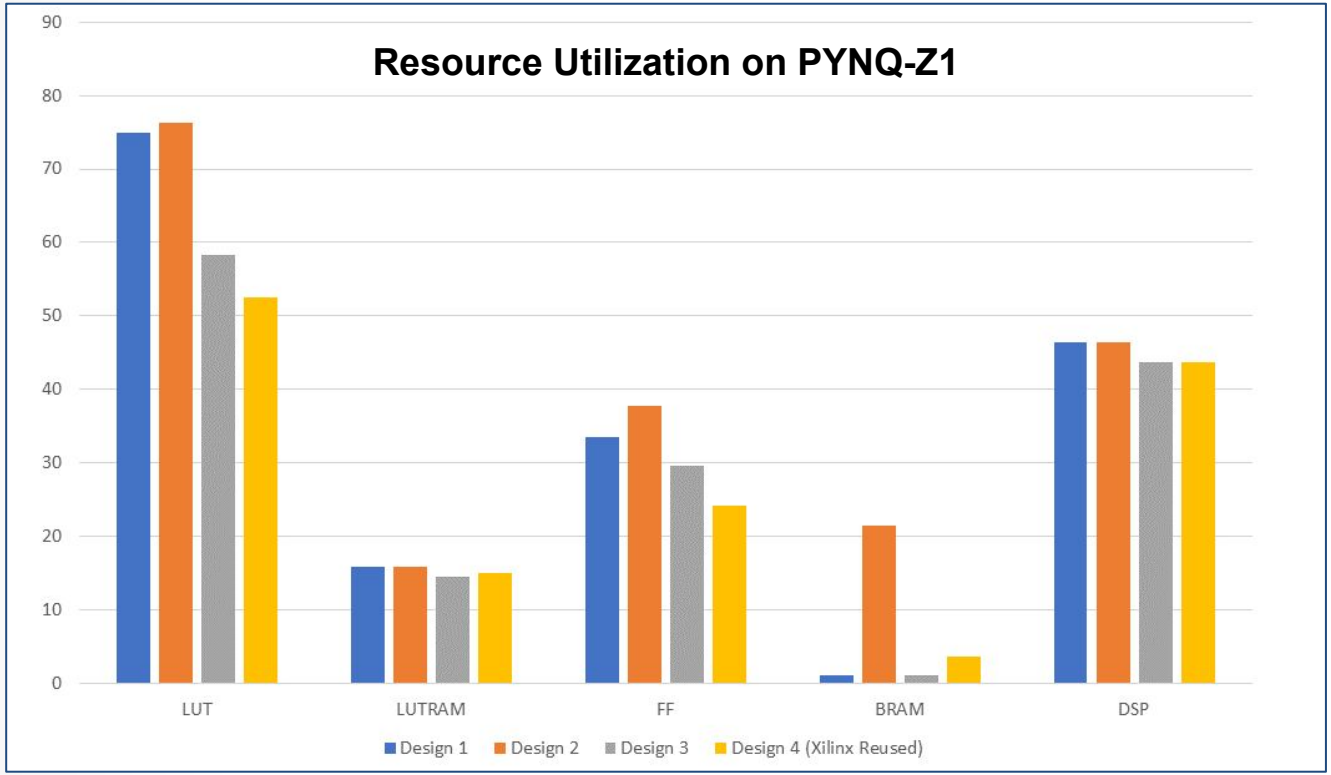
- Our Library: Stage 1

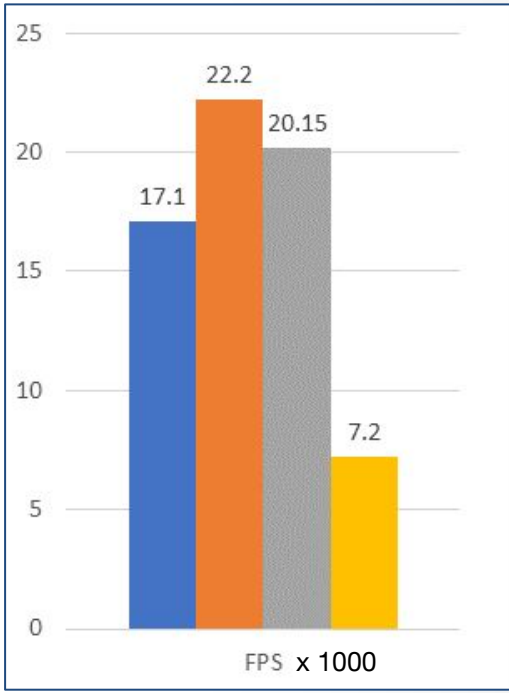
Network	**Dataset**	**Accuracy** (Top 1%)	**FPS** (on ZC706)
Lenet-W8A8	MNIST	98 %	400

Results: Lenet-MNIST Model



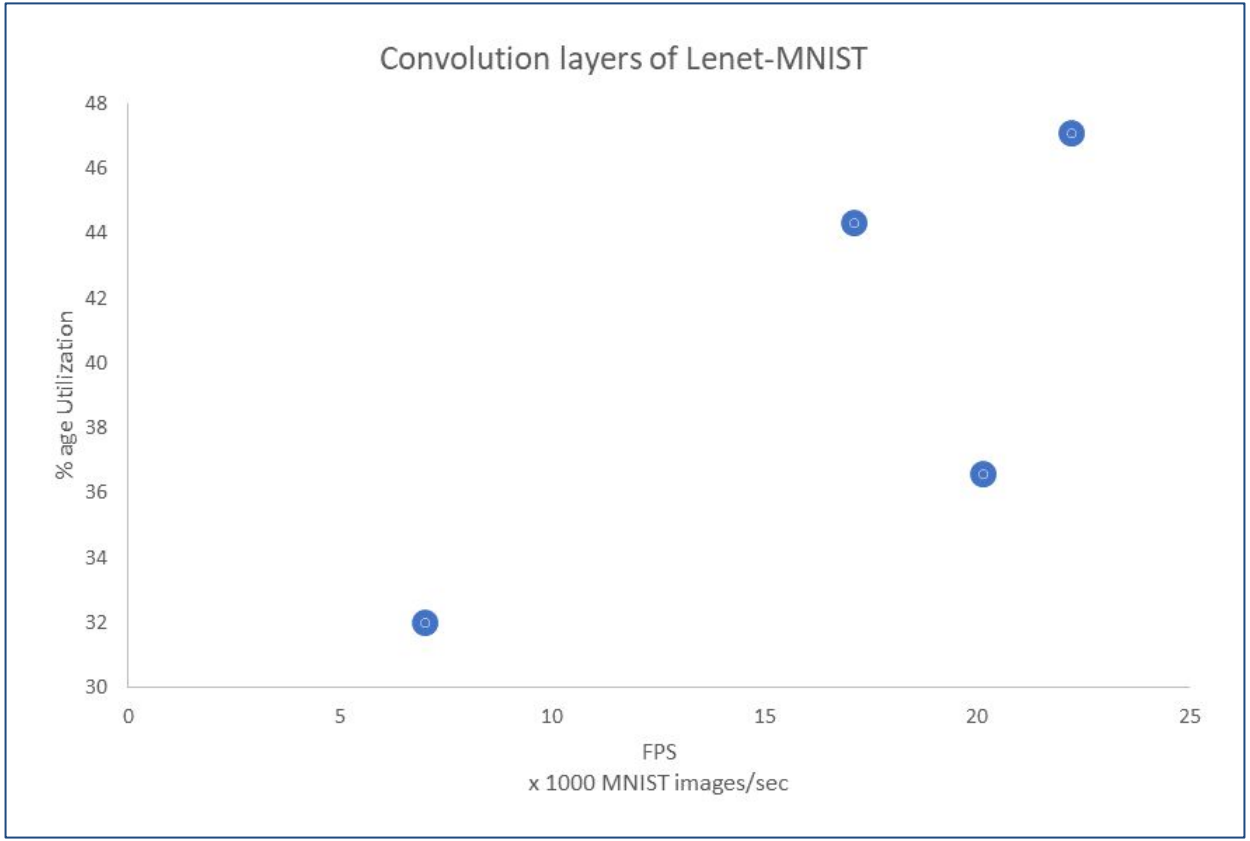
Results: Lenet-MNIST





* For the same SIMD/PE configuration
*100 MHz clock frequency

Results: Lenet-MNIST

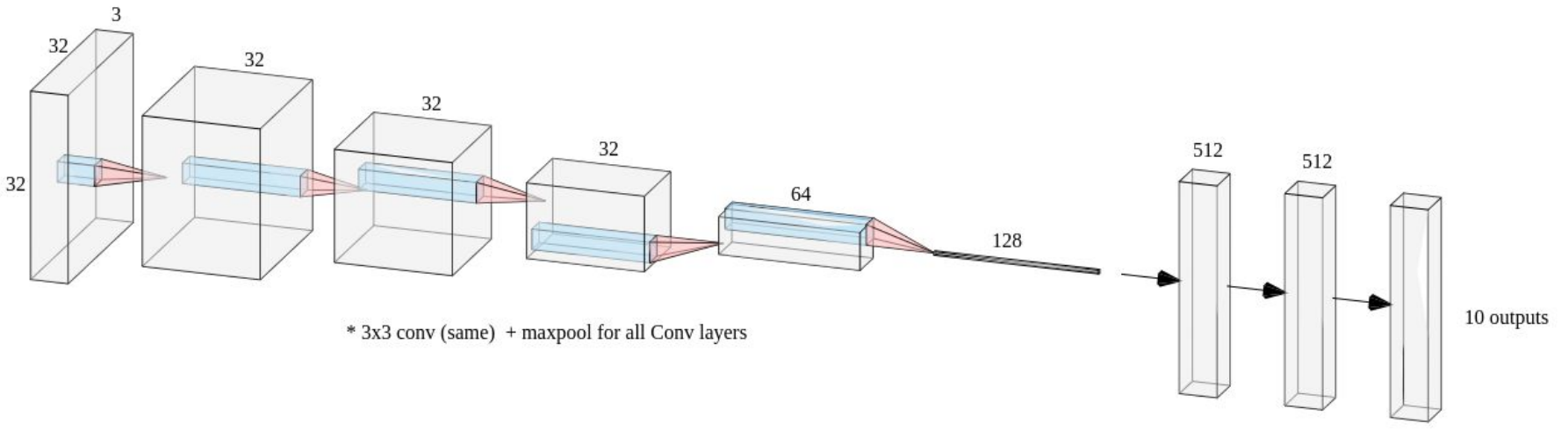


Results: Comparison with Other Platforms

Device	Frequency	FPS	Images/J
Intel i7-7700HQ CPU	1093 MHz	6.6K	-
Tesla K80 (150 W)	562 MHz	3.5M	23.3K
PYNQ-Z1	100 MHz	22.2K	12K

- 3.36x times faster than CPU
- Significantly less FPS than GPU but full potential untapped

Results: Cifar10-CNV Model



Accuracy = 84.1%

Results: Cifar10-CNV

Deployment on PYNQ-Z1

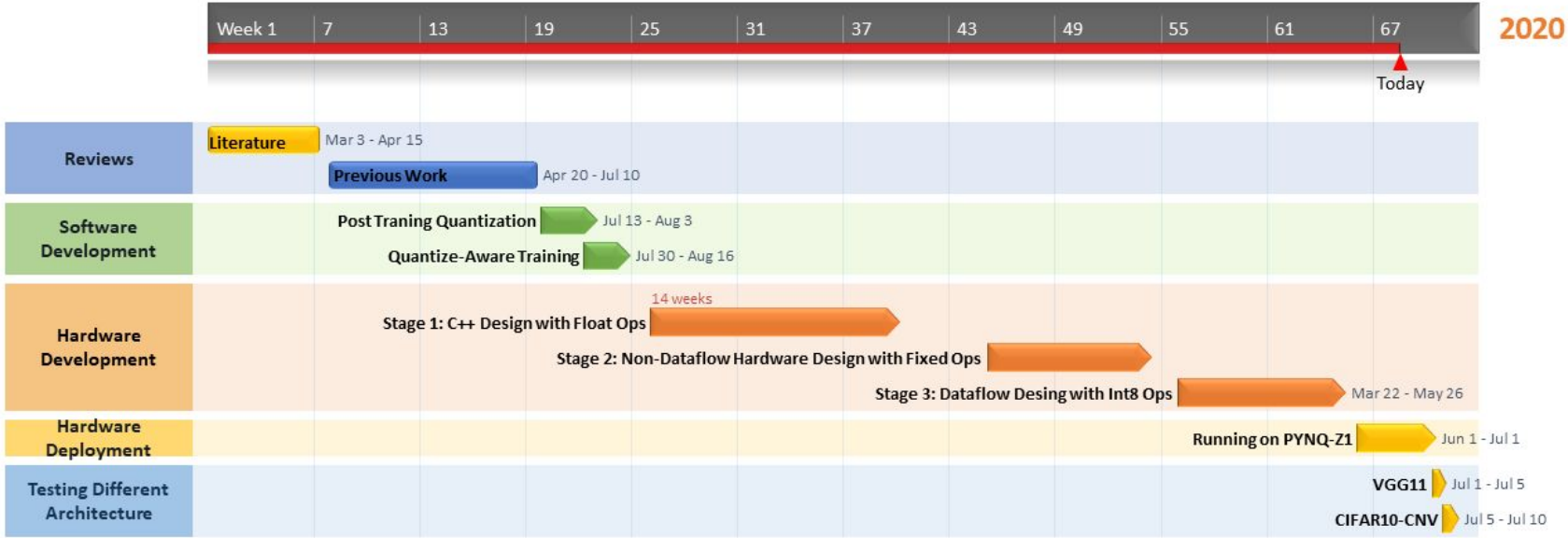
	LUT	LUTRAM	FF	BRAM	DSP	Power (W)	FPS
Design 4 (Xilinx reuse)	53.71%	5.03%	58.98%	85%	42.73%	1.94	573

* For the minimum SIMD/PE configuration
*100 MHz clock frequency

Achievements

- Complete flow from training to inference on hardware
- Hardware works for both Quantized training and Post-training Quantization
- Easy to use dataflow library implementation:
  - Conv layer
  - Maxpool layer
  - Relu activation
  - Batchnorm
- Convolution models deployed using our library on PYNQ-Z1:
  - Lenet - MNIST
  - CNV - CIFAR10

Timeline


Week 1
7
13
19
25
31
37
43
49
55
61
67
2020
Today
Reviews
Literature
Mar 3 - Apr 15
Previous Work
Apr 20 - Jul 10
Software Development
Post Traning Quantization
Jul 13 - Aug 3
Quantize-Aware Training
Jul 30 - Aug 16
Hardware Development
14 weeks
Stage 1: C++ Design with Float Ops
Stage 2: Non-Dataflow Hardware Design with Fixed Ops
Stage 3: Dataflow Desing with Int8 Ops
Mar 22 - May 26
Hardware Deployment
Running on PYNQ-Z1
Jun 1 - Jul 1
Testing Different Architecture
VGG11
Jul 1 - Jul 5
CIFAR10-CNV
Jul 5 - Jul 10

Work Division

Syed Tihaam Ahmad	Abdullah Ashfaq
• Quantization-aware training • Defining library structure • Winograd Convolution Implementation • Relu optimization • Maxpool optimization • FC layer optimization • Batch Norm optimization • Tiling • Streaming sliding window design • Testing and Debugging	• Post-training quantization • Explored open-source libraries e.g. ChaiDNN • Ported CHaiDNN to ZC706 • Conv, ReLU, Maxpool implementation • FC layer implementation • PS code • Streaming architecture design • Running on Hardware

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Thank You

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Thank You
Any Questions?